International Workshop on

Parsing Technologies

28-31 August 1989 • Pittsburgh, PA

Carnegie Mellon

International Workshop on Parsing Technologies

28-31 August 1989

Carnegie Mellon University

Pittsburgh, Pennsylvania - USA

Holiday Inn - Oakland, CMU, Hidden Valley - Somerset

QA
76.9
.N38
I58
1989

006.35
I612p
1989

Bq

Preface

WELCOME to the International Workshop on Parsing Technologies.

The interest and the progress being made in the field of parsing is exciting. The technical program has been assembled to include all aspects of this technology. We hope it will stimulate further discussion, research and development in the field.

We hope the emphasis of this workshop will center around the exchange of ideas rather than the presentation of results. In this workshop, presenters should be prepared to discuss problems and techniques - with a particular emphasis on work-in-progress and unresolved difficulties.

The program committee has selected a variety of areas for discussion, hoping that you will choose the set of presentations that best match your interests and specialties. The variety should encourage discussion and exchange that should benefit all. We hope to encourage participation, discussion and even argument.

The workshop program and organization have been created through the efforts of a large number of people who have given generously of their time and talent.

I would like to thank each of the comittee members: Bob Berwick, Harry Bunt, Jaime Carbonell, Eva Hajicova, Aravind Joshi, Ron Kaplan, Bob Kasper, Martin Kay, Match Marcus, Makoto Nagao and Yorick Wilks.

In addition, I would like to acknowledge contributions and extend my gratitude to the local arrangement people. Especially Joan Maddamma, the workshop secretary, who did most of the administrative work for this volume and the workshop.

Masaru Tomita
Workshop Chairman
Carnegie Mellon University

Workshop Committee

Workshop Chairman:

Masaru Tomita, Carnegie Mellon University

Program Committee:

Robert Berwick, Massachusetts Institute of Technology
Harry Bunt, Tilburg University
Jaime Carbonell, Carnegie Mellon University
Eva Hajicova, Charles University
Aravind Joshi, University of Pennsylvania
Ronald Kaplan, Xerox PARC
Robert Kasper, University of Southern California
Martin Kay, Xerox PARC
Mitch Marcus, University of Pennsylvania
Makoto Nagao, Kyoto University
Yorick Wilks, New Mexico State University

For Your Information

For additional copies of this book, write:

Carnegie Mellon University
Attn: Joan Maddamma
School of Computer Science
Pittsburgh, PA 15213-3890 USA

PRICE: $50.00 within USA - $55.00 for overseas mail

TABLE OF CONTENTS

International Workshop on Parsing Technologies

28-31 August 1989

Carnegie Mellon University

Pittsburgh, Pennsylvania - USA

Holiday Inn - Oakland, CMU, Hidden Valley - Somerset

UNIFICATION AND CLASSIFICATION: AN EXPERIMENT IN INFORMATION-BASED PARSING

Robert T. Kasper
USC/Information Sciences Institute
Admiralty Way, Suite 1001
Marina del Rey, CA 90292

When dealing with a phenomenon as vast and complex as natural language, an experimental approach is often the best way to discover new computational methods and determine their usefulness. The experimental process includes designing and selecting new experiments, carrying out the experiments, and evaluating the experiments. Most conference presentations are about finished experiments, completed theoretical results, or the evaluation of systems already in use. In this workshop setting, I would like to depart from this tendency to discuss some experiments that we are beginning to perform, and the reasons for investigating a particular approach to parsing. This approach builds on recent work in unification-based parsing and classification-based knowledge representation, developing an architecture that brings together the capabilities of these related frameworks.

1. Background: Two General Frameworks for Representing Information

1.1. Unification-based Grammars

A variety of current approaches to parsing in computational linguistics emphasize declarative representations of grammar with logical constraints stated in terms of feature and category structures. These approaches have collectively become known as the "unification-based" grammars, because unification is commonly used as the primary operation for building and combining feature structures. Some of the simplest of these grammatical frameworks, as exemplified by the PATR-II system [Shieber 1984], state constraints on features entirely in terms of sets of unifications that must be simultaneously satisfied whenever a grammatical rule is used. In such systems all constraints on a rule or lexical item are interpreted conjunctively. Many of the more recent frameworks also use other general logical connectives, such as disjunction, negation and implication, in their representation of constraints. The usefulness of such logical constraints is abundantly illustrated by linguistic models, including Systemic Grammar (SG) [Halliday 1976] and Head Driven Phrase Structure Grammar (HPSG) [Pollard&Sag 1987], and by computational tools such as Functional Unification Grammar (FUG) [Kay 1985]. For example, SG and FUG even use disjunctive alternations of features, instead of structural rules, as the primary units of grammatical organization. While the intuitive interpretation of these logical constraints is rather straightforward, and they are quite natural for linguists to formulate, large-scale implementations of them have typically involved finding a delicate balance between expressive power and computational efficiency.

Some difficulties can be expected in developing a system for computing with disjunctive and negative feature constraints, because it has been established that common operations on such descriptions, such as unification and subsumption, are NP-complete and require exponential time in the worst case. The most common and obvious way to deal with disjunctive constraints is to expand the grammatical description to disjunctive normal form (DNF) during a pre-processing step, thereby eliminating disjunction from the rules that are actually used by the parser. This method works reasonably well for small grammars, but it is clearly unsatisfactory for larger grammars, because it actually requires exponential space and time in all cases. For even modest amounts of disjunction, the parser is forced to operate on a huge description, even in many cases where no exponential expansion would be necessary.

It is possible to avoid exponential expansion for most practical grammars, and several unification algorithms for disjunctive feature descriptions have been developed in recent years. The first of these algorithms was developed by Karttunen [Karttunen 1984]. His method of representing disjunction allowed value disjunction (i.e. alternative values of a single feature), but it did not allow general disjunction (i.e. constraints involving multiple features). Although it is possible to transform any description that contains general disjunction into a formally equivalent description that contains only value disjunction, this transformation may sometimes result in loss of efficiency or lack of clarity in the structures produced by a parser.

Two more recent algorithms [Kasper 1987, Eisele&Doerre 1988] allow general disjunctive descriptions, and avoid expansion to DNF by exploiting logical equivalences between descriptions to produce normal forms that allow a more compact representation. Kasper's algorithm is based on a normal form that divides each description into definite and indefinite components. The definite component contains no disjunction, and the indefinite component contains a list of disjunctions that must be satisfied. The Eisele&Doerre algorithm uses a different normal form that guarantees the detection of any inconsistencies during the normalization process by selectively expanding disjunctions that might possibly interact with other information in the description. Although a precise characterization of the differences in performance between these algorithms involves many subtleties, the Eisele&Doerre algorithm usually handles value disjunction more efficiently, and the Kasper algorithm usually handles general disjunction more efficiently. The crucial technique shared by both algorithms is the use of a normal form that allows early elimination of alternatives when they are inconsistent with definite information.

The Kasper algorithm was first implemented as an extension to the unification algorithm of the PATR-II parser, and it has been further developed to handle conditional descriptions and a limited type of negation [Kasper 1988a]. These extensions to PATR-II have been used to construct an experimental parser for systemic grammars [Kasper 1988b], which has been tested with a large grammar of English.

Although these methods for processing complex feature constraints are generally much more efficient than expansion to DNF, they still have several significant sources of inefficiency:

1. a large amount of structure must be copied in order to guarantee correct unification;

2. consistency checks are required between components of a description that do not share any features in common, because unification cannot determine whether any dependencies exist between two structures without actually unifying them;

3. repeated computations are often required over sub-expressions of descriptions, because the results of prior consistency checks are not saved.

These sources of inefficiency are not unique to one method of parsing with disjunctive descriptions; similar shortcomings are commonly reported for most unification-based systems. For example, the Eisele&Doerre algorithm eliminates some redundant consistency checks, but it generally requires copying significant portions of a description to do so. The unification literature contains several techniques for reducing the amount of copying by structure sharing, but these techniques appear to solve only part of the problem. A more general approach to improving the efficiency of unification may be available by adopting methods that are used in classification-based systems.

1.2. Classification-based Knowledge Representation

The KL-ONE family of knowledge representation systems organize information about objects and the relations between them into conceptual hierarchies (a combination of semantic networks and frames) according to class membership, where X is below Y in the hierarchy if X is a subclass or instance of the class Y. For example, a hierarchy of English word classes would probably contain Verbs, Modal-Verbs as a subclass of Verbs, and the word "should" as an instance of Modal-Verbs. More formally, the hierarchy is a subsumption-ordered lattice based upon logical properties that can be deduced from the definitions of concepts and the facts known about particular objects. In these systems, *classification* is the operation that places a new class or object into the lattice according to the subsumption order. A primary benefit of classification is that it organizes large collections of knowledge in such a way that properties shared in common by many objects only need to be represented once, yet they can still be efficiently accessed.

KL-ONE and similar frameworks have been used for semantic interpretation in some natural language processing systems, but usually in a way that is quite separate from the grammatical parsing process. Recent research indicates that it may be advantageous to make use of a classification-based framework for processing grammatical knowledge as well. Many formal properties are shared by the feature descriptions used in unification-based grammars and the terminological definitions used in KL-ONE. Generally speaking, linguistic categories correspond to concepts, and their features (or attributes) correspond to binary relations in the knowledge representation system. The similarity between these two types of descriptions has been most clearly documented by Smolka [Smolka 1988] in his development of a logic that integrates a significant combination of their expressive capabilities. Smolka has also shown that the subsumption and unification problems for this logic can be reduced to each other in linear time. Thus, systems based on either term subsumption or unification can be expected to solve a similar range of problems, although differing levels of non-asymptotic time/space efficiency can be expected. Theoretical results have also been based on the observation that feature structures can be implicitly organized into a subsumption lattice of types according to their information content. In most unification-based systems the lattice is not explicitly constructed, but a classification-based system can be used to place the feature structures of a grammar and lexicon into a structure-sharing lattice, potentially improving both space and time efficiency.

Despite the underlying similarities between the KL-ONE framework and unification-based grammars, there are significant differences in the expressive capabilities that are usually provided. In particular, the knowledge representation systems typically have general constraints on relations with multiple values, whereas most unification-based systems do not provide a direct representation for features with set values. On the other hand, complex logical constraints involving disjunction and negation have been more extensively developed in unification-based systems than in classification-based systems. The LOOM system [MacGregor 1988], which has been developed at USC/ISI, appears to be the first in the KL-ONE family to have included general disjunction and negation in its concept definition language. The implementation of classification for disjunctive concepts has been based on the same strategy that was originally developed for unification with disjunctive feature descriptions [Kasper 1987]. The implementation of classification for concepts defined by negation is still in progress. With these extensions, the LOOM system should be able to handle the full range of constraints that have been used in linguistic descriptions of feature structures.

2. An Experiment in Classification-based Parsing

In order to explore a strategy for parsing based on classification, our first experiment will be to emulate the unification component of our parser for a large systemic grammar of English [Kasper 1988b] within the framework of LOOM. It appears to be straightforward to convert the feature constraints of the grammar into a set of definitions that can be processed by LOOM, because of the underlying correspondences between LOOM's concept definitions and linguistic feature descriptions that we have already described. It is also straightforward to perform an operation that is equivalent to the unification of feature structures within LOOM. This is accomplished by forming an object which is defined as the conjunction of the objects corresponding to the feature structures.

Motivating this experiment are two primary goals:

1. to investigate the extent to which classification can be used to organize the knowledge contained in linguistic descriptions so that it can be more efficiently accessed during the parsing process;

2. to develop a suitable architecture for integrating semantic information into the parsing process, in a way that knowledge specific to application domains does not have to be re-organized for parsing.

2.1. Efficiency Considerations

The classification-based architecture used by LOOM solves a whole class of related efficiency problems by explicitly constructing and maintaining a subsumption-ordered lattice of terms with inheritance. In particular, it may provide substantial improvements for some of the above mentioned sources of inefficiency that have been observed with unification-based parsers.

2.1.1. Structure Sharing

The organization of objects into a lattice automatically provides a great amount of structure sharing. Pointers are copied instead of structures whenever objects are defined or modified.

In most unification-based parsers, it is necessary to make new copies of the feature structures that are associated with lexical items or grammatical rules whenever they are used in building a description of a sentence (or one of its constituents). In a classification-based system the entire structure does not need to be copied, because the description of a constituent can contain pointers to the classes of objects that it instantiates. This representation not only saves space, but it also allows the parser to make use of information that has already been precomputed (during the classification process) for classes of objects in the grammar and lexicon.

2.2. Integrating Semantic Information Into the Parsing Process

In order for practical natural language parsers to be produced with less effort per application, it is desirable for the knowledge base of an application to also be usable by a general purpose parser. Existing systems often use semantic grammars that are specific to a particular application domain, or require substantial reorganization of the information used by an application so that it can be used by the parser. A more effective use of knowledge sources may be possible if linguistic features and information about an application's semantic domain are defined in the same general knowledge representation framework. Using a classification-based system, links can be established between terms of the semantic domain and terms of the linguistic knowledge base that correspond to them. This approach has already been explored in text generation research [Kasper 1989], where the links are established by stipulating that terms of the application domain specialize one or more terms of the linguistic model. This condition generally holds, because the linguistic model contains primarily abstract features.

Another potential benefit of using an integrated knowledge organization is early disambiguation according to features of the semantic domain. If objects of the semantic domain are directly linked in a knowledge base to lexical or grammatical features, the parser can use information about those objects without any special purpose machinery.

3. Summary

We are developing an experimental parser using the classification-based architecture of the LOOM knowledge representation system. The initial goal is to reproduce the functionality of an existing unification-based parser, using a large grammar of English. If successful, this experiment should enable a comparison of classification and unification as mechanisms for parsing. A classification scheme appears to provide a way of substantially reducing several of the most general sources of inefficiency that are observed in current unification-based parsers. However, this conjecture needs to be examined by performing experiments with several real grammars and applications. Because the classification mechanism is based on general logical properties of feature descriptions, it should be applicable to a broad class of grammars, just as unification-based parsers have been developed for grammars from a diverse range of linguistic theories and applications. In addition to providing an efficient engine for processing the constraints of linguistic feature descriptions, we also expect this type of information organization to provide a strong basis for integrating semantic knowledge and knowledge specific to particular applications into the parsing process.

2.1.2. Indexing Dependencies

The process of classification also keeps track of dependencies between different objects, eliminating the need for checking consistency between components of a description that have no features in common. In effect, an index is incrementally constructed from features to descriptions that contain them.

In most unification-based systems, feature structures are represented by directed graphs or terms. These representations effectively provide an index of features possessed by each object. This type of indexing is generally sufficient if only conjunctive constraints on features are used. When disjunctive constraints are also used, it becomes useful to keep track of dependencies between different parts of a complex description, in order to avoid repeated consistency checks between parts that share no features in common. A reverse index (from features to objects having those features) can be used to avoid these useless consistency checks. This second kind of index is created automatically when feature structures are classified into an explicit lattice.

2.1.3. Avoiding Redundant Computations

The first time that a component of a description is classified, it is placed into a lattice containing all other descriptions in the knowledge base. The lattice structure makes full consistency checks unnecessary between objects that are known to be in a subsumption relationship. The object-oriented representation of the lattice also makes it possible to store the results of consistency checks between components of a description, so that they do not need to be repeated.

2.1.4. Using Classification as a Grammar Compiler

The classification-based architecture is also able to impose a system of type constraints on feature structures. Constraints may be placed on the sets of features that are required or prohibited for particular types of objects, and on the types of objects that may occur as the values of particular features. Structures that violate one of these constraints are automatically marked as incoherent. In contrast, many of the unification methods used in computational linguistics have untyped feature structures. For applications of limited scale, an untyped unification-based system may provide acceptable results with somewhat less overhead than a classification-based approach. In particular, an untyped feature system allows greater flexibility in the early stages of developing a grammar. However, for applications that are necessarily knowledge-intensive, a classification-based system is likely to be preferable, because it organizes a large collection of linguistic knowledge (and related nonlinguistic knowledge) in such a way that it can be more efficiently processed.

From another perspective, the classification-based system can be seen as carrying out a compilation procedure on a linguistic knowledge base. The initial loading (or compilation) of a large grammar into the system may be computationally expensive, but the result is a parser that may be considerably more efficient at run-time than current unification-based systems. In the early stages of developing a grammar, when not many sentences are parsed with a particular version of the grammar before it is substantially revised, the benefits of compilation may not be appreciated. When the system is actually used in an application, or tested on a large body of text, it may significantly improve performance.

REFERENCES

Brachman, R. and Schmolze, J. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, Vol. 9:2, 1985.

Eisele, Andreas and Doerre, Jochen. Unification of Disjunctive Feature Descriptions. *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, NY: June 7-10, 1988.

Halliday, Michael. *System and Function in Language*. Kress G., editor, Oxford University Press, London, England, 1976.

Karttunen, Lauri. Features and Values. *Proceedings of the Tenth International Conference on Computational Linguistics: COLING 84*, Stanford, CA: July 2-7, 1984.

Kasper, Robert. A Flexible Interface for Linking Applications to Penman's Sentence Generator. *Proceedings of the DARPA Workshop on Speech and Natural Language*, Philadelphia: February, 1989.

Kasper, Robert. Conditional Descriptions in Functional Unification Grammar. *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, NY: June 7-10, 1988a.

Kasper, Robert. An Experimental Parser for Systemic Grammars. *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest: August, 1988b.

Kasper, Robert. A Unification Method for Disjunctive Feature Descriptions. *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford, CA: July 6-9, 1987.

Kay, Martin. Parsing in Functional Unification Grammar. *Natural Language Parsing*, Dowty D., Karttunen L., and Zwicky A. (eds.), Cambridge University Press, Cambridge, England, 1985.

MacGregor, Robert. A Deductive Pattern Matcher. *Proceedings of AAAI-88, The National Conference on Artificial Intelligence*, St. Paul, MN: August, 1988.

Pollard, Carl and Sag, Ivan. *Information Based Syntax*. CSLI Lecture Notes Number 13, Univeristy of Chicago Press, 1987.

Shieber, Stuart. The Design of a Computer Language for Linguistic Information. *Proceedings of the Tenth International Conference on Computational Linguistics: COLING 84*, Stanford, CA: July 2-7, 1984.

Smolka, Gert. A Feature Logic with Subsorts. LILOG Report 33, IBM Deutschland, Stuttgart, West Germany, May 1988.

Using Restriction to Optimize Unification Parsing

Dale Gerdemann [*] [†]

Department of Linguistics
Cognitive Science Group
Beckman Institute for Advanced Science and Technology
University of Illinois

1 Introduction

Since Shieber (1985), restriction has been recognized as an important operation in unification parsing. [1] As Shieber points out, the most straightforward adaptation of Earley's algorithm [2] for use with unification grammars fails because the infinite number of categories in these grammars can cause the predictor step in the algorithm to go into an infinite loop, creating ever more and more new predictions (i.e. the problem is that new predictions are not subsumed by previous predictions). The basic idea of restriction is to avoid making predictions on the basis of all of the information in a DAG, but rather to take some subset of that information (i.e. a restricted DAG–henceforth RD) and use just that information to make new predictions. Since there are only a finite number of possible RDs the predictor step will no longer go into the infinite loop described above. The price you pay for this move is that some spurious predictions will be made, but as Shieber points out, the algorithm is still correct since any spurious predictions will be weeded out by the completer step.

[*] Cognitive Science Group, Beckman Institute, 405 N. Mathews, Urbana, Ill 61801; dale@tarski.cogsci.uiuc.edu

[†] I would like to thank Alan Frisch, Erhard Hinrichs, Lucja Ivańska, Jerry Morgan, Mike Mendelson, and Tsuneko Nakazawa for their useful comments. Any deficiencies must rest with me. Thanks also to the UIUC Cognitive Science/Artificial Intelligence fellowship committee for the support that made this research possible.

[1] By unification parsing I mean parsing of unification grammars. See Seifert (1988) for a precise definition of a unification grammar.

[2] I will assume familiarity with the basic steps of Earley's algorithm as presented in Earley (1970). For an introduction to Earley's algorithm and its relationship to chart parsing in general see Winograd (1983).

Shieber's use of restriction in the predictor step is by now well established. On the other hand, there has been little discussion of the uses of restriction in other stages of parsing. In this paper, I will argue that restriction can be used to advantage in at least three additional ways. First, restriction can be used to significantly speed up the subsumption check on new predictions. Second, it can be used in the completer step in order to speed up the process of finding the correct states in the state sets to be completed. And third, it can be used to add a lookahead component to the unification parser. I will begin this paper by briefly reviewing Shieber's use of restriction and then I will discuss the three additional uses for restriction mentioned above.

2 Restriction in the Predictor Step

The original motivation for restriction was to avoid infinite cycles in the predictor step of Earley's algorithm. Shieber illustrates this problem with a "counting grammar" but the same point can be made using a type of grammar that is somewhat more familiar in recent linguistic theory. Specifically, infinite cycles can arise in grammars that handle subcategorization with list valued features such as Head Driven Phrase Structure Grammar (Pollard and Sag, 1987) or PATR style grammars (Shieber, 1986). To illustrate the problem, suppose that we are parsing a sentence using a grammar with the PATR style rules in (1,2). The problem of non-termination can arise with this grammar since rule (2) allows for lexical items with indefinitely long subcategorization lists.

(1) $x0 \rightarrow x1\ x2$

$$\begin{bmatrix} x0 & \begin{bmatrix} \mathit{cat} & s \end{bmatrix} \\ x1 & \boxed{1}\begin{bmatrix} \mathit{cat} & np \end{bmatrix} \\ x2 & \begin{bmatrix} \mathit{cat} & vp \\ \mathit{subcat} & \begin{bmatrix} \mathit{first} & \boxed{1} \\ \mathit{rest} & \mathit{end} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

(2) $x0 \rightarrow x1\ x2$

$$\begin{bmatrix} x0 & \begin{bmatrix} \mathit{cat} & vp \\ \mathit{subcat} & \boxed{1} \end{bmatrix} \\ x1 & \boxed{1}\begin{bmatrix} \mathit{cat} & vp \\ \mathit{subcat} & \begin{bmatrix} \mathit{first} & \boxed{2} \\ \mathit{rest} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ x2 & \boxed{2} \end{bmatrix}$$

The first step in parsing a sentence with this grammar is to find a rule whose left hand side unifies with the DAG described by the path equation $\langle \mathit{cat} \rangle = s$

(i.e. the start DAG). Since the rule in (1) satisfies this requirement, the next step is to make a prediction for the $x1$ daughter. In Earley's algorithm as it was originally formulated (Earley 1970), the prediction for $x1$ would simply be its category label (i.e. np). In this unification style grammar, however, category labels are just features like any other feature. Since the DAGs associated with each of the non-terminals $(x0, x1, \ldots, xn)$ in a rule may express just partial information about that non-terminal, it is possible that some non-terminals (such as $x2$ in the second rule) will not be associated with any category label at all. The natural solution, then, would be to make a prediction using the entire DAG associated with a given non-terminal. Suppose, now, that we have parsed the np in rule (1) and we're ready to parse $x2$. The DAG associated with $x2$ would be (3).

(3) $$\begin{bmatrix} \mathit{cat} & \mathit{vp} \\ \mathit{subcat} & \begin{bmatrix} \mathit{first} & \begin{bmatrix} \mathit{cat} & \mathit{np} \end{bmatrix} \\ \mathit{rest} & \mathit{end} \end{bmatrix} \end{bmatrix}$$

When this DAG unifies with the category on the left hand side of (2) we get the rule shown in (4).

(4) $$x0 \rightarrow x1\ x2$$
$$\begin{bmatrix} x0 & \begin{bmatrix} \mathit{cat} & \mathit{vp} \\ \mathit{subcat} & [2] \end{bmatrix} \\ x1 & \begin{bmatrix} \mathit{cat} & \mathit{vp} \\ \mathit{subcat} & \begin{bmatrix} \mathit{first} & [1] \\ \mathit{rest} & [2] \begin{bmatrix} \mathit{first} & \begin{bmatrix} \mathit{cat} & \mathit{np} \end{bmatrix} \\ \mathit{rest} & \mathit{end} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ x2 & [1] \end{bmatrix}$$

Now, following the same procedure, the predictor would next make a prediction for the non-terminal $x1$ in (4). It can easily be seen that when the DAG associated with $x1$ unifies with the left hand side of rule (2) the predicted rule is almost the same as (4) except that the value for ⟨*subcat rest*⟩ in (4) becomes the value for ⟨*subcat rest rest*⟩ in the new predict ı. In fact, the predictor step can continue making such predictions ad infinitum and, crucially, the new predictions will not be subsumed by previous predictions.

To solve this problem Shieber proposes that the predictor step should not use all of the information in the DAG associated with a non-terminal, but rather it should use some limited subset of that information. Of course, when some nodes of the DAG are eliminated the predictor step can overpredict, but this does not affect the correctness of the algorithm since these spurious predictions will not be completable. Shieber's proposal is basically that before the predictor step is applied, a RD should be created which contains just the information associated with a finite set of paths (i.e. a restrictor). [3] In this way, Shieber's algorithm allows an infinite number of categories to be divided into a finite number of equivalence classes. Since the number of possible RDs is finite it becomes impossible to make the kind of infinite cycle of predictions illustrated above.

Primarily for notational reasons, I will define restriction in a slightly different manner from Shieber (1985). For our purpose here we can define the RD D' of DAG D to be the least specific DAG D' $\sqsubseteq$ D such that for every path P in the restrictor if the value of P in D is atomic then the value of P in D' is the same as the value of P in D and if the value of P in D is complex then the value of P in D' is a variable. This differs from Shieber's definition in that reentrancies are eliminated in the RD. Thus the RD is not really a DAG but rather is a tree and hence it can be represented more easily by a simple list structure. For example,given the restrictor [⟨a b⟩, ⟨d e f⟩, ⟨d i j f⟩], the RD for the DAG in (5) (from Shieber 1985) will be represented by the indented list shown in (6), in which variables are indicated by [].[4]

(5) $$\left[\begin{array}{ll} a & \left[\begin{array}{ll} b & c \end{array}\right] \\ d & \left[\begin{array}{ll} e & [1]\left[\begin{array}{ll} f & \left[\begin{array}{ll} g & h \end{array}\right] \end{array}\right] \\ i & \left[\begin{array}{ll} j & [1] \end{array}\right] \\ k & l \end{array}\right] \end{array}\right]$$

(6)
```
[[a,[[b,c]]],
 [d,[[e,[[f,[]]]],
     [i,[[j,[[f,[]]]]]]]]]
```

3 Restriction in the Subsumption Test

The first use of restriction I will discuss involves the subsumption check on new predictions. In the original Earley's algorithm (Earley 1970), a check was made on each new prediction to see that an identical prediction had not already been made in the same state set. Of course, if duplicate predictions are retained the parser can fall into the left recursion trap. In Shieber's adaptation, however, this identity check is changed to the more general notion of a subsumption check. If a new DAG is predicted that is subsumed by a previous (more general) DAG,

[3] The question of how to select an appropriate restrictor for greatest efficiency must remain a question for further research. See the conclusion of this paper for further discussion.

[4] Eliminating reentrancies from RDs may also be a reasonable thing to do from a computational point of view. Judging from the particular restrictors used in Shieber (1985,1986) it would appear that reentrancies rarely occur in RDs. However, for some purposes it may be desirable to include more information in RDs. A possible example would be the use of parsing algorithms for generation, in which it would be desirable to use as much top down information as possible.

the new DAG is not retained since any DAGs that could be predicted on the basis of the new DAG could already have been predicted on the basis of the more general DAG. Clearly, the move from an identity check to a subsumption check is the right sort of move to make, but a subsumption check on arbitrarily large DAGs can be an expensive operation. This seems to be an ideal area in which restriction could be used to optimize the algorithm.

The move I propose is the following. Initially, new predictions are made in the manner suggested by Shieber; i.e. make a RD for the category "to the right of the Dot" and then collect all the rules from the grammar whose left hand side category unifies with this RD–these rules then constitute the new predictions. At this point I suggest that the RD used to find these predictions should be retained along with the new predictions; that is, a list of RDs that have been used to make predictions should be kept for each state set. I will call this list the RD_List. Then, the next time the parser enters the predictor step and creates a new RD from which to make new predictions, a subsumption check can be made directly between this RD and the RD_List. If the new RD is subsumed by any member of the RD_List then we can immediately give up trying to make any new predictions from this RD. Any predictions made from th RD would necessarily already have been made when the predictor encountered the more general RD in the RD_List. Thus we avoid both the expense of making new predictions and the expense of applying the subsumption test to weed these new predictions out. Moreover, since RDs are typically very small (at least given the sample restrictors given in Shieber (1985,1986)), the subsumption test that is performed on them can be applied very quickly.

As an example, suppose that some set of predictions has already been made using the RD, [[cat, np]], then there is no point in making predictions using [[cat, np],[num, sing]] since any such predictions would necessarily fail the subsumption check; i.e., rules expanding singular noun phrases are more specific than (or subsumed by) rules expanding noun phrases unspecified for number. This particular case probably does not arise often in actual parsing, but cases of left recursion do arise for which this optimization can make a very significant difference in processing speed. In fact our experience with the UNICORN natural language processing system (Gerdemann and Hinrichs 1988), has shown that for grammars with a large amount of left recursion, this simple optimization can make the difference between taking several minutes of processing time and several seconds of processing time.

4 Restriction in the Completer Step

The next use of restriction I propose involves the completer step. The completer applies, in Earley's algorithm, at the point where all of the right hand side of a rule in some state has been consumed, i.e., the point at which the "Dot" has been moved all the way to the right in some rule. At this point the completer goes back to the state set in which the state to be completed was originally predicted and searches for a prediction in this state set which has a category "to the right of the Dot" which can unify with the mother node of the rule in the state to be completed. This search can be quite time consuming since the completer must attempt to perform a unification for each state in this state set.

In each state, there is a variable F which indicates in which state set that state was predicted so the completer can immediately go back to the Fth state set in order to make the completion. But there is no variable which indicates which state in the Fth state set could have been responsible for making that prediction. And, in fact, it would be quite difficult to implement such a direct backpointer since in many cases a particular state is really only indirectly responsible for some prediction in the sense that it would have been responsible for the prediction if it had not been for the subsumption check. For example, suppose we try to implement a system of backpointers as follows. Each state will be a quintuple ⟨Lab,BP,Dot,F,Dag⟩ where Lab is an arbitrary label, BP is a kind of backpointer which takes as its value the label of the state that was responsible for predicting the current state and Dot, F, and Dag are as in Shieber's adaptation of Earley's algorithm; i.e., Dot is a pointer to the current position in the rule represented by Dag, and F is the more general kind of backpointer which only indicates in which state set the original prediction was made. To illustrate the problem with this scheme, consider the partial state set in (7), in which the subscripted i indicates that this is the ith state set.

(7) $_i[\ldots[Lab1, BP1, Dot1, F1, Dag1], [Lab2, BP2, Dot2, F2, Dag2], \ldots]$

Now suppose the RD for Dag1 is [[cat,np]] and that the RD for Dag2 is [[cat,np],[num,sing]]. When the predictor looks at state Lab1 it will make some number of predictions with backpointers to Lab1 as in (8) (For example, [Lab3,Lab1,0,i,Dag3] is a new state with an arbitrary label, Lab3, a backpointer to state Lab1, the Dot set at 0 indicating the beginning of the left hand side, F set to i indicating that the prediction was made in state set i, and Dag3 representing the new rule).

(8) $_i[\ldots[Lab1, BP1, Dot1, F1, Dag1], [Lab2, BP2, Dot2, F2, Dag2],$
$[Lab3, Lab1, Dot3, Dag3], [Lab4, Lab1, Dot4, F4, Dag4], \ldots]$

But when the predictor looks at Lab2 no predictions will be made since its RD is subsumed by the RD of Lab1. Thus even though (without the subsumption check) Lab2 could have been responsible for the predictions Lab3 and Lab4, no backpointers are created for Lab2.

It is at this point that RDs can again help us out. The idea is that when the predictor attempts to make predictions on the basis of some state it adds a RD to that state and to all predictions made from that state as a kind of marker (or coindexing between a state and the predictions resulting from that state). The RD used for this coindexing will be either 1.) the RD used to make the predictions or 2.) if no predictions were made because a more general RD had already been used to make predictions, then this more general RD is used as the marker. Now the completion step is greatly simplified. The completer can go back to the Fth state set and attempt unification only on states that have identical RD-markers. Clearly this move eliminates many attempted unifications that would be doomed to failure. To implement this idea states will be defined as quintuples ⟨BP,FP,Dot,F,Dag⟩ where BP is a RD acting as a backpointer, FP is a RD acting as a forward pointer and F,Dot, and Dag are as before. Now the analog of (7) will be (9).

(9) $_i[\ldots[BP1, FP1, Dot1, F1. Dag1], [BP2, FP2, Dot2, F2, Dag2], \ldots]$

In (9) BP1 and BP2 will each be instantiated to the value of the RD responsible for the prediction which created their respective state. FP1 and FP2, however will be uninstantiated variable since these two states have not yet been responsible for creating any new predictions. Now assuming that the RDs for Dag1 and Dag2 are as in (7) then when the predictor applies to the first state shown in (9), the result will be the state set shown in (10).

(10)
$$\begin{aligned}&_i[\ldots[BP1, [[cat, np]], Dot1, F1, Dag1],\\&[BP2, FP2, Dot2, F2, Dag2],\\&[[[cat, np]], FP3, Dot3, F3, Dag3],\\&[[[cat, np]], FP4, Dot4, F4, Dag4], \ldots]\end{aligned}$$

Then when the predictor looks at the second state in (10), no predictions will be made as before, however the predictor will register the attempt to make a prediction by instantiating the variable FP2 as in (11).

(11)
$$\begin{aligned}&_i[\ldots[BP1, [[cat, np]], Dot1, F1, Dag1],\\&[BP2, [[cat, np]], Dot2, F2, Dag2],\\&[[[cat, np]], FP3, Dot3, F3, Dag3],\\&[[[cat, np]], FP4, Dot4, F4, Dag4], \ldots]\end{aligned}$$

Now whenever the descendants of states 3 and 4 are ready to be completed, it will be easy to go back to this state set and find the states whose forward pointers are identical to the backpointers of the states to be completed. Thus many candidates for completion are immediately ruled out.

5 Restriction Used in Lookahead

The final use for restriction that I propose involves lookahead. Lookahead is one aspect of Earley's algorithm which clearly needs modification in order to be used efficiently with unification grammars or natural language grammars in general. In the original algorithm, a calculation of lookahead was performed as part of the prediction step. A simple example can show the problem with Earley's version of this procedure. In the S $\rightarrow$ NP VP rule, when the predictor makes a prediction for NP, it is required to add a state for each possible lookahead string that can be derived from the VP. But given the large number of verbs or adverbs that can start a VP in a natural language this would require adding a huge number of states to the state set. Clearly we don't want to simply list all the possible lookahead strings, but rather the correct approach would be to find what features these strings have in common and then add a smaller number of states with feature based lookaheads.

Aside from the question of what kind of lookahead to calculate, there are two other questions that need to be considered: first the question of when to calculate lookahead and second how to calculate it. Beginning with the when question, it is clear that unification grammars require lookahead to be calculated at a later point than it is in Earley's approach. The reason for this is illustrated by rules like (2) repeated here as (12)

(12)
$$x0 \rightarrow x1\ x2$$
$$\begin{bmatrix} x0 & \begin{bmatrix} \mathit{cat} & \mathit{vp} \\ \mathit{subcat} & [1] \end{bmatrix} \\ x1 & [1]\begin{bmatrix} \mathit{cat} & \mathit{vp} \\ \mathit{subcat} & \begin{bmatrix} \mathit{first} & [2] \\ \mathit{rest} & [1] \end{bmatrix} \end{bmatrix} \\ x2 & [2] \end{bmatrix}$$

According to Earley's approach, when a prediction is made for x1, the lookahead for $x2$ should be calculated. But in this case, no features for $x2$ will be specified until after x1 is parsed. This is an extreme situation, but it illustrates a general problem. It is the normal case in a unification grammar for the result of parsing one category to affect the feature instantiations on its sister. Clearly, what needs to be done in this case is to parse x1 and *then* perform a lookahead on $x2$. Thus, lookahead should be calculated for a category immediately before the predictor applies to that category; i.e., lookahead can be considered a quick check to be made immediately before applying prediction. Unlike Earley's original algorithm, then, it is not necessary to put a lookahead string into a state to be checked at a later point.

The question, then, is how to calculate lookahead. In Earley's version of the algorithm, there is a function, H_k which when applied to a category C returns a set of k-symbol strings of terminals which could begin a phrase of category C. When applied to unification grammars, however, the problem of having an infinite number of categories again appears. We certainly cannot list possible strings of preterminals that can begin each category. It is clear, then, that some form of restriction is again going to be necessary in order to implement any kind of lookahead. One, relatively simple, way of implementing this idea is as follows. When the predictor applies to a category C, the first thing it does is make a RD for C. Then a table lookup is performed to determine what preterminal categories could begin C. Since there are potentially infinite preterminal categories, restriction must be applied here too. So more precisely, the table lookup finds a set of RDs that could unify with whatever actual preterminal could begin a phrase of category C. Let us call these RDs the preterminal RDs. Then before the predictor can actually make a prediction a check must be performed to verify that the next item in the input is an instance of a category that can unify with one of the preterminal RDs. If the check fails, then the prediction is abandoned. All that remains is to specify how the lookup table is constructed. One way such a table might be constructed would be to run the parser in reverse for generation as in Shieber (1988) . Thus, for each possible RD (given a particular restrictor), the generator is used to determine what preterminal RDs can begin a phrase of this category.

6 Conclusion

I have argued here that restriction can be used in unification parsing to effect three optimizations. First, it can be used to greatly speed up the subsumption test for adding new predictions to the state set, second it can be used to speed up the searching used in the completer step, and finally it can be used to implement a form of lookahead. The first two of these uses have been fully implemented within the UNICORN natural language processing system (Gerdemann and Hinrichs 1988). The use of restriction with lookahead is still under development.

In general, the fact that unification grammars may have categories of indefinite complexity necessitates some way of focusing on limited portions of the information contained in these categories. It seems quite likely, then, that restriction would be useful even in other parsing algorithms for unification grammars. The primary question that remains is what portion of the information in complex DAGs should be used in these algorithms; that is, the question is how to choose a restrictor. Up to now, no general principles have been given for choosing a restrictor for greatest efficiency. Given the proposals in this paper, it becomes even more critical to find such general principles since restriction can affect the efficiency of several steps in the parsing algorithm.

References

[1] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 1970.

[2] Dale Gerdemann and Erhard Hinrichs. UNICORN: a unification parser for

attribute-value grammars. *Studies in the Linguistic Sciences*, 1988.

[3] Carl Pollard and Ivan Sag. *An Information-Based Approach to Syntax and Semantics: Volume 1 Fundamentals. CSLI Lecture Notes No. 13*, Chicago University Press, Chicago, 1987.

[4] Roland Seiffert. Chart-parsing of unification-based grammars with ID\LP-rules. In Ewan Klein and Johan van Benthem, editors, *Categories, Polymorphism and Unification*, pages 335–54, CCS/ILLI, Edinburgh/Amsterdam, 1987.

[5] Stuart Shieber. *An Introduction to Unification-Based Approaches to Grammar. CSLI Lecture Notes No. 4*, Chicago University Press, Chicago, 1986.

[6] Stuart Shieber. A uniform architecture for parsing and generation. In *COLING-88*, pages 614–9, 1988.

[7] Stuart Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *ACL Proceedings, 23rd Annual Meeting*, pages 145–52, 1985.

[8] Terry Winograd. *Language as a Cognitive Process: Syntax*. Ablex, Norwood, 1983.

An Overview of Disjunctive Constraint Satisfaction

John T. Maxwell III and Ronald M. Kaplan

Xerox Palo Alto Research Center

Introduction

This paper presents a new algorithm for solving disjunctive systems of constraints. The algorithm determines whether a system is satisfiable and produces the models if the system is satisfiable. There are three main steps for determining whether or not the system is satisfiable:

1) turn the disjunctive system into an equi-satisfiable conjunctive system in polynomial time
2) convert the conjunctive system into canonical form using extensions of standard techniques
3) extract and solve a propositional 'disjunctive residue'

Intuitively, the disjunctive residue represents the unsatisfiable combinations of disjuncts in a propositional form based on the content of the constraints. Each of the transformations above preserves satisfiability, and so the original disjunctive system is satisfiable if and only if the disjunctive residue is satisfiable. If the disjunctions are relatively independent (as frequently happens in grammatical specifications), then the disjunctive residue is significantly easier to solve than the original system.

The first three sections of this paper cover the steps outlined above. The fourth section describes how models can be produced. Finally, the last section compares this approach with some other techniques for dealing with disjunctive systems of constraints.

Turning Disjunctions into Conjunctions

Basic Lemma

Our method depends on a simple lemma for converting a disjunction into a conjunction of implications:

(1) $\phi_1 \vee \phi_2$ is satisfiable iff $(p \rightarrow \phi_1) \wedge (\neg p \rightarrow \phi_2)$ is satisfiable,
where p is a new propositional variable.

Proof:

1) If $\phi_1 \vee \phi_2$ is satisfiable, then either ϕ_1 is satisfiable or ϕ_2 is satisfiable. Suppose that ϕ_1 is satisfiable. Then if we choose p to be true, then $p \rightarrow \phi_1$ is satisfiable because ϕ_1 is satisfiable, and $\neg p \rightarrow \phi_2$ is vacuously satisfiable because its antecedent is false. Therefore $(p \rightarrow \phi_1) \wedge (\neg p \rightarrow \phi_2)$ is satisfiable.

2) If $(p \rightarrow \phi_1) \wedge (\neg p \rightarrow \phi_2)$ is satisfiable, then both clauses are satisfiable. One clause will be vacuously satisfiable because its antecedent is false and the other will have a true antecedent. Suppose that $p \rightarrow \phi_1$ is the clause with the true antecedent. Then ϕ_1 must be satisfiable for $p \rightarrow \phi_1$ to be satisfiable. But if ϕ_1 is satisfiable, then so is $\phi_1 \vee \phi_2$. Q.E.D.

Intuitively, the new variable p is used to encode the requirement that at least one of the disjuncts be true. In the remainder of the paper we use lower-case p to refer to a single propositional variable, and upper-case P to refer to a boolean combination of propositional variables. We call $P \rightarrow \phi$ a *contexted* constraint, where P is the *context* and ϕ is called the *base* constraint.

(Note that this lemma is stated in terms of *satisfiability*, not logical equivalence. A form of the lemma that emphasized logical equivalence would be: $\phi_1 \vee \phi_2 \leftrightarrow \exists p: (p \rightarrow \phi_1) \wedge (\neg p \rightarrow \phi_2)$.)

Turning a Disjunctive System into a Conjunctive System

The lemma given above can be used to convert a disjunctive system of constraints into an flat conjunction of contexted constraints in polynomial time. The resulting conjunction is satisfiable if and only if the original system is satisfiable. The algorithm for doing so is as follows:

(2) a) push all of the negations down to the literals
b) turn all of the disjunctions into conjunctions using the lemma above
c) flatten nested contexts with: $(P_i \rightarrow (P_j \rightarrow \phi)) \Leftrightarrow (P_i \wedge P_j \rightarrow \phi)$
d) separate conjoined constraints with: $(P_i \rightarrow \phi_1 \wedge \phi_2)) \Leftrightarrow (P_i \rightarrow \phi_1) \wedge (P_i \rightarrow \phi_2)$

This algorithm is a variant of the reduction used to convert disjunctive systems to CNF in the proof that CNF is NP-complete[4], and is thus known to run in polynomial time. In effect, we are simply converting the disjunctive system to an implicational form of CNF (note that $P \rightarrow \phi$ is logically equivalent to $\neg P \vee \phi$). CNF has the desirable property that if any one clause can be shown to be unsatisfiable, then the entire system is unsatisfiable.

Example

The functional structure f of an uninflected verb in English has the following constraints in the formalism of Lexical-Functional Grammar[6]:

(3) $((f\ INF) = - \wedge (f\ TENSE) = PRES \wedge \neg [(f\ SUBJ\ NUM) = SG \wedge (f\ SUBJ\ PERS) = 3]) \vee (f\ INF) = +$

(In LFG notation, a constraint of the form (f a) = v asserts that f(a) = v, where f is a function, a is an attribute, and v is a value. (f a b) = v is shorthand for f(a)(b) = v.) These constraints say that an uninflected verb in English is either a present tense verb which is not third person singular or it is infinitival. In the left column below this system has been reformatted so that it can be compared with the results of applying algorithm (2) to it, shown on the right:

reformatted:		*converts to*:	
(	$(f\ INF) = -$	$(p_1 \rightarrow$	$(f\ INF) = -\ ) \wedge$
$\wedge$	$(f\ TENSE) = PRES$	$(p_1 \rightarrow$	$(f\ TENSE) = PRES\ ) \wedge$
$\wedge \neg [$	$(f\ SUBJ\ NUM) = SG$	$(p_1 \wedge p_2 \rightarrow$	$(f\ SUBJ\ NUM) \neq SG\ ) \wedge$
$\wedge$	$(f\ SUBJ\ PERS) = 3\ \ ])$	$(p_1 \wedge \neg p_2 \rightarrow$	$(f\ SUBJ\ PERS) \neq 3\ ) \wedge$
$\vee$	$(f\ INF) = +$	$(\neg p_1 \rightarrow$	$(f\ INF) = +\ )$

Converting the Constraints to Canonical Form

A conjunction of contexted constraints can be put into an equi-satisfiable canonical form that makes it easy to identify all unsatisfiable combinations of constraints. The basic idea is to start with algorithms that determine the satisfiability of purely conjunctive systems and extend each rule of inference or rewriting rule so that it can handle contexted constraints. We illustrate this approach by modifying two conventional satisfiability algorithms, one based on deductive expansion and one based on rewriting.

Deductive Expansion

Deductive expansion algorithms work by determining all the deductions that could lead to unsatisfiability given an initial set of clauses and some rules of inference. The key to extending a deductive expansion algorithm to contexted constraints is to show that for every rule of inference that is applicable to the base constraints, there is a corresponding rule of inference that works for contexted constraints. The basic observation is that base constraints can be conjoined if their contexts are conjoined:

(4) $(P_1 \rightarrow \phi_1) \wedge (P_2 \rightarrow \phi_2) \Rightarrow (P_1 \wedge P_2 \rightarrow \phi_1 \wedge \phi_2)$

If we know from the underlying theory of conjoined base constraints that $\phi_1 \wedge \phi_2 \rightarrow \phi_3$, then the transitivity of implication gives us:

(5) $(P_1 \rightarrow \phi_1) \wedge (P_2 \rightarrow \phi_2) \Rightarrow (P_1 \wedge P_2 \rightarrow \phi_3)$

Equation (5) is the contexted version of $\phi_1 \wedge \phi_2 \rightarrow \phi_3$. Thus the following extension of a standard deductive expansion algorithm works for contexted constraints:

(6) For every pair of contexted constraints $P_1 \rightarrow \phi_1$ and $P_2 \rightarrow \phi_2$ such that:
a) there is a rule of inference $\phi_1 \wedge \phi_2 \rightarrow \phi_3$
b) $P_1 \wedge P_2 \neq$ FALSE
c) there are no other clauses $P_3 \rightarrow \phi_3$ such that $P_1 \wedge P_2 \rightarrow P_3$
add $P_1 \wedge P_2 \rightarrow \phi_3$ to the conjunction of clauses being processed.

Condition (6b) is based on the observation that any constraint of the form FALSE $\rightarrow \phi$ can be discarded since no unsatisfiable constraints can ever be derived from it. This condition is not necessary for the correctness of the algorithm, but may have performance advantages. Condition (6c) corresponds to the condition in the standard deductive expansion algorithm that redundant constraints must be discarded if the algorithm is to terminate. We extend this condition by noting that any constraint of the form $P_i \rightarrow \phi$ is redundant if there is already a constraint of the form $P_j \rightarrow \phi$, where $P_i \rightarrow P_j$. This is because any unsatisfiable constraints derived from $P_i \rightarrow \phi$ will also be derived from $P_j \rightarrow \phi$. Our extended algorithm terminates if the standard algorithm for simple conjunctions terminates. When it terminates, an equi-satisfiable disjunctive residue can be easily extracted, as described below.

Rewriting

Rewriting algorithms work by repeatedly replacing conjunctions of constraints with logically equivalent conjunctions until a normal form is reached. This normal form usually has the property that all unsatisfiable constraints can be determined by inspection. Rewriting algorithms use a set of rewriting rules that specify what sorts of replacements are allowed. These are based on logical equivalences so that no information is lost when replacements occur. Rewriting rules are interpreted differently from logical equivalences, however, in that they have directionality: whenever a logical expression matches the left-hand side of a rewriting rule, it is replaced by an instance of the logical expression on the right-hand side, but not vice-versa. To distinguish the two, we will use $\leftrightarrow$ for logical equivalence and $\Leftrightarrow$ for rewriting rules. (This corresponds our use of $\rightarrow$ for implication and $\Rightarrow$ for deduction above.)

A rewriting algorithm for contexted constraints can be produced by showing that for every rewrite rule that is applicable to the base constraints, there is a corresponding rewrite rule for contexted constraints. Suppose that $\phi_1 \wedge \phi_2 \Leftrightarrow \phi_3$ is a rewriting rule for base constraints. An obvious candidate for the contexted version of this rewrite rule would be to treat the deduction in (5) as a rewrite rule:

(7) $(P_1 \rightarrow \phi_1) \wedge (P_2 \rightarrow \phi_2) \Leftrightarrow (P_1 \wedge P_2 \rightarrow \phi_3)$ (incorrect)

This is incorrect because it is not a logical equivalence: the information that ϕ_1 is true in the context $P_1 \wedge \neg P_2$ and that ϕ_2 is true in the context $P_2 \wedge \neg P_1$ has been lost as the basis of future deductions. If we add clauses to cover these cases, we get the logically correct:

(8) $(P_1 \rightarrow \phi_1) \wedge (P_2 \rightarrow \phi_2) \Leftrightarrow (P_1 \wedge \neg P_2 \rightarrow \phi_1) \wedge (P_2 \wedge \neg P_1 \rightarrow \phi_2) \wedge (P_1 \wedge P_2 \rightarrow \phi_3)$

This is the contexted equivalent of $\phi_1 \wedge \phi_2 \Leftrightarrow \phi_3$. Note that the effect of this is that the contexted constraints on the right-hand side have unconjoinable contexts (that is, their conjunction is tautologically false). Thus, although the right-hand side of the rewrite rule has more conjuncts than the left-hand side, there are fewer implications to be derived from them.

Loosely speaking, a rewriting algorithm is constructed by iterative application of the contexted versions of the rewriting rules of a conjunctive theory. Rather than give a general outline here, let us consider the particular case of attribute value logic.

Application to Attribute-Value Logic

Attribute-value logic is used by both LFG and unification-based grammars. We will start with a simple version of the rewriting formalism given in Johnson[5]. For our purposes, we only need two of the rewriting rules that Johnson defines[5 pp. 38-39]:

(9) $t_1 \approx t_2 \Leftrightarrow t_2 \approx t_1$ when $\|t_1\| < \|t_2\|$ ($\|t_i\|$ is Johnson's norm for terms.)

(10) $t_2 \approx t_1 \wedge \phi \Leftrightarrow t_2 \approx t_1 \wedge \phi[t_2/t_1]$ where ϕ contains t_2 and $\|t_2\| > \|t_1\|$

($\phi[t_2/t_1]$ denotes "ϕ with every occurrence of t_2 replaced by t_1".)

We turn equation (10) into a contexted rewriting rule by a simple application of (7) above:

(11) $(P_1 \rightarrow t_2 \approx t_1) \wedge (P_2 \rightarrow \phi)$
$\Leftrightarrow (P_1 \wedge \neg P_2 \rightarrow t_2 \approx t_1) \wedge (\neg P_1 \wedge P_2 \rightarrow \phi) \wedge (P_1 \wedge P_2 \rightarrow (t_2 \approx t_1 \wedge \phi[t_2/t_1]))$

We can collapse the two instances of $t_2 \approx t_1$ together by observing that $(P \rightarrow A \wedge B) \leftrightarrow (P \rightarrow A) \wedge (P \rightarrow B)$ and that $(P_i \rightarrow A) \wedge (P_j \rightarrow A) \leftrightarrow (P_i \vee P_j \rightarrow A)$, giving the simpler form:

(12) $(P_1 \rightarrow t_2 \approx t_1) \wedge (P_2 \rightarrow \phi) \Leftrightarrow (P_1 \rightarrow t_2 \approx t_1) \wedge (P_2 \wedge \neg P_1 \rightarrow \phi) \wedge (P_2 \wedge P_1 \rightarrow \phi[t_2/t_1])$

Formula (12) is the basis for a very simple rewriting algorithm for a conjunction of contexted attribute-value constraints.

(13) For each pair of clauses $P_1 \rightarrow t_j \approx t_i$ and $P_2 \rightarrow \phi$:
a) if $\|t_j\| > \|t_i\|$, then set t_2 to t_j and t_1 to t_i, else set t_2 to t_i and t_1 to t_j
b) if ϕ mentions t_2 then replace $P_2 \rightarrow \phi$ with $(P_2 \wedge \neg P_1 \rightarrow \phi) \wedge (P_2 \wedge P_1 \rightarrow \phi[t_2/t_1])$

Notice that since $P_1 \rightarrow t_2 \approx t_1$ is carried over unchanged in (12), we only have to replace $P_2 \rightarrow \phi$ in (13b). Note also that if $P_2 \wedge P_1$ is FALSE, there is no need to actually add the clause $P_2 \wedge P_1 \rightarrow \phi[t_2/t_1]$ since no unsatisfiable constraints can be derived from it. Similarly if $P_2 \wedge \neg P_1$ is FALSE there is no need to add $P_2 \wedge \neg P_1 \rightarrow \phi$.

Example

The following example illustrates how this algorithm works. Suppose that (15) is the contexted version of (14):

(14) $[f_2 = f_1 \vee (f_1\ a) = c_1] \wedge [(f_2\ a) = c_2 \vee (f_1\ a) = c_3]$ where $c_i \neq c_j$ for all $i \neq j$

(15) a. $p_1 \rightarrow f_2 = f_1$
b. $\neg p_1 \rightarrow (f_1\ a) = c_1$
c. $p_2 \rightarrow (f_2\ a) = c_2$
d. $\neg p_2 \rightarrow (f_1\ a) = c_3$

(For clarity, we omit the $\wedge$'s whenever contexted constraints are displayed in a column.) There is an applicable rewrite rule for constraints (15a) and (15c) that produces three new constraints:

(16)
$$\begin{array}{rcl} p_1 & \rightarrow & f_2 = f_1 \\ p_2 & \rightarrow & (f_2\ a) = c_2 \end{array} \quad \Leftrightarrow \quad \begin{array}{rcl} p_1 & \rightarrow & f_2 = f_1 \\ \neg p_1 \wedge p_2 & \rightarrow & (f_2\ a) = c_2 \\ p_1 \wedge p_2 & \rightarrow & (f_1\ a) = c_2 \end{array}$$

Although there is an applicable rewrite rule for (15d) and the last clause of (16), we ignore it since $p_1 \wedge p_2 \wedge \neg p_2$ is FALSE. The only other pair of constraints that can be rewritten are (15b) and (15d), producing three more constraints:

(17)
$$\begin{array}{rcl} \neg p_1 & \rightarrow & (f_1\ a) = c_1 \\ \neg p_2 & \rightarrow & (f_1\ a) = c_3 \end{array} \quad \Leftrightarrow \quad \begin{array}{rcl} \neg p_1 & \rightarrow & (f_1\ a) = c_1 \\ p_1 \wedge \neg p_2 & \rightarrow & (f_1\ a) = c_3 \\ \neg p_1 \wedge \neg p_2 & \rightarrow & c_1 = c_3 \end{array}$$

Since no more rewrites are possible, the normal form of (15) is thus:

(18)
a. $p_1 \rightarrow f_2 = f_1$
b. $\neg p_1 \rightarrow (f_1\ a) = c_1$
c. $\neg p_1 \wedge p_2 \rightarrow (f_2\ a) = c_2$
d. $p_1 \wedge \neg p_2 \rightarrow (f_1\ a) = c_3$
e. $p_1 \wedge p_2 \rightarrow (f_1\ a) = c_2$
f. $\neg p_1 \wedge \neg p_2 \rightarrow c_1 = c_3$

Extracting the Disjunctive Residue

When the rewriting algorithm is finished, all unsatisfiable combinations of base constraints will have been derived. But more reasoning must be done to determine from base unsatisfiabilities whether the disjunctive system is unsatisfiable. Consider the contexted constraint $P \rightarrow \phi$, where ϕ is unsatisfiable. In order for the conjunction of contexted constraints to be satisfiable, it must be the case that $\neg P$ is true. We call $\neg P$ a *nogood*, following deKleer's terminology[1]. Since P contains propositional variables indicating disjunctive choices, information about which conjunctions of base constraints are unsatisfiable is thus back-propagated into information about the unsatisfiability of the conjunction of the disjuncts that they come from. The original system as a whole is satisfiable just in case the conjunction of all its nogoods is true. We call the conjunction of all of the nogoods the *residue* of the disjunctive system.

For example, clause (18f) asserts that $\neg p_1 \wedge \neg p_2 \rightarrow c_1 = c_3$. But $c_1 = c_3$ is unsatisfiable, since we know that $c_1 \neq c_3$. Thus $\neg(\neg p_1 \wedge \neg p_2)$ is a nogood. Since $c_1 = c_3$ is the only unsatisfiable base constraint in (18), this is also the disjunctive residue of the system. Thus (14) is satisfiable because $\neg(\neg p_1 \wedge \neg p_2)$ has at least one solution (e.g. p_1 is true and p_2 is true).

Since each nogood may be a complex boolean expression involving conjunctions, disjunctions and negations of propositional variables, determining whether the residue is satisfiable may not be easy. In fact, the problem is NP complete. However, we have accomplished two things by reducing a disjunctive system to its residue. First, since the residue only involves propositional variables, it can be solved by propositional reasoning techniques (such as deKleer's ATMS) that do not require specialized knowledge of the problem domain. Second, we believe that for the particular case of linguistics, the final residue will be simpler than the original disjunctive problem. This is because the disjunctions introduced from different parts of the sentence usually involve different attributes in the feature structure, and thus they tend not to interact.

Another way that nogoods can be used is to reduce contexts while the rewriting is being carried out, using identities like the following:

(19) $\neg P_1 \wedge (\neg P_1 \wedge P_2 \rightarrow \phi) \Leftrightarrow \neg P_1 \wedge (P_2 \rightarrow \phi)$

(20) $\neg P_1 \wedge (P_1 \wedge P_2 \rightarrow \phi) \Leftrightarrow \neg P_1$

(21) $P_1 \wedge \neg P_1 \Leftrightarrow \text{FALSE}$

Doing this can improve the performance since some contexts are simplified and some constraints are eliminated altogether. However, the overhead of comparing the nogoods against the contexts may outweigh the potential benefit.

Producing the Models

Assuming that there is a method for producing a model for a conjunction of base constraints, we can produce models from the contexted system. Every assignment of truth values to the propositional variables introduced in (1) corresponds to a different conjunction of base constraints in the original system, and each such conjunction is an element of the DNF of the original system. Rather than explore the entire space of assignments, we need only enumerate those assignments for which the disjunctive residue is true.

Given an assignment of truth values that is consistent with the disjunctive residue, we can produce a model from the contexted constraints by assigning the truth values to the propositional variables in the contexts, and then discarding those base constraints whose contexts evaluate to false. The minimal model for the remaining base constraints can be determined by inspection if the base constraints are in normal form, as is the case for rewriting algorithms. (Otherwise some deductions may have to be made to produce the model, but the system is guaranteed to be satisfiable.) This minimal model will satisfy the original disjunctive system.

Example

The residue for the system given in (18) is $\neg(\neg p_1 \wedge \neg p_2)$. This residue has three solutions : p_1 and p_2 both true, p_1 true and p_2 false, and p_1 false and p_2 true. We can produce models for these solutions by extracting the appropriate constraints from (18), and reading off the models. Here are the solutions for this system:

	solution:	*constraints:*	*model:*
(22)	p_1 true, p_2 true:	$f_2 = f_1 \wedge (f_1\ a) = c_2$	$\begin{array}{l} f1 \\ f2 \end{array}\begin{bmatrix} a & c2 \\ & \end{bmatrix}$
(23)	p_1 true, p_2 false:	$f_2 = f_1 \wedge (f_1\ a) = c_3$	$\begin{array}{l} f1 \\ f2 \end{array}\begin{bmatrix} a & c3 \\ & \end{bmatrix}$
(24)	p_1 false, p_2 true:	$(f_1\ a) = c_1 \wedge (f_2\ a) = c_2$	$f1 \begin{bmatrix} a & c1 \end{bmatrix}$ & $f2 \begin{bmatrix} a & c2 \end{bmatrix}$

Comparison with Other Techniques

In this section we compare disjunctive constraint satisfaction with some of the other techniques that have been developed for dealing with disjunction as it arises in grammatical processing. These other techniques are framed in terms of feature-structure unification and a unification version of our approach would facilitate the comparisons. Although we do not provide a detailed specification of context-extended unification here, we note that unification can be thought of as an indexing scheme for rewriting. We start with a simple illustration of how such an indexing scheme might work.

Unification Indexing

Regarding unification as an indexing scheme, the main question that needs to be answered is where to index the contexts. Suppose that we index the contexts with the values under the attributes. Then the attribute-value (actually, attribute-*context*-value) matrix for (25a) would be (25b):

(25) a. $(f\,a) = c_1 \vee ((f\,b) = c_2 \vee (f\,a) = c_3)$ b. $\left[\begin{array}{ll} a & \left[\begin{array}{ll} p1 & c1 \\ \text{-}p1\&\text{-}p2 & c3 \end{array}\right] \\ b & \left[\begin{array}{ll} \text{-}p1\&p2 & c2 \end{array}\right] \end{array}\right]$

Since the contexts are indexed under the attributes, two disjunctions will only interact if they have attributes in common. If they have no attributes in common, their unification will be linear in the number of attributes, rather than multiplicative in the number of disjuncts. For instance, suppose that (26b) is the attribute value matrix for (26a):

(26) a. $(f\,c) = c_4 \vee ((f\,d) = c_5 \vee (f\,e) = c_6)$ b. $\left[\begin{array}{ll} c & \left[\begin{array}{ll} p3 & c4 \end{array}\right] \\ d & \left[\begin{array}{ll} \text{-}p3\&p4 & c5 \end{array}\right] \\ e & \left[\begin{array}{ll} \text{-}p3\&\text{-}p4 & c6 \end{array}\right] \end{array}\right]$

Since these disjunctions have no attributes in common, the attribute-value matrix for the conjunction of (25a) and (26a) will be simply the *concatenation* of (25b) and (26b):

(27) $\left[\begin{array}{ll} a & \left[\begin{array}{ll} p1 & c1 \\ \text{-}p1\&\text{-}p2 & c3 \end{array}\right] \\ b & \left[\begin{array}{ll} \text{-}p1\&p2 & c2 \end{array}\right] \\ c & \left[\begin{array}{ll} p3 & c4 \end{array}\right] \\ d & \left[\begin{array}{ll} \text{-}p3\&p4 & c5 \end{array}\right] \\ e & \left[\begin{array}{ll} \text{-}p3\&\text{-}p4 & c6 \end{array}\right] \end{array}\right]$

The DNF approach to this problem would produce nine f-structures with eighteen attribute-value pairs. In contrast, our approach produces one f-structure with eleven attribute-value or context-value pairs. In general, if disjunctions have independent attributes, then a DNF approach is exponential in the number of disjunctions, whereas our approach is linear. This independence feature is very important for language processing, since, as we have suggested, disjunctions from different parts of a sentence usually constrain different attributes.

Karttunen's Disjunctive Values

Karttunen[7] introduced a special type of value called a "disjunctive value" to handle certain types of disjunctions. Disjunctive values allow simple disjunctions such as:

(28) $(f\ \text{CASE}) = \text{ACC} \vee (f\ \text{CASE}) = \text{NOM}$

to be represented in the unification data structure as:

(29) $\left[\text{CASE}\ \{\text{ACC NOM}\}\right]$

where the curly brackets indicate a disjunctive value. Karttunen's disjunctive values are not limited to atomic values, as the example he gives for the German article "die" shows:

(30) $\text{die} = \left[\begin{array}{ll} \text{INFL} & \left[\begin{array}{ll} \text{CASE} & \{\text{NOM ACC}\} \\ \text{AGR} & \left\{\begin{array}{l} \left[\begin{array}{ll} \text{GENDER} & \text{FEM} \\ \text{NUMBER} & \text{SG} \end{array}\right] \\ \left[\begin{array}{ll} \text{NUMBER} & \text{PL} \end{array}\right] \end{array}\right\} \end{array}\right] \end{array}\right]$

The corresponding attribute-context-value matrix for our scheme would be:

(31) $$\text{die} = \begin{bmatrix} \text{INFL} & \begin{bmatrix} \text{CASE} & \begin{bmatrix} p1 & \text{NOM} \\ \neg p1 & \text{ACC} \end{bmatrix} \\ \text{AGR} & \begin{bmatrix} \text{GENDER} & \begin{bmatrix} p2 & \text{FEM} \end{bmatrix} \\ \text{NUMBER} & \begin{bmatrix} p2 & \text{SG} \\ \neg p2 & \text{PL} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The advantage of disjunctive constraint satisfaction is that it can handle all types of disjunctions, whereas disjunctive values can only handle atomic values or simple feature-value matrices with no external dependencies. Furthermore, disjunctive constraint satisfaction can often do better than disjunctive values for the types of disjunctions that they can both handle. This can be seen in (31), where disjunctive constraint satisfaction has pushed a disjunction further down the AGR feature than the disjunctive value approach in (30). This means that if AGR were given an attribute other than GENDER or NUMBER, this new attribute would not interact with the existing disjunction.

However, disjunctive values may have an advantage of reduced overhead, because they do not require embedded contexts and they do not have to keep track of nogoods. It may be worthwhile to incorporate disjunctive values in our scheme to represent the very simple disjunctions, while disjunctive constraint satisfaction is used for the more complex disjunctions.

Kasper's Successive Approximation

Kasper[8, 9] proposed that an efficient way to handle disjunctions is to do a step-wise approximation for determining satisfiability. Conceptually, the step-wise algorithm tries to find the inconsistencies that come from fewer disjuncts first. The algorithm starts by unifying the non-disjunctive constraints together. If the non-disjunctive constraints are inconsistent, then there is no need to even consider the disjunctions. If they are consistent, then the disjuncts are unified with them one at a time, where each unification is undone before the next unification is performed. If any of these unifications are inconsistent, then its disjunct is discarded. Then the algorithm unifies the non-disjunctive constraints with all possible pairs of disjuncts, and then all possible triples of disjuncts, and so on. (This technique is called "k-consistency" in the constraint satisfaction literature[3].) In practice, Kasper noted that only the first two steps are computationally useful, and that once bad singleton disjuncts have been eliminated, it is more efficient to switch to DNF than to compute all of the higher degrees of consistency.

Kasper's technique is optimal when most of the disjuncts are inconsistent with the non-disjunctive constraints, or the non-disjunctive constraints are themselves inconsistent. His scheme tends to revert to DNF when this is not the case. Although simple inconsistencies are prevalent in many circumstances, we believe they become less predominate as grammars are extended to cover more and more linguistic phenomena. The coverage of a grammar increases as more options and alternatives are added, either in phrasal rules or lexical entries, so that there are fewer instances of pure non-disjunctive constraints and a greater proportion of inconsistencies involve higher-order interactions. This tendency is exacerbated because of the valuable role that disjunctions play in helping to control the complexity of broad-coverage grammatical specifications. Disjunctions permit constraints to be formulated in local contexts, relying on a general global satisfaction procedure to enforce them in all appropriate circumstances, and thus they improve the modularity and manageability of the overall grammatical system. We have seen this trend towards more localized disjunctive specifications particularly in our developing LFG grammars, and have observed a corresponding reduction in the number of disjuncts that can be eliminated using Kasper's technique. On the other hand, the number of independent disjunctions, which our approach does best on, tends to go up as modularity increases.

One other aspect of LFG grammatical processing is worth noting. Many LFG analyses are ruled out not because they are inconsistent, but rather because they are incomplete. That is, they fail to have an attribute that a predicate requires (e.g. the object is missing for a transitive verb). Since incomplete solutions cannot be ruled out incrementally (an incomplete solution may become complete with the addition of more information), completeness requirements provide no information to eliminate disjuncts in Kasper's successive approximation. These requirements can only be evaluated in what is effectively a disjunctive normal form computation. But our technique avoids this problem, since independent completeness requirements will be simply additive, and any incomplete contexts can be easily read off of the attribute-value matrix and added to the nogoods before solving the residue.

Kasper's scheme works best when disjuncts can be eliminated by unification with non-disjunctive constraints, while ours works best when disjunctions are independent. It is possible to construct a hybrid scheme that works well in both situations. For example, we can use Kasper's scheme up until some critical point (e.g. after the first two steps), and then switch over to our technique instead of computing the higher degrees of consistency.

Another, possibly more interesting, way to incorporate Kasper's strategy is to always process the sets of constraints with the fewest number of propositional variables first. That is, if $P_3 \wedge P_4$ had fewer propositional variables than $P_1 \wedge P_2$, then the rewrite rule in (32b) should be done before (32a):

(32) a. $(P_1 \rightarrow \phi_1) \wedge (P_2 \rightarrow \phi_2) \Rightarrow (P_1 \wedge P_2 \rightarrow \phi_5)$

b. $(P_3 \rightarrow \phi_3) \wedge (P_4 \rightarrow \phi_4) \Rightarrow (P_3 \wedge P_4 \rightarrow \phi_6)$

This approach would find smaller nogoods earlier, which would allow combinations of constraints that depended on those nogoods to be ignored, since the contexts would already be known to be inconsistent.

Eisele and Dörre's techniques

Eisele and Dörre[2] developed an algorithm for taking Karttunen's notion of disjunctive values a little further. Their algorithm allows disjunctive values to be unified with reentrant structures. The algorithm correctly detects such cases and "lifts the disjunction due to reentrancy". They give the following example:

(33)
$$\begin{bmatrix} a: \left\{ \begin{matrix} \begin{bmatrix} b: & + \\ c: & - \end{bmatrix} \\ \begin{bmatrix} b: & - \\ c: & + \end{bmatrix} \end{matrix} \right\} \end{bmatrix} \cup \begin{bmatrix} a: & \begin{bmatrix} b: & \langle d \rangle \end{bmatrix} \\ d: & [\,] \end{bmatrix} = \left\{ \begin{matrix} \begin{bmatrix} a: & \begin{bmatrix} b: & \langle d \rangle \\ c: & - \end{bmatrix} \\ d: & + \end{bmatrix} \\ \begin{bmatrix} a: & \begin{bmatrix} b: & \langle d \rangle \\ c: & + \end{bmatrix} \\ d: & - \end{bmatrix} \end{matrix} \right\}$$

Notice that the disjunction under the "a" attribute in the first matrix is moved one level up in order to handle the reentrancy introduced in the second matrix under the "b" attribute.

This type of unification can be handled with embedded contexts without requiring that the disjunction be lifted up. In fact, the disjunction is moved down one level, from under "a" to under "b" and "c":

(34)
$$\begin{bmatrix} a: & \begin{bmatrix} b: & \begin{bmatrix} p1 & + \\ \neg p1 & - \end{bmatrix} \\ c: & \begin{bmatrix} p1 & - \\ \neg p1 & + \end{bmatrix} \end{bmatrix} \end{bmatrix} \cup \begin{bmatrix} a: & \begin{bmatrix} b: & \langle d \rangle \end{bmatrix} \\ d: & [\,] \end{bmatrix} = \begin{bmatrix} a: & \begin{bmatrix} b: & \langle d \rangle \\ c: & \begin{bmatrix} p1 & - \\ \neg p1 & + \end{bmatrix} \end{bmatrix} \\ d: & \begin{bmatrix} p1 & + \\ \neg p1 & - \end{bmatrix} \end{bmatrix}$$

Overall

The major cost of using disjunctive constraint satisfaction is the overhead of dealing with contexts and the disjunctive residue. Our technique is quite general, but if the only types of disjunction that occur are covered by one of the other techniques, then that technique will probably do better than our scheme. For example, if all of the nogoods are the result of singleton inconsistencies (the result of unifying a single disjunct with the non-disjunctive part), then Kasper's successive approximation technique will work better because it avoids our overhead. However, if many of the nogoods involve multiple disjuncts, or if some nogoods are only produced from incomplete solutions, then disjunctive constraint satisfaction will do better than the other techniques, sometimes exponentially so. We also believe that further savings can be achieved by using hybrid techniques if the special cases are sufficiently common to warrant the extra complexity.

Acknowledgements

The approach described in this paper emerged from discussion and interaction with a number of our colleagues. We are particularly indebted to John Lamping, who suggested the initial forumulation of the basic lemma, and to Bill Rounds for pointing out the relationship between our conversion algorithm and the NP completeness reduction for CNF. We are also grateful for many helpful discussions with Dan Bobrow, Johan deKleer, Jochen Dörre, Andreas Eisele, Pat Hayes, Mark Johnson, Lauri Karttunen, and Martin Kay.

References

[1] deKleer, J. (1986). An Assumption-based TMS. *Artificial Intelligence* 28, 127-162.

[2] Eisele, A. and Dörre, J. (1988). Unification of Disjunctive Feature Descriptions. *Proceedings of the 26th Annual Meeting of the ACL.* Buffalo, New York.

[3] Freuder, E.C. (1978). Synthesizing Constraint Expressions. *Communications of the ACM* **21**, 958-966.

[4] Hopcroft, J. and Ullman, J. (1979). *Introduction to Automata Theory, Languages and Computation.* p. 328-330.

[5] Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar.* Ph.D. Thesis. Stanford University.

[6] Kaplan, R. and Bresnan, J. (1982). Lexical Functional Grammar: A Formal System for Grammatical Representation. In J. Bresnan (ed.), *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts.

[7] Karttunen, L. (1984). Features and Values. *Proceedings of COLING 1984*, Stanford, CA.

[8] Kasper, R.T. (1987). *Feature Structures: A Logical Theory with Application to Language Analysis.* Ph.D. Thesis. University of Michigan.

[9] Kasper, R.T. (1987). A Unification Method for Disjunctive Feature Descriptions. *Proceedings of the 25th Annual Meeting of the ACL*, Stanford, CA.

A Uniform Formal Framework for Parsing

Bernard Lang

INRIA

B.P. 105, 78153 Le Chesnay, France

lang@inria.inria.fr

1 Introduction

Many of the formalisms used to define the syntax of natural (and programming) languages may be located in a continuum that ranges from propositional Horn logic to full first order Horn logic, possibly with non-Herbrand interpretations. This structural parenthood has been previously remarked: it lead to the development of Prolog [Col-78, Coh-88] and is analyzed in some detail in [PerW-80]. A notable outcome is the parsing technique known as Earley deduction [PerW-83].

These formalisms play (at least) three roles:

descriptive: they give a finite and organized description of the syntactic structure of the language,

analytic: they can be used to analyze sentences so as to retrieve a syntactic structure (i.e. a representation) from which the meaning can be extracted,

generative: they can also be used as the specification of the concrete representation of sentences from a more structured abstract syntactic representation (e.g. a parse tree).

The choice of a formalism is essential with respect to the descriptive role, since it controls the perspicuity with which linguistic phenomena may be understood and expressed in actual language descriptions, and hence the tractability of these descriptions for the human mind.

However, computational tractability is required by the other two roles if we intend to use these descriptions for mechanical processing of languages.

The aim of our work, which is partially reported here, is to obtain a uniform understanding of the computational aspects of syntactic phenomena within the continuum of Horn-like formalisms considered above, and devise general purpose algorithmic techniques to deal with these formalisms in practical applications.

To attain this goal, we follow a three-sided strategy:

- Systematic study of the lower end of the continuum, represented by context-free (CF) grammars (simpler formalisms, such as propositional Horn logic do not seem relevant for our purpose).

- Systematic study of the higher end of the continuum, i.e. first order Horn clauses.
- Analysis of the relations between intermediate formalisms and Horn clauses, so as to reuse for intermediate formalisms the understanding and algorithmic solutions developed for the more powerful Horn clauses.

This strategy is motivated by two facts:

- the computational properties of both CF grammars and Horn clauses may be expressed with the same computational model: the non-deterministic pushdown automaton,
- the two formalisms have a compatible concept of syntactic structure: the parse-tree in the CF case, and the proof-tree in the Horn clause case.

The greater simplicity of the CF formalism helps us in understanding more easily most of the computational phenomena. We then generalize this knowledge to the more powerful Horn clauses, and finally we specialize it from Horn clauses to the possibly less powerful but linguistically more perspicuous intermediate formalisms.

In the rest of this paper we present two aspects of our work:

1. a new understanding of shared parse forests and their relation to CF grammars, and
2. a generalization to full Horn clauses, also called *Definite Clause (DC) programs*, of the push-down stack computational model developed for CF parsers.

2 Context-Free Parsing

Though much research has been devoted to this subject in the past, most of the practically usable work has concentrated on deterministic push-down parsing which is clearly inadequate for natural language applications and does not generalize to more complex formalisms. On the other hand there has been little formal investigation of general CF parsing, though many practical systems have been implemented based on some variant of Earley's algorithm.

Our contribution has been to develop a formal model which can describe these variants in a uniform way, and encompasses the construction of parse-trees, and more generally of parse-forests. This model is based on the *compilation paradigm* common in programming languages and deterministic parsing: we use the non-deterministic[1] *Pushdown Automaton (PDA)* as a virtual parsing machine which we can simulate with an Earley-like construction; variations on Earley's algorithm are then expressed as variations in the compilation schema used to produce the PDA code from the original CF grammar. This uniform framework has been used to compare experimentally parsing schemata w.r.t. parser size, parsing speed and size of shared forest, and in reusing the wealth of PDA construction techniques to be found in the literature.

This work has been reported elsewhere [Lan-74, BilL-88, Lan-88a]. An essential outcome, which is the object of this section, is a new understanding of the relation between CF grammars, parse-trees and parse-forests, and the parsing process itself. The presentation is informal since our

[1] In this paper, the abbreviation PDA always implies the possibility of non-determinism

(1)	S	::=	NP	VP
(2)	S	::=	S	PP
(3)	NP	::=	n	
(4)	NP	::=	det	n
(5)	NP	::=	NP	PP
(6)	PP	::=	prep	NP
(7)	VP	::=	v	NP

Figure 1: A Context-Free Grammar

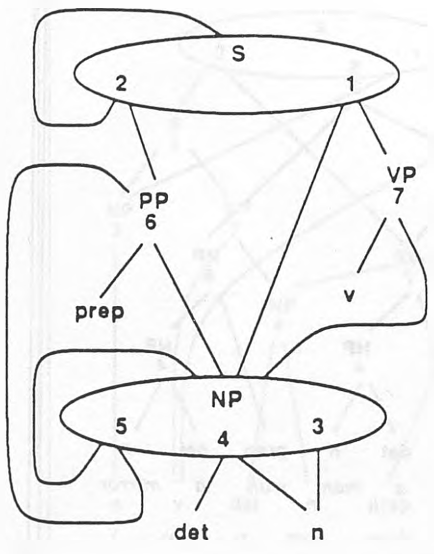


Figure 2: Graph of the Grammar

purpose is to give an intuitive understanding of the concepts, which is our interpretation of the earlier theoretical results.

Essentially, we shall first show that both CF grammars and shared parsed forest may be represented by AND-OR graphs, with specific interpretations. We shall then argue that this representational similarity is not accidental, and that there is no difference between a shared forest and a grammar.

2.1 Context-free Grammars

Our running example for a CF grammar is the pico-grammar of English, taken from [Tom-87], which is given in figure 1.

In figure 2 we give a graphical representation of this grammar as an AND-OR graph. The notation for this AND-OR graph is unusual and emphasizes the difference between AND and OR nodes. OR-nodes are represented by the non-terminal categories of the grammar, and AND-nodes are represented by the rules (numbers) of the grammar. There are also leaf-nodes corresponding to the terminal categories.

The OR-node corresponding to a non-terminal X has exiting arcs leading to each AND-node n representing a rule that defines X. This arc is not explicitly represented in the graphical formalism chosen. If there is only one such arc, then it is represented by placing n immediately under X. This is the case for the OR-node representing the non-terminal PP. If there are several such arcs, they are implicitly represented by enclosing in an ellipse the OR-node X above all its son nodes n, n', ... This is the case for the OR-node representing the non-terminal NP.

The sons of an AND-node (i.e. a rule) are the grammatical categories found in the right-hand-side of the rule, *in that order*. The arcs leading from an AND-node to its sons are represented explicitly. The convention for orienting the arcs is that they leave a node from below and reach a node from above.

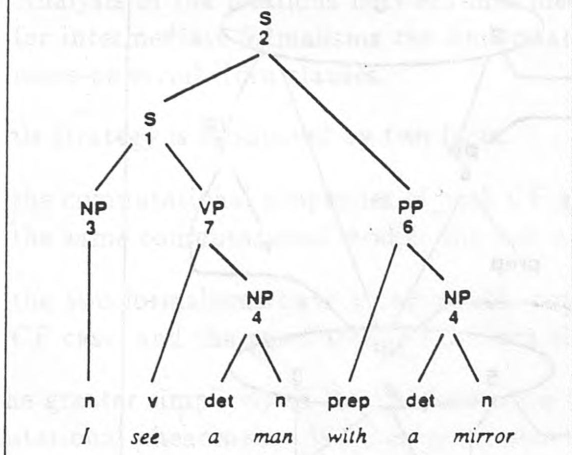


Figure 3: A parse tree

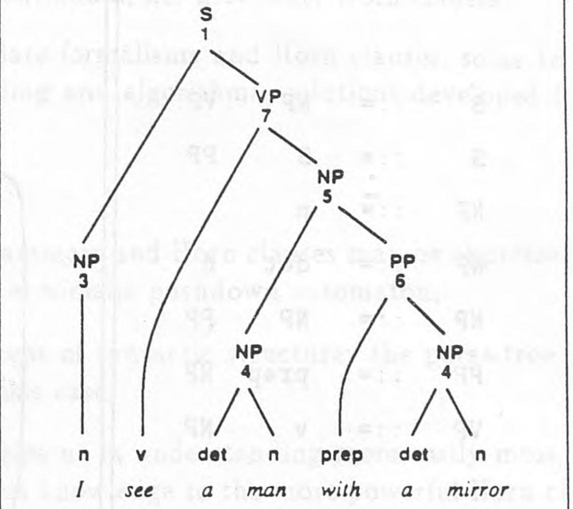


Figure 4: Another parse tree

This graph accurately represents the grammar, and is very similar to the graphs used in some parsers. For example, LR(0) parsing uses a graph representation of the grammar that is very similar, the main difference being that the sons of AND-nodes are linked together from left to right, rather than being attached separately to the AND-node [AhoU-72, DeR-71]. More simply, this graph representation is very close to the data structures often used to represent conveniently a grammar in a computer memory.

A characteristic of the AND/OR graph representing a grammar is that all nodes have different labels. Conversely, any labelled AND/OR graph such that all node labels are different may be read as — translated into — a CF grammar such that AND-node labels are rule names, OR-node labels represent non-terminal categories, and leaf-node labels represent terminal categories.

2.2 Parse trees and parse forests

Given a sentence in the language defined by a CF grammar, the parsing process consists in building a tree structure, *the parse tree*, that shows how this sentence can be constructed according to the grammatical rules of the language. It is however frequent that the CF syntax of a sentence is ambiguous, i.e. that several distinct parse-trees may be constructed for it.

Let us consider the grammar of figure 1.

If we take as example the sentence "`I see a man with a mirror`", which translate into the terminal sequence "`n v det n prep det n`", we can build the two parse trees given in figures 3 and 4. Note that we label a parse tree node with its non-terminal category and with the rule used to decompose it into constituents. Hence such a parse tree could be seen as an AND-OR tree similar to the AND-OR grammar graph of figure 2. However, since all OR-nodes are degenerated (i.e. have a unique son), a parse tree is just an AND-tree.

The number of possible parse trees may become very large when the size of sentences increases: it may grow exponentially with that size, and may even be infinite for cyclic grammars (which seem of little linguistic usefulness [PerW-83, Tom-85]). Since it is often desirable to consider all

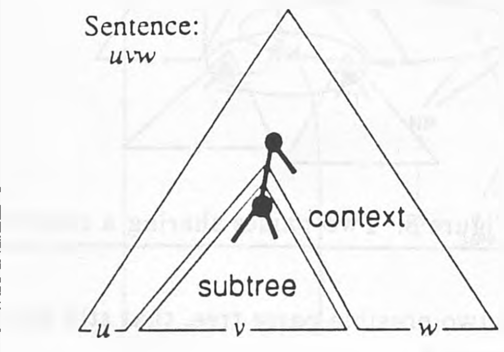


Figure 5: Context and Subtree

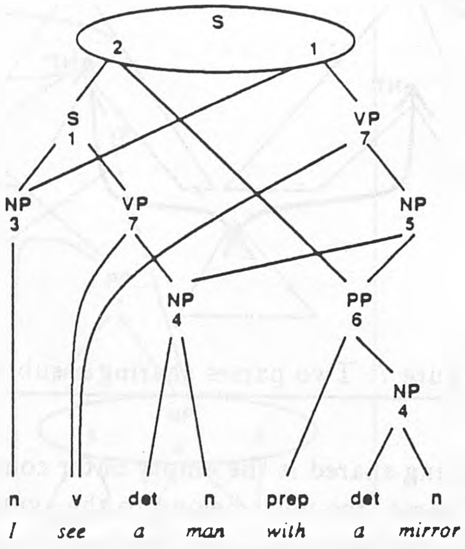


Figure 6: A shared parse forest

possible parse trees (e.g. for semantic processing), it is convenient to merge as much as possible these parse trees into a single structure that allows them to share common parts. This sharing save on the space needed to represent the trees, and also on the later processing of these trees since it may allows to share between two trees the processing of some common parts[2]. The shared representation of all parse trees is called *shared parse forest*, or just *parse forest*.

To analyze how two trees can share a (*connected*) part, we first notice that such a part may be isolated by cutting the tree along an edge (or arc) as in figure 5. this actually give us two parts: a *subtree* and a *context* (cf. figure 5). Either of these two parts may be shared in forests representing two trees. When a subtree is the same for two trees, it may be shared as shown in figure 7. When contexts are equal and may thus be shared, we get the structure depicted in figure 8.

The sharing of context actually corresponds to ambiguities in the analyzed sentence: the ellipse in figure 8 contains the head nodes for two distinct parses of the same subsentence v, that both recognize v in the same non-terminal category NT. Each head node is labelled with the (number of) the rule used to decompose v into constituents in that parse, and the common syntactical category labels the top of the ellipse. Not accidentally, this structure is precisely the structure of the OR-nodes we used to represent CF grammars. Indeed, an ambiguity is nothing but a choice between two possible parses of the same sentence fragment v as the same syntactic category NT.

Using a combination of these two forms of sharing, the two parse trees of figures 3 and 4 may be merged into the shared parse forest[3] of figure 6. Note that, for this simple example, the only

[2]The direct production of such shared representation by parsing algorithms also corresponds to sharing in the parsing computation [Tom-87, Lan-74, BilL-88].

[3]This graphical representation of shared forests is not original: to our knowledge it was first used by Tomita [Tom-87]. However, we believe that its comparative understanding as context sharing, as AND-OR tree

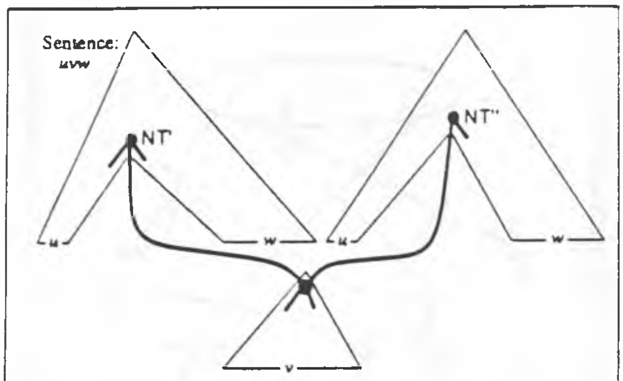


Figure 7: Two parses sharing a subtree

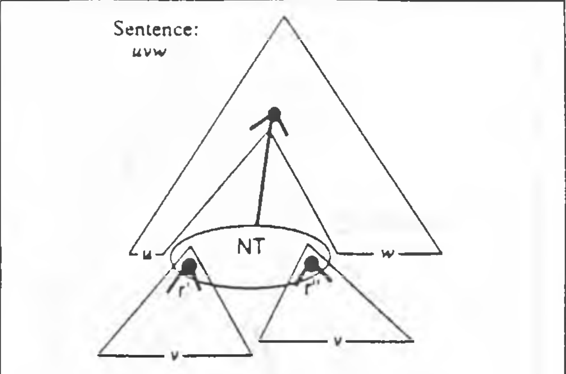


Figure 8: Two parses sharing a context

context being shared is the empty outer context of the two possible parse tree, that still states that a proper parse tree must belong to the syntactic category S.

In this representation we keep our double labelling of parse tree nodes with both the non-terminal category and the rule used to decompose it into its constituents. As indicated above, ambiguities are represented with context sharing, i.e. by OR-nodes that are the exact equivalent of those of figure 2. Hence *a shared parse forest is an AND-OR graph*[4]. Note however that the same rule (resp. non-terminal) may now label several AND-nodes (resp. OR-nodes) of the shared parse forest graph.

If we make the labels distinct, for example by indexing them so as not to lose their original information, we can then read the shared forest graph of a sentence s as a grammar $\mathcal{F}_s$. The language of this grammar contains only the sentence s, and it gives s the same syntactic structure(s) — i.e. the same parse tree(s) and the same ambiguities — as the original grammar, up to the above renaming of labels.

2.3 Parse forests for incomplete sentences

Our view of parsing may be extended to the parsing of incomplete sentences [Lan-88a].

An example of incomplete sentence is "... `see` ... `mirror`". Assuming that we know that the first hole stands for a single missing word, and that the second one stands for an arbitrary number of words, we can represent this sentence by the sequence "? v * n". The convention is that "?" stands for one unknown word, and "*" for any number of them.

Such an incomplete sentence s may be understood as defining a sublanguage $\mathcal{L}_s$ which contains all the correct sentences matching s. Any parse tree for a sentence in that sublanguage may then be considered a possible parse tree for the incomplete sentence s. For example, the sentences "`I see a man with a mirror`" and "`You see a mirror`" are both in the sublanguage of the incomplete sentence above. Consequently, the two parse trees of figures 3 and 4 are possible parse trees for this sentence, along with many others.

or as grammar has never been presented. Context sharing is called *local ambiguity packing* by Tomita.

[4]This graph may have cycles for infinitely ambiguous sentences when the grammar of the language is itself cyclic.

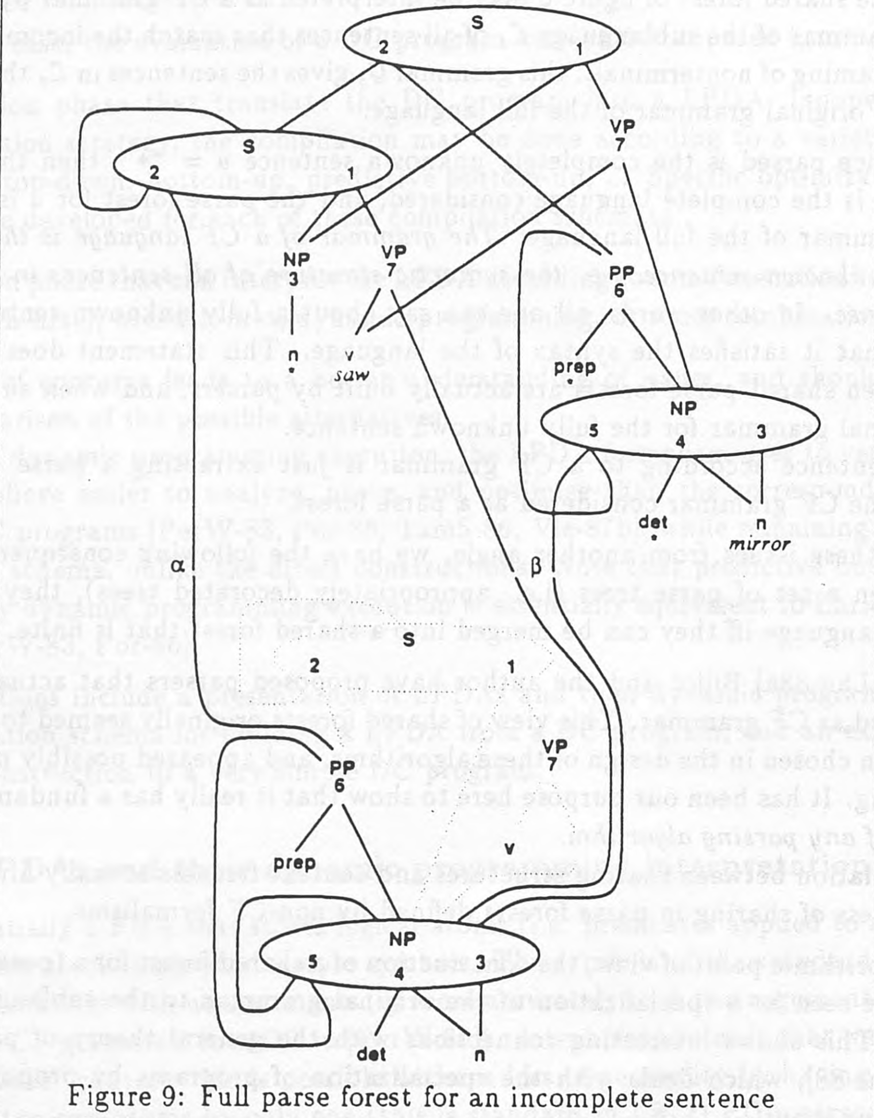


Figure 9: Full parse forest for an incomplete sentence

All parse trees for the sentence $s =$ "? v $*$ n" may be merged into a shared parse forest that is represented in figure 9.

The graph of this forest has been divided into two parts by the horizontal grey line $\alpha - \beta$.

The terminal labels underscored with a "$*$" represent any word in the corresponding terminal category. This is also true for all the terminal labels in the bottom part of the graph.

The forest fragment below the horizontal line is a (closed) subgraph of the original grammar of figure 2 (which we have completed in grey to emphasize the fact). It corresponds to parse trees of constituents that are completely undefined, within their syntactical categories, and may thus be any tree in that category that the grammar can generate. This occurs once in the forest for non-terminal PP at arc marked α and twice for NP at arcs marked β.

This bottom part of the graph brings no new information (it is just the part of the original grammar reachable from nodes PP and NP). Hence the forest could be simplified by eliminating this bottom subgraph, and labelling the end node of the α (resp. β) arc with PP$*$ (resp. NP$*$), meaning an arbitrary PP (resp. NP) constituent.

The complete shared forest of figure 6 may be interpreted as a CF grammar $\mathcal{G}_s$. This grammar is precisely a grammar of the sublanguage $\mathcal{L}_s$ of all sentences that match the incomplete sentence s. Again, up to renaming of nonterminals, this grammar $\mathcal{G}_s$ gives the sentences in $\mathcal{L}_s$ the same syntactic structure as the original grammar of the full language.

If the sentence parsed is the completely unknown sentence u = "*", then the corresponding sublanguage $\mathcal{L}_u$ is the complete language considered, and the parse forest for u is quite naturally the original grammar of the full language: *The grammar of a CF language is the parse-forest of the completely unknown sentence, i.e. the syntactic structure of all sentences in the language, in a non-trivial sense.* In other words, all one can say about a fully unknown sentence assumed to be correct, is that it satisfies the syntax of the language. This statement does take a stronger signification when shared parse forests are actually built by parsers, and when such a parser does return the original grammar for the fully unknown sentence.

Parsing a sentence according to a CF grammar is just extracting a parse tree fitting that sentence from the CF grammar considered as a parse forest.

Looking at these issues from another angle, we have the following consequence of the above discussion: given a set of parse trees (i.e. appropriately decorated trees), they form the set of parses of a CF language iff they can be merged into a shared forest that is finite.

In [BilL-88, Lan-88a] Billot and the author have proposed parsers that actually build shared forests formalized as CF grammar. This view of shared forests originally seemed to be an artifact of the formalization chosen in the design of these algorithms, and appeared possibly more obfuscatory than illuminating. It has been our purpose here to show that it really has a fundamental character, *independently of any parsing algorithm.*

This close relation between sharing structures and context-freeness actually hints to limitations of the effectiveness of sharing in parse forests defined by non-CF formalisms.

From an algorithmic point of view, the construction of a shared forest for a (possibly incomplete) sentence may be seen as a specialization of the original grammar to the sublanguage defined by that sentence. This shows interesting connections with the general theory of partial evaluation of programs [Fut-88], which deals with the specialization of programs by propagation of known properties of their input.

In practice, the published parsing algorithms do not always give shared forest with maximum sharing. This may result in forests that are larger or more complex, but does not invalidate our presentation.

3 Horn Clauses

The PDA based compilation approach proved itself a fruitful theoretical and experimental support for the analysis and understanding of general CF parsing à la Earley. In accordance with our strategy of uniform study of the "Horn continuum", we extended this approach to general Horn clauses, i.e. DC programs.

This lead to the definition of the *Logical Push-Down Automaton (LPDA)* which is an operational engine intended to play for Horn clauses the same role as the usual PDA for CF languages. Space limitations prevent giving here a detailed presentation of LPDAs, and we only sketch the underlying ideas. More details may be found in [Lan-88b, Lan-88].

As in the CF case, the evaluation of a DC program may be decomposed into two phases:

- a compilation phase that translate the DC program into a LPDA. Independently of the later execution strategy, the compilation may be done according to a variety of evaluation schemata: top-down, bottom-up, predictive bottom-up, ... Specific optimization techniques may also be developed for each of these compilation schemata.
- an execution phase that can interpret the LPDA according to some execution technique: back-track (depth-first), breadth-first, dynamic programming, or some combination [TamS-86].

This separation of concerns leads to a better understanding of issues, and should allow a more systematic comparison of the possible alternatives.

In the case of dynamic programming execution, the LPDA formalism uses to very simple structures that we believe easier to analyze, prove, and optimize than the corresponding direct constructions on DC programs [PerW-83, Por-86, TamS-86, Vie-87b], while remaining independent of the computation schema, unlike the direct constructions. Note that predictive bottom-up compilation followed by dynamic programming execution is essentially equivalent to Earley deduction as presented in [PerW-83, Por-86].

The next sections include a presentation of LPDAs and their dynamic programming interpretation, a compilation schema for building a LPDA from a DC program, and an example applying this top-down construction to a very simple DC program.

3.1 Logical PDAs and their dynamic programming interpretation

A LPDA is essentially a PDA that stores logical atoms (i.e. predicates applied to arguments) and substitutions on its stack, instead of simple symbols. The symbols of the standard CF PDA stack may be seen as predicates with no arguments (or more accurately with two argument similar to those used to translate CF grammars into DC in [PerW-80]). A technical point is that we consider PDAs without "finite state" control: this is possible without loss of generality by having pop transitions that replace the top two atoms by only one (this is standard in LR(k) PDA parsers[AhoU-72]).

Formally a LPDA $\mathcal{A}$ is a 6-tuple: $\mathcal{A} = (\mathbf{X}, \mathbf{F}, \mathbf{\Delta}, \overset{\circ}{\$}, \$_f, \Theta)$
where $\mathbf{X}$ is a set of variables, $\mathbf{F}$ is a set of functions and constants symbols, $\mathbf{\Delta}$ is a set of stack predicate symbols, $\overset{\circ}{\$}$ and $\$_f$ are respectively the initial and final stack predicates, and Θ is a finite set of *transitions* having one of the following three forms:

horizontal transitions:	$B \mapsto C$	-- replace B by C on top of stack
push transitions:	$B \mapsto CB$	-- push C on top of former stack top B
pop transitions:	$BD \mapsto C$	-- replace BD by C on top of stack

where B, C and D are $\mathbf{\Delta}$-atoms, i.e. atoms built with $\mathbf{\Delta}$, $\mathbf{F}$ and $\mathbf{X}$.

Intuitively (and approximately) a pop transition $BD \mapsto C$ is applicable to a stack configuration with atoms A and A′ on top, iff there is a substitution s such that $Bs = As$ and $Ds = A's$. Then A and A′ are removed from the stack and replaced by Cs, i.e. the atom C to which s has been applied.

Things are similar for other kinds of transitions. Of course a LPDA is usually non-deterministic w.r.t. the choice of the applicable transition.

In the case of dynamic programming interpretations, all possible computation paths are explored, with as much sub-computation sharing as possible. The algorithm proceeds by building a collection of *items* (analogous to those of Earley's algorithm) which are pairs of atoms. An item $<\mathrm{A}\ \mathrm{A}'>$ represents a stack fragment of two consecutive atoms [Lan-74, Lan-88a]. If another item $<\mathrm{A}'\ \mathrm{A}''>$ was also created, this means that the sequence of atoms $\mathrm{AA}'\mathrm{A}''$ is to be found in some possible stack configuration, and so on (*up to the use of substitutions, not discussed here*). The computation is initialized with an initial item $\overset{\circ}{U} = <\ \overset{\circ}{\$}\ \dashv\ >$. New items are produced by applying the LPDA transitions to existing items, until no new application is possible (an application may often produce an already existing item). The computation terminates under similar conditions as specialized algorithms [PerW-83, TamS-86, Vie-87b]. If successful, the computation produces one or several *final items* of the form $<\$_f\ \overset{\circ}{\$}\ >$, where the arguments of $\$_f$ are an answer substitution of the initial DC program. In a parsing context, one is usually interested in obtaining parse-trees rather than "answer substitutions". A parse tree is here a proof tree corresponding to the original DC program. Such proof trees may be obtained by the same techniques that are used in the case of CF parsing [Lan-74, BilL-88, Bil-88], and that actually interpret the items and their relations as a shared parse forest structure.

Substitutions are applied to items as follows (we give as example the most complex case): a pop transition $\mathrm{BD} \longmapsto \mathrm{C}$ is applicable to a pair of items $<\mathrm{A}\ \mathrm{A}'>$ and $<\mathrm{E}\ \mathrm{E}'>$, iff there is a unifier s of $<\mathrm{A}\ \mathrm{A}'>$ and $<\mathrm{B}\ \mathrm{D}>$, and a unifier s' of $\mathrm{A}'s$ and E. This produces the item $<\mathrm{C}ss'\ \mathrm{E}'s'>$.

3.2 Top-down compilation of DC programs into LPDAs

Given a DC program, *many different compilation schemata may be used to build a corresponding LPDA* [Lan-88]. We give here a very simple and unoptimized top-down construction. The DC program to be compiled is composed of a set of clauses γ_k: $\mathrm{A}_{k,0}$:- $\mathrm{A}_{k,1}, \ldots, \mathrm{A}_{k,n_k}$, where each $\mathrm{A}_{k,i}$ is a logical literal. The query is assumed to be the head literal $\mathrm{A}_{0,0}$ of the first clause γ_0.

The construction of the top-down LPDA is based on the introduction of new predicate symbols $\nabla_{k,i}$, corresponding to positions between the body literals of each clause γ_k. The predicate $\nabla_{k,0}$ corresponds to the position before the leftmost literal, and so on. Literals in clause bodies are refuted from left to right. The presence of an instance of a position literal $\nabla_{k,i}(\mathrm{t}_k)$ in the stack indicates that the first i subgoals corresponding to the body of some instance of clause γ_k have already been refuted. The argument bindings of that position literal are the partial answer substitution computed by this partial refutation.

For every clause γ_k: $\mathrm{A}_{k,0}$:- $\mathrm{A}_{k,1}, \ldots, \mathrm{A}_{k,n_k}$, we note t_k the vector of variables occurring in the clause. Recall that $\mathrm{A}_{k,i}$ is a literal using some of the variables in γ_k, while $\nabla_{k,i}$ is only a predicate which needs to be given the argument vector t_k to become the literal $\nabla_{k,i}(\mathrm{t}_k)$.

Then we can define the top-down LPDA by the following transitions:

1. $\overset{\circ}{\$} \longmapsto \nabla_{0,0}(t_0)\ \overset{\circ}{\$}$
2. $\nabla_{k,i}(t_k) \longmapsto A_{k,i+1}\ \nabla_{k,i}(t_k)$ — *for every clause γ_k and for every position i in its body: $0 \leq i < n_k$*
3. $A_{k,0} \longmapsto \nabla_{k,0}(t_k)$ — *for every clause γ_k*
4. $\nabla_{k,n_k}(t_k)\,\nabla_{k',i}(t_{k'}) \longmapsto \nabla_{k',i+1}(t_{k'})$[5] — *for every pair of clauses γ_k and $\gamma_{k'}$ and for every position i in the body of $\gamma_{k'}$: $0 \leq i < n_{k'}$*

The final predicate of the LPDA is the stack predicate ∇_{0,n_0} which corresponds to the end of the body of the first "query clause" of the DC program. The rest of the LPDA is defined accordingly.

The following is an informal explanation of the above transitions:

1. *Initialization*: We require the refutation of the body of clause γ_0, i.e. of the query.
2. *Selection of the leftmost remaining subgoal*: When the first i literals of clause γ_k have been refuted, as indicated by the position literal $\nabla_{k,i}(t_k)$, then select the $i + 1^{st}$ literal $A_{k,i+1}$ to be now refuted.
3. *Selection of clause γ_k*: Having to satisfy a subgoal that is an instance of $A_{k,0}$, eliminate it by resolution with the clause γ_k. The body of γ_k is now considered as a sequence of new subgoals, as indicated by the position literal $\nabla_{k,0}(t_k)$.
4. *Return to calling clause $\gamma_{k'}$*: Having successfully refuted the head of clause γ_k by refuting successively all literals in its body as indicated by position literal $\nabla_{k,n_k}(t_k)$, we return to the calling clause $\gamma_{k'}$ and "increment" its position literal from $\nabla_{k',i}(t_{k'})$ to $\nabla_{k',i+1}(t_{k'})$, since the body literal $A_{k',i+1}$ has been refuted as instance of the head of γ_k.

Backtrack interpretation of a LPDA thus constructed essentially mimics the Prolog interpretation of the original DC program.

3.3 A very simple example

The following example has been produced with a prototype implementation realized by Eric Villemonte de la Clergerie and Alain Zanchetta [VilZ-88].

The definite clause program to be executed is given in figure 11. Note that a search for all solutions in a backtrack evaluator would not terminate.

The solutions found by the computer are:

```
X2 = f(f(a))
X2 = f(a)
X2 = a
```

[5]If $k = k'$ then we rename the variable in $t_{k'}$ since the transition corresponds to the use of two distinct variants of the clause γ_k.

Note also that we need not define such a transition for all triples of integer k k' and i, but only for those triples such that the head of γ_k unifies with the literal $A_{k',i+1}$.

```
********* PUSH Transitions B->BC ***********

   predicate :nabla.2.0
nabla.2.0(X1) -> q(f(X1)) nabla.2.0(X1)

   predicate :nabla.0.0
nabla.0.0(X2) -> q(X2) nabla.0.0(X2)

   predicate :dollar0
dollar0() -> nabla.0.0(X2) dollar0()

********* Horizontal Transitions B->C ******

   predicate :q
q(f(f(a))) -> nabla.1.0()
q(X1) -> nabla.2.0(X1)

   predicate :query
query(X2) -> nabla.0.0(X2)

   predicate :nabla.0.1
nabla.0.1(X2) -> answer(X2)

********* POP Transitions BD->C ************

   predicate :nabla.2.1
nabla.2.1(X1) nabla.0.0(X2) -> nabla.0.1(X2)
nabla.2.1(X4) nabla.2.0(X1) -> nabla.2.1(X1)

   predicate :nabla.1.0
nabla.1.0() nabla.0.0(X2) -> nabla.0.1(X2)
nabla.1.0() nabla.2.0(X1) -> nabla.2.1(X1)

   predicate :nabla.0.1
nabla.0.1(X3) nabla.0.0(X2) -> nabla.0.1(X2)
nabla.0.1(X2) nabla.2.0(X1) -> nabla.2.1(X1)
```

Figure 10: Transitions of the LPDA.

```
Clauses:  q(f(f(a))):-.
          q(X1):-q(f(X1)).
Query:    q(X2)
```

Figure 11: The Definite Clause program.

```
dollar0() , ()()
nabla.0.0(X5) , dollar0()
q(X6) , nabla.0.0(X6)
nabla.2.0(X7) , nabla.0.0(X7)
nabla.1.0() , nabla.0.0(f(f(a)))
q(f(X8)) , nabla.2.0(X8)
nabla.0.1(f(f(a))) , dollar0()
nabla.2.0(f(X9)) , nabla.2.0(X9)
nabla.1.0() , nabla.2.0(f(a))
nabla.2.1(f(a)) , nabla.0.0(f(a))
nabla.0.1(f(a)) , dollar0()
q(f(f(X10))) , nabla.2.0(f(X10)) *
nabla.2.1(f(a)) , nabla.2.0(a)
nabla.2.1(a) , nabla.0.0(a)
nabla.0.1(a) , dollar0()
answer(a) , dollar0()
answer(f(a)) , dollar0()
answer(f(f(a))) , dollar0()
* subsumed by: q(f(X8)),nabla.2.0(X8)
```

Figure 12: Items produced by the dynamic programming interpretation.

These solutions were obtained by first compiling the DC program into an LPDA according to the schema defined in section 3.2, and then interpreting this LPDA with the general dynamic programming algorithm defined in section 3.1.

The LPDA transitions produced by the compilation are in figure 10. The collection of items produced by the dynamic programming computation is given in the figure 12.

In the transitions printout of figure 10, each predicate name `nabla.i.j` stands for our $\nabla_{i,j}$.

According to the construction of section 3.2, the final predicate should be `nabla.0.1`. For better readability we have added a horizontal transition to a final predicate noted `answer`.

4 Other linguistic formalisms

Pereira and Warren have shown in their classical paper [PerW-80] the link between CF grammars and DC programs. A similar approach may be applied to more complex formalisms than CF grammars, and we have done so for Tree Adjoining Grammars (TAG) [Lan-88c].

By encoding TAGs into DC programs, we can specialize to TAGs the above results, and easily build TAG parsers (using at least the general optimization techniques valid for all DC programs). Furthermore, control mechanisms akin to the agenda of chart parsers, together with some finer properties of LPDA interpretation, allow to control precisely the parsing process and produce Earley-like left-to-right parsers, with a complexity $O(n^6)$.

We expect that this approach can be extended to a variety of other linguistic formalisms, with or without unification of feature structures, such as head grammars, linear indexed grammars, combinatory categorial grammars. This is indeed suggested by the results of of Joshi, Vijay-Shanker and Weir that relate these formalisms and propose CKY or Earley parsers for some of them [VijWJ-87, VijW-89].

The parse forests built in the CF case correspond to proof forests in the Horn case. Such proof forests may be obtained by the same techniques that we used for CF parsing [BilL-88]. However it is not yet fully clear how parse trees or derivation trees may be extracted from the proof forest when DC programs are used to encode non-CF syntactic formalisms.

5 Conclusion

Our understanding of syntactic structures and parsing may be considerably enhanced by comparing the various approaches in similar formal terms. Hence we attempt to formally unify the problems in two ways:

— by considering all formalisms as special cases of Horn clauses

— by expressing all parsing strategies with a unique operational device: the pushdown automaton.

Systematic formalization of problems often considered to be pragmatic issues (e.g. parse forests) has considerably improved our understanding and has been an important success factor.

The links established with problems in other areas of computer science (e.g. partial evaluation, database recursive queries) could be the source of interesting new approaches.

References

[AhoU-72] Aho, A.V.; and Ullman, J.D. 1972 *The Theory of Parsing, Translation and Compiling.* Prentice-Hall, Englewood Cliffs, New Jersey.

[Bil-88] Billot, S. 1988 *Analyseurs Syntaxiques et Non-Déterminisme.* Thèse de Doctorat. Université d'Orléans la Source. Orléans (France).

[BilL-88] Billot, S.; and Lang, B. 1989 The structure of Shared Forests in Ambiguous Parsing. *Proc. of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver (British Columbia), 143-151. Also INRIA Research Report 1038.

[Coh-88] Cohen, J. 1988 A View of the Origins and Development of Prolog. *Communications of the ACM* 31(1) :26-36.

[Col-78] Colmerauer, A. 1978 Metamorphosis Grammars. in *Natural Language Communication with Computers*, L. Bolc ed., Springer LNCS 63. First appeared as *Les Grammaires de Métamorphose*, Groupe d'Intelligence Artificielle, Université de Marseille II, 1975.

[DeR-71] DeRemer, F.L. 1971 Simple LR(k) Grammars. *Communications ACM* 14(7): 453-460.

[Fut-88] Futamura, Y. (ed.) 1988 Proceedings of the Workshop on Partial Evaluation and Mixed Computation. *New Generation Computing* 6(2,3).

[Lan-74] Lang, B. 1974 Deterministic Techniques for Efficient Non-deterministic Parsers. *Proc. of the 2nd Colloquium on Automata, Languages and Programming*, J. Loeckx (ed.), Saarbrücken, Springer Lecture Notes in Computer Science 14: 255-269.
Also: Rapport de Recherche 72, IRIA-Laboria, Rocquencourt (France).

[Lan-88a] Lang, B. 1988 Parsing Incomplete Sentences. *Proc. of the 12th Internat. Conf. on Computational Linguistics (COLING'88)* Vol. 1 :365-371, D. Vargha (ed.), Budapest (Hungary).

[Lan-88b] Lang, B. 1988 Datalog Automata. *Proc. of the 3rd Internat. Conf. on Data and Knowledge Bases*, C. Beeri, J.W. Schmidt, U. Dayal (eds.), Morgan Kaufmann Pub., pp. 389-404, Jerusalem (Israel).

[Lan-88] Lang, B. 1988 *Complete Evaluation of Horn Clauses: an Automata Theoretic Approach.* INRIA Research Report 913.

[Lan-88c] Lang, B. 1988 *The Systematic Construction of Earley Parsers: Application to the Production of $\mathcal{O}(n^6)$ Earley Parsers for Tree Adjoining Grammars.* In preparation.

[PerW-80] Pereira, F.C.N.; and Warren, D.H.D. 1980 Definite Clause Grammars for Language Analysis — A Survey of the Formalism and a Comparison with Augmented Transition Networks. *Artificial Intelligence* 13: 231-278.

[PerW-83] Pereira, F.C.N.; and Warren, D.H.D. 1983 Parsing as Deduction. *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*: 137-144, Cambridge (Massachusetts).

[Por-86] Porter, H.H. 3rd 1986 *Earley Deduction*. Tech. Report CS/E-86-002, Oregon Graduate Center, Beaverton (Oregon).

[TamS-86] Tamaki, H.; and Sato, T. 1986 OLD Resolution with Tabulation. *Proc. of 3rd Internat. Conf. on Logic Programming*, London (UK), Springer LNCS 225: 84-98.

[Tom-85] Tomita, M. 1985 *An Efficient Context-free Parsing Algorithm for Natural Languages and Its Applications*. Ph.D. thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania.

[Tom-87] Tomita, M. 1987 An Efficient Augmented-Context-Free Parsing Algorithm. *Computational Linguistics* 13(1-2): 31-46.

[Vie-87b] Vieille, L. 1987 *Recursive Query Processing: The power of Logic*. Tech. Report TR-KB-17, European ComputerIndustry Research Center (ECRC), Munich (West Germany).

[VijWJ-87] Vijay-Shankar, K.; Weir, D.J.; and Joshi, A.K. 1987 Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. *Proceedings of the 25rd Annual Meeting of the Association for Computational Linguistics*: 104-111, Stanford (California).

[VijW-89] Vijay-Shankar, K.; and Weir, D.J. 1989 Recognition of Combinatory Categorial Grammars and Linear Indexed Grammars. *These proceedings*.

[VilZ-88] Villemonte de la Clergerie, E.; and Zanchetta, A. 1988 *Evaluateur de Clauses de Horn*. Rapport de Stage d'Option, Ecole Polytechnique, Palaiseau (France).

Head-Driven Bidirectional Parsing: A Tabular Method

Giorgio Satta (°)(°°), Oliviero Stock (°°)

(°) Università di Padova, via Belzoni 7, 35131 Padova, Italy
(°°) Istituto per la Ricerca Scientifica e Tecnologica, 38050 Povo, Trento, Italy

1. Introduction

Tabular methods for context-free language analysis [Graham and Harrison, 1976, Graham *et al.*, 1980], and in particular Earley's Algorithm [Earley, 1970], can be considered a major reference for natural language parsing. Even if independently conceived, Earley's Algorithm constitutes the basis for Chart parsing [Kay, 1980, Kaplan, 1973].

One basic aspect of known tabular methods, i.e. that the analysis proceedes monodirectionally, is a relevant limitation, that, although reasonable for artificial languages, seems reductive for natural language. A strong reason for a bidirectional approach within natural language analysis is that modern theories of grammar emphasize the role of a particular element inside each constituent (phrase), called the head; this element carries categorial as well as thematic information about other elements within the constituent. It turns out that the acceptability and the general skeleton of each constituent, crucially depend on such information. More concretely, a number of possible partial interpretations would be pruned out earlier, on the basis of functional information attached to the head, resulting in greater efficiency.

Some recent works in the framework of Chart parsing [Steel and De Roeck, 1987, Stock *et al.*, 1989] have pointed out the importance of bidirectionality for natural language analysis. Another work that deals with some form of bidirectionality [Bossi *et al.*, 1983] can be found in the formal language literature, though the analysis given there presupposes Chomsky normal form grammars.

In this paper we shall introduce a tabular method coinceived for bidirectional context-free parsing, discuss some of its relevant properties and through an example give an idea of how the algorithm works.

2. Definitions

Assume a context-free grammar $G=(N, \Sigma, P, S)$, where N is the finite set of all non-terminal symbols, Σ is the set of terminal symbols, P is a finite set of productions, and $S \in N$ is the start symbol. L(G) represents the language generated by the grammar G. The productions in P are numbered from 1 to $|P|$[1], and are all of form $D_p \rightarrow C_{p,1} \ldots C_{p,\pi(p)}$,

[1] The notation $|P|$ here indicates the cardinality of set P.

where π is a function defined over the set $\{1\dots|P|\}$ and that takes values in the set Z^+ (the set of positive integers). In the following, the natural number p often will be used instead of the production associated with it. Without loss of generality, here it is assumed that the grammar G is in *e-free* form (see [Aho and Ullman, 1972:147]); a more general formulation of the algorithm does not lead to the loss of the properties shown here.

A function τ is defined over the set $\{1\dots|P|\}$ and it takes values in Z^+. This function indicates, for every production p in P, a position in the right–hand side of the production, occupied by a symbol in $N\cup\Sigma$. This position is called the *head position*, and the corresponding symbol is said to be *in the head position* for production p. Every time, during the analysis, a symbol is recognized that is in head position for some production p, the presence of the symbol D_p relative to production p is then locally hypothesized.

DEFINITION 2.1

A *state* is defined to be any quadruple $[p, ldot, rdot, m]$, with $1 \leq p \leq |P|$, $0 \leq ldot < rdot \leq \pi(p)$, $m \in \{ -, lm, rm\}$.

The component p indicates the corresponding production in P; the components *ldot* and *rdot* represent two distinct positions, one after the other, in the right-hand side of production p. The component m is a simple indicator: $m=lm$ indicates that the value of *ldot* cannot be further diminished, even if greater than zero, while $m=rm$ indicates that the value of *rdot* cannot be increased further, even if it is less than $\pi(p)$. Note that, by definition, one limitation excludes the other. The value "-" is used for the indicator m in the absence of both the limitations just described. The use of the index m, as it will be shown, prevents the duplication of "partial analyses" for substrings of w. Every state $s=[p, ldot, rdot, m]$ may be understood to be a partial analysis relative to production p, for which the constituents $C_{p,ldot+1} \dots C_{p,rdot}$, belonging to the right-hand side, have been recognized. In the following, for convenience, the states will often be referred to in these terms. The symbol I_s denotes the set of all states.

DEFINITION 2.2

The function F is defined as follows:

$F: N\cup\Sigma \rightarrow \mathcal{P}(I_s)$

$F(X)=\{s=[p, ldot, ldot+1, -] \mid X=C_{p,ldot+1}, \tau(p)=ldot+1\}$.

The set F(X) therefore contains all the states indicating partial analyses of productions in which the symbol X occupies the head position.

DEFINITION 2.3

An equivalence relation Q in $I_s \times I_s$ is defined so that for two generic states $s=[p, ldot, rdot, m]$ and $s'=[p', ldot', rdot', m']$, sQs' holds if and only if $p=p'$, $ldot=ldot'$ and $rdot=rdot'$.

3. The Algorithm

A *recognizer* is an algorithm capable of accepting a generic string $w \in L(G)$ for a particular grammar of interest G. In all other cases, the string w is refused. A *parser*, instead, is an algorithm that can solve the problem of whether or not w belongs to L(G) and is also able to indicate the possible derivation trees[2] for every $w \in L(G)$. In this section, a recognizer algorithm for context-free languages is presented. The use of a simple algorithm able to reconstruct the derivation trees by interpreting the recognition matrix T (see for example [Graham et Harrison 1983]) is sufficient to obtain a parser algorithm.

The algorithm uses a matrix T of size $(n+1)\times(n+1)$; each component $t_{i,j}$ of this matrix takes values in the set $\mathcal{P}(I_s)$, and is initialized with as empty set. The presentation of the recognition algorithm is preceded by a schematic illustration of the computation involved.

The algorithm inserts into the recognition matrix T each state s that indicates a partial analysis previously obtained for the generic substring ${}_iw_j$. There is a one to one correspondence between the indicies of the analyzed substring ${}_iw_j$ and the indices of the component $t_{i,j}$, in which state s has been inserted. The algorithm then processes each state, combining it with nearby states in an effort to extend the portions of the string dominated by these states. When the analysis relative to a particular state is completed (for both the right and left sides), if the constituent obtained is in a head position for some production p in P, a new partial analysis for the production p itself is inserted into matrix T. Note that the algorithm straightforwardly separates the problem of the combination of different states from the problem of control. The algorithm in fact does not specify the order in which the different states must be considered, nor in which order every single state must be expanded in the two opposing sides. To that end, the algorithm uses a variable A which takes values in the set $\mathcal{P}(I_s \times N \times N)$.

ALGORITHM 3.1

Given a context-free grammar $G=(N, \Sigma, P, S)$ in *e-free* form, let $w=a_1 \ldots a_n$, $n>0$, be an input string. Develop a recognition matrix T, of size $(n+1)\times(n+1)$, whose components $t_{i,j}$ are coindexed from 0 to n for both sides.

```
     begin
1.   for i in {1 .. n} do
2.        for s in F(a_i) do
3.             add triple e=(s, i-1, i) to set A only if sQs_q
               does not hold for any triple e_q=(s_q,
               i-1, i)
4.   while A not empty do
```

[2] A *derivation tree* D associated with a string $w \in L(G)$, is a labeled tree formed by all the productions used in the derivation of$w$, representing the correct hierarchic order.

```
5.        extract any element $e=(s, i, j)$ from the set A and
          insert state $s$ in $t_{i,j}$; apply each of the following
          procedures, in any order, to element $e$:
              left-expander(e),
              right-expander(e),
              left-completer(e),
              right-completer(e),
              trigger(e);
6.    if $s=[p, 0, \pi(p), m] \in t_{0,n}$, for some $p \in P$ such that $D_p=S$
7.        then output(true)
8.        else output(error)
      end.
```

The five procedures mentioned above are described in the following.

PROCEDURE 3.1 *Left-expander*

Precondition The procedure is applied only when $e=(s, i, j)$ with $s=[p, ldot, rdot, m]$, $ldot>0$, $m \neq lm$.

Description The following two cases are possible.

Case 1: $C_{p,ldot} \in N$. For every $s'=[p', 0, \pi(p'), m'] \in t_{i',i}$, $i'<i$, such that $D_{p'}=C_{p,ldot}$, the state $s''=[p, ldot\text{-}1, rdot, \text{-}]$ is created and the triple $e'=(s'', i', j)$ is inserted in set A, only if $s''Qs_q$ does not hold for any state s_q in $t_{i',j}$ or for any triple $e_q=(s_q, i', j)$ in A. If at least one state s' is found with the above properties, set $m=rm$ in s.

Case 2: $C_{p,ldot} \in \Sigma$. If $C_{p,ldot}=a_i$, the state $s'=[p, ldot\text{-}1, rdot, \text{-}]$ is created and the triple $e'=(s', i\text{-}1, j)$ is inserted into set A, only if $s'Qs_q$ does not hold for any state s_q in $t_{i-1,j}$ or for any triple $e_q=(s_q, i\text{-}1, j)$ in A. If $C_{p,ldot}=a_i$, set $m=rm$ in s.

This procedure is applied only if state s can be extended leftward ($ldot>0$) and only if it has not already been extended rightward ($m \neq lm$), that is, if it is not subsumed to the right by a more updated state. There are two cases, depending upon whether the left-hand expansion symbol is a terminal symbol or not. If $C_{p,ldot}$ is a non-terminal symbol, the search proceeds to the left of state s, to any state s' (adjacent), that corresponds to a completed analysis ($ldot'=0$, $rdot'=\pi(p')$) of a constituent usable by state s ($D_{p'}=C_{p,ldot}$). If successful, the analysis is extended in correspondence with state s, including the constituent found nearby; state s then is marked with $m=rm$, since this has been subsumed on the left by a more updated state. If $C_{p,ldot}$ is, instead, a terminal symbol, and if $C_{p,ldot}=a_i$, an extension of the analyses corresponding to state s is made, including the terminal symbol a_i. Still, state s is marked with $m=rm$ for the same reasons as in Case 1. Furthermore, note that Procedure 3.1 never duplicates the triples in A, nor the states belonging to the same component of recognition matrix T.

PROCEDURE 3.2 *Right-expander*

Precondition The procedure is applied only when $e=(s, i, j)$ with $s=[p, ldot, rdot, m]$, $rdot<\pi(p)$, $m \neq rm$.

Description There are the following two cases.

Case 1: $C_{p,rdot+1} \in N$. For every $s'=[p', 0, \pi(p'), m'] \in t_{j,j'}$, $j'>j$, such that $D_{p'}=C_{p,rdot+1}$, state $s''=[p, ldot, rdot+1, -]$ is created and triple $e'=(s'', i, j')$ is inserted into set A, only if $s''Qs_q$ does not hold for any state s_q in $t_{i,j'}$ or for any triple $e_q=(s_q, i, j')$ in A. If at least one state s' has been found with the properties described above, set $m=lm$ in s.

Case 2: $C_{p,rdot+1} \in \Sigma$. If $C_{p,rdot+1}=a_{j+1}$, the state $s'=[p, ldot, rdot+1, -]$ is created and the triple $e'=(s', i, j+1)$ is inserted in A, only if $s'Qs_q$ does not hold for any state s_q in $t_{i,j+1}$. If $C_{p,ldot}=a_{j+1}$, set $m=lm$ in s.

This procedure is symmetric to the *left-expander* procedure, so the explanation is omitted.

PROCEDURE 3.3 *Left-completer*

Precondition The procedure is applied only when $e=(s, i, j)$, with $s=[p, 0, \pi(p), m]$.
Description For every $s'=[p', ldot', rdot', m'] \in t_{i',i}$, $i'<i$, $rdot'<\pi(p')$, $m' \neq rm$. such that $D_p=C_{p',rdot'+1}$, state $s''=[p, ldot', rdot'+1, -]$ is created and the triple $e'=(s'', i', j)$ is inserted in set A only if $s''Qs_q$ does not hold for any state s_q in $t_{i',j}$ or for any triple $e_q=(s_q, i', j)$ in A. Furthermore, set $m'=lm$ for every s' found.

This procedure is applied whenever the analysis of a constituent D_p has been completed through a state $s=[p, 0, \pi(p), m]$. It proceeds by searching leftward of state s for any adjacent state s' that has not yet been subsumed to the left ($m' \neq rm$) and is able to "expand" state s ($D_p=C_{p',rdot'+1}$). If successful, an extension of the analysis corresponding to s' is carried out, including the constituent D_p. State s' is then marked with $m=lm$, since it has now been subsumed on the right by a more updated state. Again, note that the procedure never duplicates triples in A, nor states belonging to the same component of the recognition matrix T.

PROCEDURE 3.4 *Right-completer*

Precondition The procedure is applied only when $e=(s, i, j)$, with $s=[p, 0, \pi(p), m]$.

Description For every $s'=[p', ldot', rdot', m'] \in t_{j,j'}$, $j'>j$, $ldot'>0$, $m' \neq lm$, such that $D_p=C_{p',ldot'}$, state $s''=[p, ldot'-1, rdot', -]$ is created and the triple $e'=(s'', i,j')$ is inserted in set A only if $s''Qs_q$ does not hold true for any state s_q in $t_{i,j'}$ or for any triple $e_q=(s_q, i, j')$ in A. Furthermore, set $m'=rm$ for every s' found.

This procedure is symmetric to the *left-completer* procedure, so the explanation is omitted.

PROCEDURE 3.5 *Trigger*

Precondition The procedure is applied only when $e=(s, i, j)$, with $s=[p, 0, \pi(p), m]$.

Description For every $s\in F(D_p)$, insert the triple $e=(s, i, j)$ in set A only if $s\mathcal{Q}s_q$ does not hold for any state s_q in $t_{i,j}$ or for any triple $e_q=(s_q, i, j)$ in A.

The procedure is applied whenever the analysis of a particular constituent has been completed and this constituent occupies the head position in some production p. In this case a new state corresponding to a partial analysis for production p is created, including the head. Once again, note that the procedure never duplicates triples in A, nor states belonging to the same component of the recognition matrix T.

4. Some Formal Properties of the Algorithm

In this section the most interesting properties of Algorithm 3.1 are stated. For a formal proof of what follows refer to [Satta and Stock, 1989b]. Four major properties have been grouped under Invariant 4.1 below. Note that soundness and completeness for Algorithm 3.1 follow straightforwardly from statements *(i)* and *(ii)* in Invariant 4.1.

INVARIANT 4.1

(i) $s=[p, ldot, rdot, m]\in t_{i,j}$ *only if* $C_{p,ldot+1} \cdots C_{p,rdot} \overset{\bullet}{\Rightarrow} a_{i+1} \cdots a_j$, $i<j$, $ldot+1\leq\tau(p)\leq rdot$;

(ii) $C_{p,ldot+1}\cdots C_{p,rdot} \overset{\bullet}{\Rightarrow} a_{i+1}\cdots a_j$, $i<j$, $ldot+1\leq\tau(p)\leq rdot$ *only if* a quadruple $h=[h_1, h_2, h_3, h_4]$, $h_q\geq 0$, $1\leq q\leq 4$ exists such that $s=[p, ldot-h_1, rdot+h_2, m]\in t_{i-h_3,j+h_4}$;

(iii) $s=[p, ldot, rdot, lm]\in t_{i,j}$ *only if* $s'=[p, ldot, rdot+1, m']\in t_{i,j'}$, $j'>j$;

(iv) $s=[p, ldot, rdot, rm]\in t_{i,j}$ *only if* $s'=[p, ldot-1, rdot, m']\in t_{i',j}$, $i'<i$.

Algorithm 3.1 allows the extension of a state to both the left and right sides. This possibility, if not carefully controlled, can lead to the duplication of an analysis, in the following way. If a state s, relative to a partial analysis for a constituent C_s, is independently extended to both sides, it would lead to the introduction of two partially overlapping states, s' and s'', for the same analysis. The completion of s' and s'' then would lead to the duplication of constituent C_s. The algorithm presented here uses the index m, associated with each state, so as to avoid partial overlapping for two (partial) analyses of the same constituent. Formally, we define the partial overlapping relation as follows.

DEFINITION 4.1 *Partial Overlapping Relation*

Two states $s=[p, ldot, rdot, m]\in t_{i,j}$ and $s'=[p, ldot', rdot', m']\in t_{i',j'}$ are *partially overlapped* ($s\mathcal{D}s'$) iff $i<i'<j<j'$, $ldot<ldot'<rdot<rdot'$, and, furthermore, s subsumes the same constituents $C_{p,ldot'+1}\cdots C_{p,rdot}$ subsumed in s'.

Note that for two states $s=[p, ldot, rdot, m]$ and $s'=[p, ldot', rdot', m']$ such that $s\mathcal{D}s'$, it always holds that $ldot'<\tau(p)\leq rdot$. The following theorem can now be stated.

THEOREM 4.1

Algorithm 3.1 never generates two states s and s' such that $s\mathcal{D}s'$.

The following result regards space and time complexity for Algorithm 3.1. Such a result is intended for a Random Access Machine model of computation.

THEOREM 4.2

Algorithm 3.1 requires an amount of space $O(n^2)$ and an amount of time $O(n^3)$, where n is the length of the input string.

5. A Brief Example

In order to have an insight into Algorithm 3.1, an example regarding a simple computation is given here. Assume an unambiguous context-free grammar G=(N, Σ, P, S), where N={S, A, B}, Σ={a, b, c, d, e}, and P is the production set given as follows:

1:	$S \rightarrow A\ a$,	$\tau(1)=2, \pi(1)=2$;
2:	$S \rightarrow B\ b$,	$\tau(2)=2, \pi(2)=2$;
3:	$A \rightarrow c\ A\ c$,	$\tau(3)=2, \pi(3)=3$;
4:	$A \rightarrow d$,	$\tau(4)=1, \pi(4)=1$;
5:	$B \rightarrow c\ B\ c$,	$\tau(5)=2, \pi(5)=3$;
6:	$B \rightarrow e$,	$\tau(6)=1, \pi(6)=1$.

From Definition 2.2 it follows that:

$F(A) = \{[3, 1, 2, -]\}$; $F(B) = \{[5, 1, 2, -]\}$;
$F(a) = \{[1, 1, 2, -]\}$; $F(b) = \{[2, 1, 2, -]\}$;
$F(d) = \{[4, 0, 1, -]\}$; $F(e) = \{[6, 0, 1, -]\}$;
$F(S) = F(c) = \emptyset$.

A run of Algorithm 3.1 on the string $w=ccеccb$ is simplified by the following steps (the order of application for the five procedures at line 5 is chosen at random).

1) $s_1=[6, 0, 1, -]$ is inserted in $t_{2,3}$ and $s_2=[2, 1, 2, -]$ is inserted in $t_{5,6}$, by line 3;
2) $s_3=[5, 1, 2, -]$ is inserted in $t_{2,3}$ by the *trigger* procedure;
3) $s_4=[5, 0, 2, -]$ is inserted in $t_{1,3}$ and m is set to rm in state s_3, by Case 2 of the *left-expander* procedure;
4) $s_5=[5, 0, 3, -]$ is inserted in $t_{1,4}$ and m is set to lm in state s_4, by Case 2 of the *right-expander* procedure;
5) $s_6=[5, 1, 2, -]$ is inserted in $t_{1,4}$ by the *trigger* procedure;
6) $s_7=[5, 1, 3, -]$ is inserted in $t_{1,5}$ and m is set to lm in state s_6, by Case 2 of the *right-expander* procedure;
7) $s_8=[5, 0, 3, -]$ is inserted in $t_{0,5}$ and m is set to rm in state s_7, by Case 2 of the *left-expander* procedure;

8) s_9=[5, 1, 2, -] is inserted in $t_{0,5}$ by the *trigger* procedure;
9) s_{10}=[2, 0, 2, -] is inserted in $t_{0,6}$ and m is set to rm in state s_2, by the *right-completer* procedure;
10) the algorithm outputs true and then stops.

Note how the setting of the m components in states s_3 and s_6 prevents the expansion of partial analysis at both sides. Though not shown here, in more complicated cases the setting of the m components permits the *left-completer* procedure to combine a state s with the "leftward largest" partial analyses that are adjacent to the left of s, preventing once more partial analysis duplication (*vice versa* for the *right-completer* procedure).

Finally, note that in the above example Algorithm 3.1 has constructed 10 states, while a run of the classic method of Earley on the same string would have constructed 25 states. Furthermore, by defining $\tau(p)=1$, $1 \leq p \leq |P|$, Algorithm 3.1 mimics the *left-corner* strategy as stated in [Wirén, 1987], resulting in the construction of 17 states for the same analysis.

6. Final Remarks

This paper discusses a parsing algorithm that extends bidirectionally the classic tabular methods for context-free language analysis. The algorithm is given for *e-free* form context-free grammars, but it is not difficult to extend it to the general case, for example by employing the same technique used in [Graham *et al.* 1980] in the treatment of empty categories.

With respect to natural language parsing, the presented tabular method is compatible with the well known "Active Chart Parsing" technique, as pointed out in [Satta and Stock 1989a]. Finally, the extension to Earley's Algorithm proposed in [Shieber 1985] for parsing complex-feature-based formalisms, could be equally applicable to the presented approach.

References

[Aho and Ullman, 1972] Aho, A. V., and J. D. Ullman. *The Theory of Parsing, Translation, an Compiling*, vol. 1, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[Bossi *et al.*, 1983] Bossi, A., N. Cocco, and L. Colussi. A divide-and-conquer approach to general context-free parsing. *Information Processing Letters*, 16 - pp. 203-208, 1983.

[Earley, 1970] Earley, J. An Efficient Context-Free Parsing Algorithm. *Communications of the ACM*, 13(2), pp. 94-102, 1970.

[Graham and Harrison, 1976] Graham, S. L., and M. A. Harrison. Parsing of General Context Free Languages. *Advances in Computers*, pp. 77-185 , Academic Press, New York, 1976.

[Graham *et al.*, 1980] Graham, S. L., M. A. Harrison, and W. L. Ruzzo. An Improved Context-Free Recognizer. *ACM Toplas*, 2(3), pp. 415-462, 1980.

[Kaplan, 1973] Kaplan, R. M. A General Syntactic Processor. In: (R. Rustin, ed) *Natural Language Processing*, pp. 193-241, Algorithmics Press, New York, New York, 1973.

[Kay, 1980] Kay, M. Algorithm Schemata and Data Structures in Syntactic Processing. *Tecnical Report CSL-80* Xerox-PARC, Palo Alto, California, 1980.

[Satta and Stock, 1989a] Satta, G., and O. Stock. Formal Properties and Implementation of Bidirectional Charts. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989.

[Satta and Stock, 1989b] Satta, G., and O. Stock. Bidirectional Context-Free Language Parsing within a tabular approach, submitted for publication.

[Shieber, 1985] Shieber, S. M. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. *Proceedings of the 23rd Conference of the Association for Computational Linguistics,* Chicago, Illinois, 1985.

[Steel and De Roeck, 1987] Steel, S. and A. De Roeck. Bidirectional Chart Parsing. *Proceedings of AISB-87,* Edinburgh, Scotland, 1987.

[Stock *et al.*, 1989] Stock, O., R. Falcone, and P. Insinnamo. Bidirectional Charts: a Potential Technique for Parsing Spoken Natural Language. Computer Speech and Language, 3, 1989.

[Wirén, 1987] Wirén, M. A Comparison of Rule-Invocation Strategies in Context-Free Parsing. *Proceedings of the 3rd Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark, 1987.

Head-Driven Parsing

Martin Kay

Xerox Palo Alto Research Center and Stanford University

There are clear signs of a "Back to Basics" movement in parsing and syntactic generation. Our Latin teachers were apparently right. You should start with the main verb. This will tell you what kinds of subjects and objects to look for and what cases they will be in. When you come to look for these, you should also start by trying to find the main word, because this will tell you most about what else to look for.

In the early days of research on machine translation, Paul Garvin advocated the application of what he called the "Fulcrum" method to the analysis of sentences. If he was the last to heed the injunctions of his Latin teacher, it is doubtless because America followed the tradition of rewriting systems exemplified by context-free grammar and this provided no immediate motivation for the notion of the *head* of a construction. The European tradition, and particularly the tradition of Eastern Europe, where Garvin had his roots, tend more towards dependency grammar, but away from that of mathematical formalization which has been the underpinning of computational linguistics.

But the move now is towards linguistic descriptions that put more information in the lexicon so that grammar rules take on a more schematic quality. Little by little, we moved from rules like

```
(1) VP1 -> VP2 NP
      CaseOf(VP2) = Dative
      CaseOf(NP) = Dative
```

to rules that attain greater abstraction through the use of logical variables (or the equivalent), like

```
(2) VP1 -> VP2 NP
    ObjCase(VP2) = Case
    CaseOf(NP) = Case
```

Where the underlined Case is to be taken as the name of a variable. From there, it was a short step to

```
(3) VP1 -> VP2 X
    ComplementOf(VP2) = X
```

or even

```
(4) VP1 -> VP2 X
    ComplementStringOf(VP2) = X
```

Given rule (2), that parser knows what case the noun must have only after it has encountered the verb. Rules (3) and (4), do not even tell it that the complement must be a noun phrase. In (4) we cannot even tell how many complements ther will be. For most parsers, the problem is masked in these examples by the fact that they apply rules from left to right so that the value of the variable X is known by the time it is needed. In rule (4a), the matter is different.

```
(4a) VP1 -> X VP2
    ComplementStringOf(VP2) = X
```

Needless to say, these things have not gone unnoticed, least of all by the participants in this conference. It has been noted, for example, that definite-clause grammars can be adjusted so as to look for heads before complements and adjuncts. If the head of a sentence is a verb phrase, then it is sufficient to write (6) instead of (5).

```
(5) s(Left/Right) :-
      np(Left/Middle),
      vp(Middle/Right).
(6) s(Left/Right) :-
      vp(Middle/Right),
      np(Left/Middle).
```

A rule that expands the verb phrase would be something like (7).

```
(7) vp(Left/Right) :-
        verb(Left/Middle),
        np(Middle/Right).
```

This time, the order is the usual one because the head is on the left[1].

Of course, all this works if `Left`, `Middle`, and `Right` are something like word numbers that provide random access to the parts of the sentence. To make the system work with difference lists, we need something more, for example, as in (8).

```
(8) s(Left/Right) :-
        append(X, Middle, Left),
        vp(Middle/Right), np(Left/Middle).
```

1 We have now moved to the Prolog convention of using caitalized names for variables.

The reason for the addition is that the parser, embodied here in the set of rules themselves, has no way to tell where the verb phrase will begin. It must therefore consider all possible positions in the string, an end which, against all expectation, is accomplished by the `append` predicate. If `append` is not needed when something like word numbers are used, it is because the inevitable search of the string is being quietly conducted by the Prolog system as it searches its data base, rather than being programmed explcitely.

The old-fashioned parser had no trouble finding the beginnings of things because they were always immediately adjacent, either to the boundaries of the sentence, or to another phrase whose position was already known. Given the sentence

I sold my car to a student of African languages whom I met at a party

and given appropriate rules, the head-driven parser will correctly identify "my car" as the direct object of "sold". But it will also consider for this role at least the following:

```
(8)    a student
       a student of African
       a student of African languages
       a student of African languages whom I met
       a student of African languages whom I met at a party
       African
       African languages
       African languages whom I met
       African languages whom I. met at a party
       languages
       languages whom I met
       languages whom I met at a party
       a party
```

It will reject them only when it fails to extend them far enough to the left to meet the right-hand edge of the word "sold". Likewise, the last four entries on the list will be constructed again as possible objects for the preposition "of". As we shall see, this problem is not easy to put to set aside.

Of course, definite-clause grammars have other problems, when interpreted directly by a standard Prolog processor. The most notorious of these is that, in their classical form, they cycle indefinitely when provided with a grammar that involves left recursion. However this can be overcome by using a more appropriate interpreter such as the one given in Appendix A of this paper. It does not touch the question of the additional work that has to be done in parsing a sentence.

Two solutions to the problem suggest themselves immediately. One is to use an undirected bottom-up parsing strategy, and the other is to seek an appropriate adaptation of chart parsing to a directed, head-driven, strategy. The first solution works for the simple reason that the problem we are facing simply does not arise in undirected bottom-up processing. There is no question of finding phrases that are adjacent to, or otherwise positioned relative to, other phrases. The strategy is a purely opportunistic one which finds phrases wherever, and whenever, its control strategy dictates. A simple chart parser with these properties is given in Appendix B. It accepts only unary and binary rules, but this is not a real restriction because these binary rules can function as meta-rules that interpret the more general of the actual grammar according to something like the following scheme. Real rules have a similar format to that used in the program in Appendix A, namely

```
rr(Mother, [L1, L2 ... Ln], Head, [R1, R2 ... Rm])
```

L_1 ... L_n are the non-head (complement) daughters of 'Mother' to the left of the head, and R_1 ... R_m are those to the right. For convenience, we give the ones on the left in the reverse of the order in which they actually appear so that the one nearest to the head is written first. We define the binary rule predicate referred to in the algorithm somewhat as follows;

```
rule(p(Mother, L, Rest), Head, Next) :-
   rr(Mother, L, Head, [Next|Rest]).
 rule(p(Mother, Rest, []), Next, Head) :-
   rr(Mother, [Next|Rest], Head, []).
 rule(p(Mother, L, T), p(Mother, L, [H|T]), H).
 rule(p(Mother), H, p(Mother, [H|T], [])).
```

One special unary rule is required, namely

```
rule(Mother, p(Mother, [], [])).
```

The scheme is reminiscent of categorial grammar. `p(Category, Left, Right)` is a partially formed phrase belonging to the given `Category` which can be completed by adding the items sepecified by the `Left` list on the left, and the `Right` list on the right.

This scheme has a certain elegance in that the parser itself is simple and does not reflect any peculiarities of head-driven grammar. Only the simple meta-rules given above are in any way special. Furthermore, the performance properties of the chart parser are not compromised. On the other hand, this inactive chart parser cannot be extended to make it into an active chart parser in a straightforward manner as our second solution requires. This is the crux of the matter that this paper addresses.

Suppose that the verb has been located that will be the head of a verb phrase, but that it remains to identify one or two objects for it on the right. A standard active chart parser does this by introducing active edges at the vertex to the right of the verb which will build the first object if the material necessary for its construction is available, or comes to be available. As the construction procedes, active edges stretch further and further to the right intil the construction is complete and the corresponding inactive edge is introduced. This works only because the phrase can be built incrementally starting from the left, that is, starting next to the phrase to which it must be adjacent. But this strategy is not open to the head-driven parser which must begin by locating, or constructing the head of the new phrase. The rest of the phrase must then be constructed outwards from the head. We are therefore forced to modify the standard approach.

We propose to enrich the notion of a chart so that instead of simply active and inactive edges, it contains five different types of object. Edges can be *active* and *inactive*, but they can also be *pending* or *current*. This gives four of the five kinds. The fifth we shall refer to simply as a *seek*. It is a record of the fact that phrases with a given label are being sought in a given region of the chart. A seek contains a label and also identifies a pair of vertices in the chart. It is irrelevant at the level of generality of this discussion whether we think of the seek as actually being located in, or on, one of the vertices, or being representable as a transition between them. A condition that the chart is required to maintain is that edges with the same label as that of a seek, both of whose end points lie within the region of the seek, must be current. Edges which are not so situated must be pending. The standard chart regime never calls for information in a chart to change, but that is not the case here. When a new seek is introduced, pending edges are modified to become current as necessary to maintain the above invariant.

The fundamental rule (Henry Thompson's term) of chart parsing is that an action is taken, possibly resulting in the introduction of new edges, whenever the introduction of a particular new edge brings the operative end of an active edge together, at the same vertex, with an end of an inactive edge. If the label on the inactive edge is of the kind that the active edge can consume, a new edge is introduced, possibly provoking new applications of the fundamental rule. The fundamental rule also applies in our enriched charts, but only to current edges–pending edges are ignored by it.

Suppose once again that a verb has been identified and that we are now concerned to find its sisters to the right. The verb can have been found only because there was a seek in existance for verbs covering the region in which it was found, and this, in its turn, will have come about because seeks were extant in that region for higher-level phrases, notably verb phrases. The objects we are now interested to locate must lie entirely in a region bounded on the left by the verb itself and, on the right, by the furthest right-hnd end of a VP seek that includes the verb. Accordingly, a new seek is established for NP's in this region. The immediate effect of this will be to make current any pending edges in that region that are inactive and labeled NP, or active and labeled with a rule that forms NP's.

It remains to discuss how active edges, whether current or pending, are introduced in the first place. The simplest solution seems to be to do this just as it would be in an undirected, bottom-up, parser. Whenever a current inactive edge is introduced, or a pending one becomes current, active edges are introduced, one for each rule that could accept the new item as head. However, these do not become current until a need for them emerges higher in the structure, and this is signaled by the introduction of a seek.

Consider, for example, the sentence *the dog saw the cat* and assume that *dog*, *saw*, and *cat* are nouns, *saw* is also a transitive verb, and that the grammar contains the following rules:

```
rule(s(s(NP, VP)), [np(NP)], vp(VP), []).
rule(vp(vp(V, NP)), [], v(V), [np(NP)]).
rule(np(np(D, N)), [det(D)], n(N), []).
```

The sequence of events involved in parsing the sentence with a parser that follows a simple shift reduce regime, would be as follows:

```
1. Add pending for det(det(the)) from 0 to 1,
    Left = [], Right = []
2. Add pending for n(n(dog)) from 1 to 2, Left = [],
   Right = []
3 .Add edge for v(v(saw)) from 2 to 3, Left = [],
   Right = []
4. Add edge for vp(vp(v(saw),_653)) from 2 to 3,
   Left = [], Right = [np(_653)]
5. Add edge for vp(vp(v(saw),_653)) from 2 to 3,
    Left = [], Right = [s(_653)]
 6. Add pending for n(n(saw)) from 2 to 3, Left = [],
    Right = []
 7. Add pending for det(det(the)) from 3 to 4,
    Left = [], Right = []
 8. Add edge for n(n(cat)) from 4 to 5   Rule = 0 / 0, Left = [],
    Right = []
 9. Add edge for np(np(_690,n(cat))) from 4 to 5,
    Left = [det(_690)], Right = []
10. Add edge for det(det(the)) from 3 to 4,
    Left = [], Right = []
11. Add edge for np(np(det(the),n(cat))) from 3 to 5
     Rule = r4 / 1, Left = [], Right = []
12. Add edge for vp(vp(v(saw),np(det(the),n(cat)))) from 2 to 5,
     Left = [], Right = []
13. Add edge for s(s(_1507,vp(v(saw),np(det(the),n(cat)))))
     from 2 to 5, Left = [np(_1507)], Right = []
14. Add edge for n(n(dog)) from 1 to 2, Left = [],
     Right = []
15. Add edge for np(np(_2014,n(dog))) from 1 to 2,
     Left = [det(_2014)], Right = []
16. Add edge for det(det(the)) from 0 to 1,
     Left = [], Right = []
17. Add edge for np(np(det(the),n(dog))) from 0 to 2,
     Left = [], Right = []
18. Add edge for s(s(np(det(the),n(dog)),vp(v(saw);
     np(det(the),n(cat))))) from 0 to 5, Left = [],
     Right = []

Result = [s(s(np(det(the),n(dog)),vp(v(saw),np(det(the),n(cat)))))],
```

We write `add edge...` when the edge being added is current. Notice that the edge for the word *saw*, construed as a verb, is initially introduced as current, because the goal is to find a sentence and a seek is therefore extant for S, VP, and V, covering the whole string. The N edge for *saw*, however, is pending. In step 4, the active adge is introduced that will consume the object of *saw* when it is found. This introduces a seek for NP and N between vertex 3and the end of the sentence. For this reason, when *cat* is introduced in step 8, it is as a current edge. Notice, however, that *the*, in step 7, is introduced as pending, because it is not the head of a NP. However, the introduction of the active NP edge in step 9 causes the edge for *the* to be made current, and this is what happens in step 10. The active S edge in step 13 activates the search for an NP before the verb so that all the remaining edges are introduced as current. At the end of the process all pending edges have been made current except the one corresponding to the nominal interpretation of `saw`.

The Prolog code that implements this strategy is considerably more compicated that that for the techniques discussed earlier, and I have therefore not included it.

I believe that the strategy I have outlined is the natural one for anyone to adopt who is determined to work with a head-driven active chart parser. However, it is entirely unclear that the advantages that it offers over the simple undirected chart parser are worth its considerable added expense in complexity. Notice that, if one of the other nouns in the sentence just considered also had a verbal interpretation, the search for noun phrases would have been active everywhere. The longer the sentence, and therefore the more pressing the need for high performance, the more active regions there would be in the string and the more nearly the process as a whole would approximate that of the undirected technique. This should not, of course, be taken as an indictment of head-driven parsing, which is interesting for reasons having nothing to do with performance. It does, however, suggest that the temptation to claim that it is also a natural source of efficiency should be resisted.

Appendix A – A PARSER-GENERATOR FOR HEAD-DRIVEN GRAMMAR.

This is a simple head-driven recursive-descent parser. There is a distinction between the top level `parse` predicate and the `syntax` predicate to eliminate inessential arguments to the top level call, and also because the program can, with only minor modifications in `syntax`, be used as a generator. The predicate `head` is assumed to be defined as part of the grammar. It is true of a pair of grammatical labels if the second can be the head (of the head, of the head ...) of the first. Having hypothesized the label of the eventual lexical head of a phrase that will satisfy the current goal, `syntax`calls `range` to find a word in the string with that label. If such a word is found, its position in the string will be given by the `HRange` (head range) difference list and it must, in any case, lie within the range of the string given by `Maxl` and `Maxr`. The `build` predicate constructs phrases with the given putative head so long as their labels stand in the `head` relation to the goal.

```
/**********************************************************************
* parse(String, Result)                                               *
*                                                                     *
* String is a list of words                                           *
* Struct is the structure (nondeterministically) returned if the parse *
* succedes                                                            *
**********************************************************************/
parse(String, Struct):-
   syntax(String/[]/Struct, String/[]).
/**********************************************************************
* syntax((L/R)/Goal, Maxl/Maxr)                                       *
*                                                                     *
*   G is the Goal for the parsen.                                     *
*   L/R is a DL giving the bounds of the phrase satisfying the goal   *
*   Maxl/Maxr gives the string bounds for the current search.         *
*                                                                     *
**********************************************************************/

syntax(Range/Goal, Max) :-
   head(Goal, Head),                       % Find lexical head for Goal
   range(HRange/Head, Max),                % Associate Head with actual
                                           % word and string position.
   build(Range/Goal, HRange/Head, Max).    % Build bottom up based on Head.

/**********************************************************************
* range((L/R)/Head, MaxL/MaxR)                                        *
*                                                                     *
* True of (1) position L/R in the string                              *
*         (3) with grammatical description Head                       *
*         (4) somewhere in the string range MaxL/MaxR (parsing)       *
**********************************************************************/
%
% Whole maximum range explored.
% ============================
range(_, X/X) :-   !, fail.
%
% Next word in maximum range is the required head.
% ================================================

range(L/R/Head, L/_) :-   dict(L/R, Head).
%
% Try again one place to the right.
% =================================
range(Head, [H|T]/MaxR) :-
   range(Head, T/MaxR).

/**********************************************************************
* build((GL/GR)/Goal, (HL/HR)/Head, MaxL/MaxR)                        *
*                                                                     *
* Build phrases bottom up based on the Head located in the string at  *
* HL/HR.  The location of the phrase found will be GL/GR and it must  *
* fall in the range MaxL/MaxR.                                        *
**********************************************************************/
build(X, X, _).                                  % Current head is result.
build(GL/GR/Goal, HL/HR/Head, MaxL/MaxR) :-  % Find rule matching Head
  rr(Lhs, Left, Head, Right), head(Goal, Lhs),
  build_left(Left, LL/HL, MaxL/HL),              % Check left daughters
  build_right(Right, HR/RR, HR/MaxR),            % and right daughters.
  build(GL/GR/Goal, LL/RR/Lhs, MaxL/MaxR).   % Try building further on that.
```

```
build_left([], X/X, _). build_left([H|T], L/R, MaxL/MaxR) :-
  syntax(HL/R/H, MaxL/MaxR),
  build_left(T, L/HL, MaxL/HL).build_right([], X/X, _).
build_right([H|T], L/R, MaxL/MaxR) :-
  syntax(L/HR/H, MaxL/MaxR),
  build_right(T, HR/R, HR/MaxR).
```

Appendix B – A SIMPLE INACTIVE CHART PARSER

This is a chart version of a nondeterminisitc shift-reduce parser. Vertices of the chart are constructed from left to right, one on each recursive call to `parse/3`. A vertex is a list of edges headed by a number which is provided for convenience in printing. An edge takes the form `[label, next-vertex]`. The predicate `build_edge` is given a word and its successor vertex and returns a completed vertex. It succeeds once for each entry that the word has in the dictionary and, for each one, calls `build_edge1`. This can succeed in three ways, all of which are collected into the list of edges contributing to the current vertex by virtue of the `setof` construction. The three possbilities are (1) The word's lexical entry itself labels an edge; (2) A unary rule applies to the entry, and its left-hand side labels an edge, and (3) A binary rule matches the entry and an entry in the next vertex (`member([Label, Next1], Next)`). Each new label is passed to `build-edge1` to be processed in the same manner as the original lexical entry.

```
parse(String, Result) :-
   parse(String, [0], Result).

parse([], V, V).
parse([Word|Rest], [N|Next], Vertex) :-
   setof(Edge, build_edge(Word, [N|Next], Edge), V),
   M is N+1,                               % Next vertex number
   parse(Rest, [M|V], Vertex).             % [M|V] is the vertex

build_edge(Word, Next, Edge) :-
   dict(Word, Entry),                      % Dictionary lookup
   build_edge1(Entry, Next, Edge).

build_edge1(Entry, Next, [Entry, Next]).   % Shift.
build_edge1(Entry, Next, Edge) :-          % Reduce one item
   rule(Lhs, Entry),
   build_edge1(Lhs, Next, Edge).
build_edge1(Entry, [N|Next], Edge) :-      % Reduce two items
```

```
        member([Label, Nextl], Next),
        rule(Lhs, Label, Entry),
        build_edgel(Lhs, Nextl, Edge).
```

Parsing with Principles: Predicting a Phrasal Node Before Its Head Appears[1,2]

Edward Gibson
Department of Philosophy
Carnegie Mellon University
Pittsburgh, PA 15213
eafg@cad.cs.cmu.edu

1 Introduction

Recent work in generative syntactic theory has shifted the conception of a natural language grammar from a homogeneous set of phrase structure (PS) rules to a heterogeneous set of well-formedness constraints on representations (see, for example, Chomsky (1981), Stowell (1981), Chomsky (1986a) and Pollard & Sag (1987)). In these theories it is assumed that the grammar contains principles that are independent of the language being parsed, together with principles that are parameterized to reflect the varying behavior of different languages. However, there is more to a theory of human sentence processing than just a theory of linguistic competence. A theory of performance consists of both linguistic knowledge and a parsing algorithm. This paper will investigate ways of exploiting principle-based syntactic theories directly in a parsing algorithm in order to determine whether or not a principle-based parsing algorithm can be compatible with psycholinguistic evidence.

Principle-based parsing is an interesting research topic not only from a psycholinguistic point of view but also from a practical point of view. When PS rules are used, a separate grammar must be written for each language parsed. Each of these grammars contains a great deal of redundant information. For example, there may be two rules, in different grammars, that are identical except for the order of the constituents on the right hand side, indicating a difference in word order. This redundancy can be avoided by employing a universal phrase structure component (not necessarily in the form of rules) along with parameters and associated values. A principles and parameters approach provides a single compact grammar for all languages that would otherwise be represented by many different (and redundant) PS grammars.

Any model of human parsing must dictate: a) how structures are projected from the lexicon; b) how structures are attached to one another; and c) what constraints affect the resultant structures. This paper will concentrate on the first two components with respect to principle-based parsing algorithms: node projection and structure attachment.

Two basic control structures exist for any parsing algorithm: data-driven control and hypothesis-driven control. Even if a parser is predominantly hypothesis-driven, the predictions that it makes must at some point be compared with the data that are presented to it. Some data-driven component is therefore necessary for any parsing algorithm. Thus, a reasonable hypothesis to test is that the human parsing algorithm is entirely data-driven. This is exactly the approach that is taken by a number of principle-based parsing algorithms (see, for example, Abney (1986), Kashket (1987), Gibson & Clark (1987) and Pritchett (1987)). These parsing algorithms each include a node projection algorithm that projects an input word to a maximal category, but does not cause the projection of any further nodes.

Although this simple strategy is attractive because of its simplicity, it turns out that it cannot account for certain phenomena observed in the processing of Dutch (Frazier (1987): see Section 2.1). A completely data-driven node projection algorithm also has difficulty accounting for the processing ease of adjective-noun constructions in English (see Section 2.2). As a result of this evidence, a purely data-driven node projection

[1] Paper presented at the International Workshop on Parsing Technologies, August 28-31, 1989.

[2] I would like to thank Robin Clark, Rick Kazman, Howard Kurtzman, Eric Nyberg and Brad Pritchett for their comments on earlier drafts of this paper, and I offer the usual disclaimer.

algorithm must be rejected in favor of a node projection algorithm that has a predictive (hypothesis-driven) component Frazier (1987)).

This paper describes a node projection algorithm that is part of the Constrained Parallel Parser (CPP) (Gibson (1987), Gibson & Clark (1987) and Clark & Gibson (1988)). This parser is based on the principles of Government-Binding theory (Chomsky (1981, 1986a)). Section 3.1 gives an overview of the CPP model, while Section 3.2 describes the node projection algorithm. Section 3.3 describes the attachment algorithm, and includes an example parse. These node projection and attachment algorithms demonstrate that a principle-based parsing algorithm can account for the Dutch and English data, while avoiding the existence of redundant phrase structure rules. Thus it is concluded that one should continue to investigate hypothesis-driven principle-based models in the search for an optimal psycholinguistic model.

2 Data-Driven Node Projection: Empirical Predictions and Results

2.1 Evidence from Dutch

Consider the sentence fragment in (1):

(1)
... dat het meisje van Holland ...
... "that the girl from Holland" ...

Dutch is like English in that prepositional phrase modifiers of nouns may follow the noun. Thus the prepositional phrase *van Holland* may be a modifier of the noun phrase *the girl* in example (1). Unlike English, however, Dutch is SOV in subordinate clauses. Hence in (1) the prepositional phrase *van Holland* may also be the argument of a verb to follow. In particular, if the word *glimlachte* ("smiled") follows the fragment in (1), then the prepositional phrase *van Holland* can attach to the noun phrase that it follows, since the verb *glimlachte* has no lexical requirements (see (2a)). If, on the other hand, the word *houdt* ("likes") follows the fragment in (1), then the PP *van Holland* must attach as argument of the verb *houdt*, since the verb requires such a complement (see (2b)).

(2)
a. ... dat [$_S$ [$_{NP}$ het meisje [$_{PP}$ van Holland]] [$_{VP}$ glimlachte]]
... "that the girl from Holland smiled" ...
b. ... dat [$_S$ [$_{NP}$ het meisje] [$_{VP}$ [$_{V'}$ [$_{PP}$ van Holland] [$_V$ houdt]]]]
... "that the girl likes Holland"

Following Abney (1986), Frazier (1987), Clark & Gibson (1988) and numerous others, it is assumed that attached structures are preferred over unattached structures. If we also assume that a phrasal node is not projected until its head is encountered, we predict that people will entertain only one hypothesis for the sentence fragment in (1): the modifier attachment. Thus we predict that it should take longer to parse the continuation *houdt* ("likes") than to parse the continuation *glimlachte* ("smiled"), since the continuation *houdt* forces the prepositional phrase to be reanalyzed as an argument of the verb. However, contrary to this prediction, the verb that allows argument attachment is actually parsed faster than the verb that necessitates modifier attachment in sentence fragments like (1). If the verb had been projected before its head was encountered, then the argument attachment of the PP *van Holland* would be possible at the same time that the modifier attachment is possible.[3] Thus Frazier concludes that in some cases phrasal nodes must be projected before their lexical heads have been encountered.

[3]It is beyond the scope of this paper to offer an explanation as to why the argument attachment is in fact *preferred* to the modifier attachment. This paper seeks only to demonstrate that the argument attachment possibility must at least be *available* for a psychologically real parser. See Abney (1986), Frazier (1987) and Clark & Gibson (1988) for possible explanations for the preference phenomenon.

2.2 Evidence from English

A second piece of evidence against this limited type of node projection is provided by the processing of noun phrases in English that have more than one pre-head constituent.

It is assumed that the primitive operation of attachment is associated with a certain processing cost. Hence the amount of time taken to parse a single input word is directly related to the number of attachments that the parser must execute to incorporate that structure into the existing structure(s). If a phrasal node is not projected until its head is encountered, then parsing the final word of a head-final construction will involve attaching all its pre-head structures at that point. If, in addition, there is more than one pre-head structure and no attachments are possible until the head appears, then a significant proportion of processing time should be spent in processing the head.

The hypothesis that a phrasal node is not projected until its head is encountered can be tested with the English noun phrase, since the head of an English noun phrase appears after a specifier and any adjectival modifiers. For example, consider the English noun phrase *the big red book*. First, the word *the* is read and a determiner phrase is built. Since it is assumed that nodes are not projected until their heads are encountered, no noun phrase is built at this point. The word *big* is now read and causes the projection of an adjective phrase. Attachments are now attempted between the two structures built thus far. Neither of the categories can be argument, specifier or modifier for the other, so no attachment is possible. The next word *red* now causes the projection of an adjective phrase, and once again no attachments are possible. Only when the word *book* is read and projected to a noun phrase can attachments take place. First the adjective phrase representing *red* attaches as a modifier of the noun phrase *book*. Then the AP representing *big* attaches as a modifier of the noun phrase just constructed. Finally the determiner phrase representing *the* attaches as specifier of the noun phrase *big red book*.

Thus if we assume that a phrasal node is not projected until its head is parsed, we predict that a greater number of attachments will take place in parsing the head than in parsing any other word in the noun phrase. Since it is assumed that an attachment is a significant parser operation, it is predicted that people should take more time parsing the head of the noun phrase than they take parsing the other words of the noun phrase. Since there is no psycholinguistic evidence that people take more time to process heads in head-final constructions, I hypothesize that phrasal nodes are being projected before their heads are being encountered.

3 Hypothesizing a Phrasal Node Before Its Head Appears

3.1 The Parsing Model: The Constrained Parallel Parser

This paper assumes the Constrained Parallel Parser (CPP) as its model of human sentence processing (see Gibson (1987), Gibson & Clark (1987) and Clark & Gibson (1988)). The CPP model is based on the principles of Government-Binding Theory (Chomsky (1981, 1986a)); crucially CPP has no separate module containing language-particular rules. Following Marcus (1980), structures parsed under the CPP model are placed on a stack and the most recently built structures are placed in a data structure called the *buffer*. The parser builds structure by attaching nodes in the buffer to nodes on top of the stack. Unlike Marcus' model, the CPP model allows multiple representations for the same input string to exist in a buffer or stack cell. Although multiple representations for the same input string are permitted, constraints on parallelism frequently cause one representation to be preferred over the others. Motivation for the parallel hypothesis comes from garden path effects and perception of ambiguity in addition to relative processing load effects. For information on the particular constraints and their motivations, see Gibson & Clark (1987), Clark & Gibson (1988) and the references cited in these papers.

3.1.1 Lexical Entries for CPP

A lexical entry accessed by CPP consists of, among other things, a *theta-grid*. A theta-grid is an unordered list of *theta structures*. Each theta structure consists of a thematic role and associated subcategorization formation. One theta structure in a theta-grid may be marked as *indirect* to refer to its subject. For example, the word *shout* might have the following theta-grid:[4]

(3)

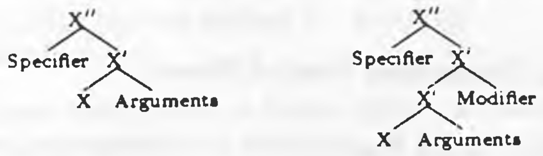
```
((Subcat = PREP, Thematic-Role = GOAL)
 (Subcat = COMP, Thematic-Role = PROPOSITION))
```

When the word *shout* (or an inflected variant of *shout*) is encountered in an input phrase, the thematic role *agent* will be assigned to its subject, as long as this subject is a noun phrase. The direct thematic roles *goal* and *proposition* will be assigned to prepositional and complementizer phrases respectively, as long as each is present. Since the order of theta structures in a theta-grid is not relevant to its use in parsing, the above theta-grid for *shout* will be sufficient to parse both sentences in (4).

(4)
a. The man shouts [$_{PP}$ to the woman] [$_{CP}$ that Ernie sees the rock]
b. The man shouts [$_{CP}$ that Ernie sees the rock] [$_{PP}$ to the woman]

3.1.2 $\overline{\text{X}}$ Theory in CPP

The CPP model assumes $\overline{\text{X}}$ Theory as present in Chomsky (1986b). $\overline{\text{X}}$ Theory has two basic principles: first, each tree structure must have a head; and second, each structure must have a maximal projection. As a result of these principles and other principles, (*e.g.*, the θ-Criterion, the Extended Projection Principle, Case Theory), the positions of arguments, specifiers and modifiers with respect to the head of a given structure are limited. In particular, a specifier may only appear as a sister to the one-bar projection below a maximal projection, and the head, along with its arguments, must appear below the one-bar projection. The orders of the specifier and arguments relative to the head is language dependent. For example, the basic structure of English categories is shown below. Furthermore, binary branching is assumed (Kayne (1983)), so that modifiers are Chomsky-adjoined to the two-bar or one-bar levels, giving one possible structure for a post-head modifier below on the right.



3.1.3 The CPP Parsing Algorithm

The CPP algorithm is essentially very simple. A word is projected via node projection (see Section 3.2) into the buffer. If attachments are possible between the buffer and the top of the stack, then the results of these attachments are placed into the buffer and the stack is popped. Attachments are attempted again until no longer possible. This entire procedure is repeated for each word in the input string. The formal CPP algorithm is given below:

1. (Initializations) Set the stack to nil. Set the buffer to nil.

[4] In a more complete theory, a syntactic category would be determined from the thematic role (Chomsky (1986a)).

2. (Ending Condition) If the end of the input string has been reached and the buffer is empty then return the contents of the stack and stop.

3. If the buffer is empty then project nodes for each lexical entry corresponding to the next word in the input string, and put this list of maximal projections into the buffer.

4. Make all possible attachments between the stack and the buffer, subject to the attachment constraints (see Clark & Gibson (1988)). Put the attached structures in the buffer. If no attachments are possible, then put the contents of the buffer on top of the stack.

5. Go to 2.

3.2 The Projection of Nodes from the Lexicon

Node projection proceeds as follows. First a lexical item is projected to a phrasal node: a *Confirmed* node (*C-node*). Following $\overline{X}$ Theory, each lexical entry for a given word is projected maximally. For example, the word *rock*, which has both a noun and a verb entry would be projected to at least two maximal projections:

(5)
a. $[_{NP}\ [_{N'}\ [_{N}$ rock $]]]$
b. $[_{VP}\ [_{V'}\ [_{V}$ rock $]]]$

Next, the parser hypothesizes nodes whose heads may appear immediately to the right of the given C-node. These predicted structures are called ***hypothesized*** nodes or ***H-nodes***. An H-node is defined to be any node whose head is to the right of all lexical input. In order to determine which H-node structures to hypothesize from a given C-node, it is necessary to consult the argument properties associated with the C-node together with the specifier and modifier properties of the nodal category and the word order properties of the language in question. It is assumed that the ability of one category to act as specifier, modifier or argument of another category is part of unparameterized Universal Grammar. On the other hand, the relative order of two categories is assumed to be parameterized across different languages. For example, a determiner phrase, if it exists in a given language, is universally allowable as a specifier of a noun phrase. Whether the determiner appears before or after its head noun depends on the language-particular values associated with the parameters that determine word order.

Three parameters are proposed to account for variation in word order, one for each of argument, specifier and modifier projections.[5] For each language, each of these parameters is associated with at least one value, where the parameter values come from the following set: {*head*, *satellite*}.[6] The value *head* indicates that a category C causes the projection to the right of those categories for which C may be head. Thus this value indicates head-initial word order. The value *satellite* indicates that a category C causes the projection to the right of those categories for which C may be a satellite category. Hence this value indicates head-final word order. H-node projection from a category C is defined in (6):

(6)
(Argument, Specifier, Modifier) H-Node Projection from category C: If the value associated with the (argument, specifier, modifier) projection parameter is *head*, then cause the projection of (argument, specifier, modifier) satellites, and attach them to the right below the appropriate projection of C. If the value associated with the (argument, specifier, modifier) projection parameter is *satellite*, then cause the projection of (argument, specifier, modifier) heads, and attach them to the right above the appropriate projection of C.

In English the argument projection parameter is set to *head*, so that arguments appear after the head. Hence, if a lexical entry has requirements that must be filled, then structures corresponding to subcategorized

[5]Furthermore, it is assumed that the value of the modifier projection parameter defaults to the value of the argument projection parameter.

[6]I will use the term *satellite* to indicate non-head constituents: arguments, specifiers and modifiers.

categories are hypothesized and attached. For example, the verb *see* subcategorizes for a noun phrase, so an empty noun phrase node is hypothesized and attached as argument of the verb:

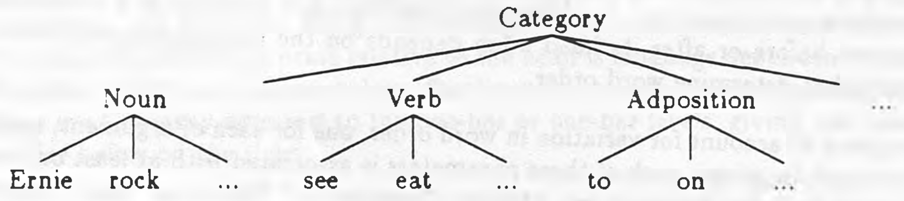
(7)
$[_{VP}\ [_{V'}\ [_{V}\ \text{see}\ ]\ [_{NP}\ e\ ]]]$

The specifier projection parameter, on the other hand, is set to the value *satellite* in English so that specifiers appear before their heads. If the category associated with a C-node is an allowable specifier for other categories, then an H-node projection of each of these categories is built and the C-node specifier is attached to each. For example, since a determiner may specify a noun phrase, an H-node noun phrase is hypothesized when parsing a determiner in English:

(8)
$[_{NP}\ [_{DetP}\ [_{Det'}\ [_{Det}\ \text{the}\ ]]]\ [_{N'}\ [_{N}\ e\ ]]]$

Thus the node projection algorithm provides a new derivation of language-particular word order. In previous principle-based systems, word order is derived from parameterized direction of attachment (see Gibson & Clark (1987), Nyberg (1987), Wehrli (1988)). An attachment takes place from buffer to stack in head-initial constructions and from stack to buffer in head-final constructions. Since attachment is now a uniform operation as defined in (17), this parameterization is no longer necessary. Instead, in head-initial constructions, nodes now project to the nodes that they may immediately dominate. In head-final constructions, nodes now project to those nodes that they may be immediately dominated by.

The projection parameters as defined in (6) account for many facts about word order across languages. However, most, if not all, languages have cases that do not fit this clean picture. For example, while modifiers in English are predominantly post-head, adjectives appear before the head. A single global value for modifier projection predicts that this situation is impossible. Hence we must assume that the values given for the projection parameters are only default values. In order to formalize this idea, I assume the existence of a hierarchy of categories and words as shown below:



It is assumed that the value for each of the projection parameters is the default value for that projection type with respect to a particular language. However, a particular category or word may have a value associated with it for a projection parameter in addition to the default one. If this is the case, then only the most specific value is used. For example, in English, the category adjective is associated with the value *satellite* with respect to modifier projection. Thus English adjectives appear before the head. The adjective *tall* will therefore cause the projection of both a C-node adjective phrase and an H-node noun phrase:

(9)
a. $[_{AP}\ \text{tall}\ ]$
b. $[_{NP}\ [_{N'}\ [_{AP}\ \text{tall}\ ]\ [_{N'}\ [_{N}\ e\ ]]]]$

If recursive application of projection to H-nodes were allowed, then it would be possible, in principle, to project an infinite number of nodes from a single lexical entry. In English, for example, a genitive noun phrase can specify another noun phrase. This noun phrase may also be a genitive noun phrase, and so on. If H-nodes could project to further H-nodes, then it would be necessary to hypothesize an infinite number of genitive NP H-nodes for every genitive NP that is read. As a result of this difficulty, the H-node Projection Constraint is proposed:

(10)
The H-node Projection Constraint: Only a C-node may cause the projection of an H-node.

As a result of the H-node Projection Constraint, H-nodes may not invoke H-node projection. For example, if a specifier causes the projection of its head, the resulting head cannot then cause the projection of those categories that it may specify. As a result, the number of nodes that may be projected from a single lexical item is severely restricted.

3.3 Node Attachment

Given the above node projection algorithm, it is necessary to define an algorithm for attachment of nodes. Since structures are predicted by the node projection algorithm, the attachment algorithm must dictate how subsequent structures match these predictions. Consider the following two examples from English: the first is an example of specifier attachment; the second is an example of argument attachment. In English, specifiers precede the head and arguments follow the head. It is desirable for the attachment algorithm to handle both kinds of attachments without word order particular stipulations.

First, suppose that the word *the* is on the stack as both a determiner phrase and an H-node noun phrase. Furthermore, suppose that the word *woman* is projected into the buffer as both a noun phrase and an H-node clausal phrase:[7]

(11)
Stack: $[_{DetP}$ $[_{Det'}$ $[_{Det}$ the]]]
$[_{NP}$ $[_{DetP}$ $[_{Det'}$ $[_{Det}$ the]]] $[_{N'}$ $[_{N}$ e]]]
Buffer: $[_{NP}$ $[_{N'}$ $[_{N}$ woman]]]
$[_{XP_{clause}}$ $[_{NP}$ $[_{N'}$ $[_{N}$ woman]]] $[_{X'_{clause}}$ $[_{X_{clause}}$ e]]]

The attachment algorithm should allow two attachments at this point: the H-node NP on the stack uniting with each NP C-node in the buffer. It might also seem reasonable to allow the bare determiner phrase to attach directly as specifier of each noun phrase. However, this kind of attachment is undesirable for two reasons. First of all, it makes the attachment operation a disjunctive operation: an attachment would involve *either* matching an H-node *or* meeting the satellite requirements of a category. Second of all, it makes H-node projection unnecessary in most situations and therefore somewhat stipulative. That is, allowing a disjunctive attachment operation would permit many derivations that never use an H-node, so that the need for H-nodes would be restricted to head-final constructions with pre-head satellites (see Section 2). It is therefore desirable for all attachments to involve matching an H-node.

Two structures should be returned after attachments in (11): a C-node noun phrase and an H-node clausal phrase:

(12)
a. $[_{NP}$ $[_{DetP}$ the] $[_{N'}$ $[_{N}$ woman]]]
b. $[_{XP_{clause}}$ $[_{NP}$ $[_{DetP}$ the] $[_{N'}$ $[_{N}$ woman]]] $[_{X'_{clause}}$ $[_{X_{clause}}$ e]]]

Now consider an English argument attachment. Suppose that a prepositional phrase representing the word *beside* is on the stack and the noun *Frank* is represented in the buffer as a noun phrase and a clausal phrase:

(13)
Stack: $[_{PP}$ $[_{P'}$ $[_{P}$ beside] $[_{NP}$ e]]]
Buffer: $[_{NP}$ $[_{N'}$ $[_{N}$ Frank]]]
$[_{XP_{clause}}$ $[_{NP}$ $[_{N'}$ $[_{N}$ Frank]]] $[_{X'_{clause}}$ $[_{X_{clause}}$ e]]]

---

[7] A noun phrase is projected to an H-node clausal (or predicate) phrase since nouns may be the subjects of predicates.

Since the preposition *beside* subcategorizes for a noun phrase, there is an H-node NP attached as its object. The attachment algorithm should allow a single attachment at this point: the noun phrase representing *Frank* uniting with the H-node NP object of *beside*:

(14)
$[_{PP}\ [_{P'}\ [_{P}$ beside $]\ [_{NP}$ Frank $]]]$

As should be clear from the two examples, the process of attachment involves comparing a previously predicted category with a current category. If the two categories are *compatible*, then attachment may be viable.

3.3.1 Node Compatibility

Compatibility is defined in terms of *unification*, which is defined [illegible] terms of *subsumption*.[8] A structure X is said to *subsume* a structure Y if X is more general than Y. That [illegible] contains less specific information than Y. So, for example, a structure that is specified as *clausal* (e.g. t[illegible]ead of a predicate), but is not specified for a particular category subsumes a structure having the categor[illegible]*erb*, since verbs are predicative and thus clausal categories. Hence structure (15a) subsumes structure (15b):

(15)
a. $[_{XP_{clause}}\ [_{X'_{clause}}\ [_{X_{clause}}\ e\ ]]]$
b. $[_{VP}\ [_{V'}\ [_{V}$ walk $]]]$

The *unification* operation is the least upper bound operator in the subsumption ordering on information in a structure. Since structure (15a) subsumes structure (15b), the result of unifying structure (15a) with structure (15b) is structure (15b). Two structures are *compatible* if the unification of the two structures is non-nil. The information on a structure that is relevant to attachment consists of the node's bar level (*e.g.*, zero level, intermediate or maximal), and the node's lexical features, (*e.g.* category, case, *etc*).

3.3.2 Attachment

Roughly speaking, the attachment operation should locate an H-node in a structure on the stack along with a compatible node in a structure in the buffer. If both of these structures have parent tree structures, then these parent tree structures must also be compatible. In order to keep the process of attachment simple, it is proposed that each attachment have at most one compatibility check. This constraint is given in (16):[9]

(16)
Attachment Constraint: At most one nontrivial lexical feature unification is permitted per attachment.

A nontrivial unification is one that involves two nontrivial structures; a trivial unification is one that involves at least one trivial structure. For example, if the parent node of the buffer site is as of yet undefined, then the parent node of the stack site trivially unifies with this parent node. Only when both parents are defined is there a nontrivial unification.

Consider the effect of the following three requirements: first, the lexical features of the stack and buffer attachment sites must be compatible; second, the tree structures above the buffer and stack attachment sites must be compatible; and third, at most one lexical feature unification is permissible per derivation, (16). Since any attachment must involve at least one nontrivial lexical feature unification, that of the stack and buffer sites, any additional nontrivial unifications will violate the attachment constraint in (16). If both

[8] See Sheiber (1986) for background on the possible uses of unification in particular grammar formalisms.

[9] In fact, this constraint follows from the two assumptions: first, a compatibility check takes a certain amount of processing time; and second, attachments that take less time are preferred over those that take more time. See Gibson (forthcoming) for further discussion.

the buffer and stack attachment sites have parent tree structures, then the lexical features of these parents will need to be unified. Since the child structures will also need to be unified, (16) will be violated. Thus it follows that, in an attachment, either the buffer site or the stack site has no parent tree structure.[10]

Since the order of the words in the input must be maintained in a final parse, only those nodes in a buffer structure that dominate all lexical items in that structure are permissible as attachment sites. For example, suppose that the buffer contained a representation for the noun phrase *women in college*. Furthermore, suppose that there is an H-node NP on the stack representing the word *the*. Although it would be suitable for the buffer structure representing the entire noun phrase *women in college* to match the stack H-node, it would not be suitable for the C-node NP representing *college* to match this H-node. This attachment would result in a structure that moved the lexical input *women in* to the left of the lexical input dominated by the matched H-node, producing a parse for the input *women in the college*. Since the word order of the input string must be maintained, sites for buffer attachment must dominate all lexical items in the buffer structure.

Once suitable maximal projections in each of the buffer and stack structures have been identified for matching, it is still necessary to check that their internal structures are compatible. For example, suppose that an identified buffer site is a C-node whose head allows exactly one specifier and a specifier is already attached. If the stack H-node site also contains a specifier, then the attachment should be blocked. On the other hand, if the stack H-node site does not contain a specifier, and other requirements are satisfied, then the attachment should be allowed.

Testing for internal structure compatibility is quite simple if all tree structures are assumed to be binary branching ones. The only possible attachment sites inside the stack H-node are those nodes that dominate no other nodes. As long as there is some buffer node that both dominates all the buffer input and matches the H-node attachment site for bar level, then the attachment is possible.

Attachment is formally defined in (17):

(17)
A structure W in the buffer can attach to a structure X on the stack iff all of (a), (b), (c), (d) and (e) are true:

a. Structure W contains a maximal projection node, Y, such that Y dominates all lexical material in W;
b. Structure X contains a maximal projection H-node structure, Z;
c. The tree structure above Y is compatible with the tree structure above Z, subject to the attachment constraint in (16);
d. The lexical features of structure Y are compatible with the lexical features of structure Z;
e. Structure Y is *bar-level compatible* with structure Z.

Bar-level compatibility is defined in (18):

(18)
A structure U in the buffer is ***bar-level compatible*** with a structure V on the stack iff all of (a), (b) and (c) are true:

a. Structure U contains a node, S, such that S dominates all lexical material in U;
b. Structure V contains an H-node structure, T, that dominates no lexical material;
c. The bar level of S is compatible with the bar level of T.

If attachment is viable, then W contains a structure Y that is bar-level compatible with a structure Z that is part of X. Since Y and Z are bar-level compatible, there are structures S and T inside Y and Z

[10] It might seem that some possible attachments are being thrown away at this point. That is, in principle, there might be a structure that can only be formed by attaching a buffer site to a stack site where both sites have parent tree structures. This attachment would be blocked by (16). However, it turns out that any attachment that could have been formed by an attachment involving more than one lexical feature unification can always be arrived at by a different attachment involving a single lexical feature unification. For the proof, see Gibson (forthcoming).

respectively, that satisfy the conditions of bar-level compatibility, (18).

When the conditions for attachment are satisfied, structures W and X are united in the following way. First, W and X are copied to nodes W' and X' respectively. Inside X' there is a node, Z', that is a copy of Z. The lexical features of Z' are set to the unification of the lexical features of structures Y and Z. Next, structure T' in Z' (corresponding to structure T in Z) is replaced by S', the copy of structure S inside W. The bar level of T' is set to the unification of the bar levels of structures S and T.

Finally, the tree structures above Y and Z are unified and this tree structure is attached above Z'. That is, if Z has some parent tree structure and Y does not, then the copy of this structure inside X' is attached above Z'. Similarly, if Y has some parent tree structure and Z does not, then the copy of this structure inside W is attached above Z'. If neither node has any parent tree structure (*i.e.*, $W = Y$, $X = Z$), then the unification is trivial and no attachment is made. Since Y and Z cannot both have parent tree structures (see (16) and the discussion following it), unifying the parent tree structures is a very simple process.

3.3.3. Example Attachments

As an illustration of how attachments take place, consider once again the noun phrase *the big red book*. First the determiner *the* is read and is projected to a C-node determiner phrase. Since a determiner is allowable as the specifier of a noun phrase and specifiers occur before the head in English, an H-node NP is also built. These two structures are depicted in (19):

(19)
a. $[_{DetP}$ the $]$
b. $[_{NP}\ [_{DetP}$ the $]\ [_{N'_1}\ [_{N}\ e\ ]]]$

Since there is nothing on the stack, these structures are shifted to the top of the stack. The word *big* projects to both a C-node AP and an H-node NP since an adjective is allowable as a pre-head modifier in English. These two structures are placed in the buffer (depicted in (20)).

(20)
a. $[_{AP}$ big $]$
b. $[_{NP}\ [_{N'_2}\ [_{AP}$ big $]\ [_{N'}\ [_{N}\ e\ ]]]]$

An attachment between nodes (19b) and (20b) is now attempted. Note that: a) node (20b) is a maximal projection dominating all lexical material in its buffer structure; b) node (19b) is a maximal projection H-node on the stack; c) the tree structures above these two nodes are compatible (both are undefined); and d) the categories of the two nodes are compatible. It remains to check for bar-level compatibility of the two structures. Since: a) the N'_2 in structure (20b) dominates all the buffer input; b) the H-node N'_1 in structure (19b) dominates no C-nodes; and c) N'_1 and N'_2 are compatible in bar level, the structures in (19b) and (20b) can be attached. The two structures are therefore attached by uniting N'_1 and N'_2. The resultant structure is given in (21):

(21)
$[_{NP}\ [_{DetP}$ the $]\ [_{N'}\ [_{AP}$ big $]\ [_{N'_3}\ [_{N}\ e\ ]]]]$

Structure (21), the only possible attachment between the buffer and the stack, is placed back in the buffer, and the stack is popped. Since there is now nothing left on the stack, no further attachments are possible at this time. Structure (21) is thus shifted to the stack. The word *red* now enters the buffer as a C-node adjective phrase and an H-node noun phrase:

(22)
a. $[_{AP}$ red $]$
b. $[_{NP}\ [_{N'_4}\ [_{AP}$ red $]\ [_{N'}\ [_{N}\ e\ ]]]]$

An attachment between nodes (21) and (22b) is now attempted. Requirements (17a)-(17d) are satisfied and the requirement for bar-level compatibility is satisfied by the node labeled N'_3 in (21) together with N'_4 in (22b). Hence the structures are united, giving (23):

(23)
$[_{NP}\ [_{DetP}\ \text{the}\ ]\ [_{N'}\ [_{AP}\ \text{big}\ ]\ [_{N'}\ [_{AP}\ \text{red}\ ]\ [_{N'_5}\ [_{N}\ e\ ]]]]]$

Since (23) is the only possible attachment between the buffer and the stack, it is placed in the buffer and the stack is popped. Since the stack is now empty, structure (23) shifts to the stack. The noun *book* now enters the buffer as both a C-node noun phrase and an H-node clausal phrase:

(24)
a. $[_{NP}\ [_{N'}\ [_{N}\ \text{book}\ ]]]$
b. $[_{XP_{clause}}\ [_{NP}\ [_{N'}\ [_{N}\ \text{book}\ ]]]\ [_{X'_{clause}}\ [_{X_{clause}}\ e\ ]]]$

Two attachments are possible at this point. The NP structure in (23) unites with each NP C-node on the stack, resulting in the structures in (25):

(25)
a. $[_{NP}\ [_{DetP}\ \text{the}\ ]\ [_{N'}\ [_{AP}\ \text{big}\ ]\ [_{N'}\ [_{AP}\ \text{red}\ ]\ [_{N'}\ [_{N'}\ [_{N}\ \text{book}\ ]]\ [_{PP}\ e]\ [_{CP}\ e\ ]]]]]$
b. $[_{XP_{clause}}\ [_{NP}\ \text{the big red book}\ ]\ [_{X'_{clause}}\ [_{X_{clause}}\ e\ ]]]$

Note that only one attachment per structure takes place in the final parse step. Crucially, no more attachments per structure take place when parsing the head of the noun phrase than when parsing the pre-head constituents in the noun phrase.[11] Thus, in contrast with the situation when nodes are only projected when their heads are encountered, the node projection and attachment algorithms described here predict that there should not be any slowdown when parsing the head of a head-final construction.

The Dutch data described in Section 2.1 are handled in a similar manner.

4 Conclusions

This paper has described a) a principle-based algorithm for the projection of phrasal nodes before their heads are parsed, and b) an algorithm for attaching the predicted nodes. It is worthwhile to compare the new projection algorithm with algorithms that do not project H-nodes. The projection algorithm provided here involves more work and hence, on the surface, may seem somewhat stipulative compared to one that does not project H-nodes. However, it turns out that although projecting to H-nodes is more complicated than not doing so, attachment when H-nodes are not present is more complicated than attachment when they are present. That is, if a projection algorithm causes the projection of H-nodes, it will have a more complicated attachment algorithm. For example, if H-nodes are projected when parsing the noun phrase *the woman*, the determiner *the* is immediately projected to an H-node noun phrase, which leads to a simple attachment. If H-nodes are not projected, then projection is easier, but attachment is that much more complicated. When attaching, it will be necessary to check if a determiner is an allowable specifier of a noun phrase: the same operation that is performed when projecting to H-nodes. Thus although the complexity of particular components changes , the complexity of the entire parsing algorithm does not change, whether or not H-nodes are projected. Since the proposed projection and attachment algorithms make better empirical predictions than ones that do not predict structure, the new algorithms are preferred.

[11]Note that it is the number of attachments per structure that is crucial here, and not the number of total attachments, since attachments made upon two independent structures may be performed in parallel, whereas attachments made on the same structure must be performed serially. For example, since structures (24a) and (24b) are independent, attachments may be made to each of these in parallel. But if an attachment, B relies on the result of another attachment A, then attachment A must be performed first.

5 References

Abney (1986), "Licensing and Parsing", *Proceedings of the Seventeenth North East Linguistic Society Conference*, MIT, Cambridge, MA.

Chomsky, N. (1981), *Lectures on Government and Binding*, Foris, Dordrecht, The Netherlands.

Chomsky, N. (1986a), *Knowledge of Language: Its Nature, Origin and Use*, Praeger Publishers, New York, NY.

Chomsky, N. (1986b), *Barriers*, Linguistic Inquiry Monograph 13, MIT Press, Cambridge, MA.

Clark, R. & Gibson, E. (1988), "A Parallel Model for Adult Sentence Processing", *Proceedings of the Tenth Cognitive Science Conference*, McGill University, Montreal, Quebec.

Frazier, L. (1987) "Syntactic Processing Evidence from Dutch", *Natural Language and Linguistic Theory* 5, pp. 519-559.

Gibson, E. (1987), *Garden-Path Effects in a Parser with Parallel Architecture*, Eastern States Conference on Linguistics, Columbus Ohio.

Gibson, E. (forthcoming), *Parsing with Principles: A Computational Theory of Human Sentence Processing*, Ms., Carnegie Mellon University, Pittsburgh, PA.

Gibson, E. & Clark, R. (1987), "Positing Gaps in a Parallel Parser", *Proceedings of the Eighteenth North East Linguistic Society Conference*, University of Toronto, Toronto, Ontario.

Kashket, M. (1987), *Government-Binding Parser for Warlpiri, a Free Word Order Language*, MIT Master's Thesis, Cambridge, MA.

Kayne, R. (1983) *Connectedness and Binary Branching*, Foris, Dordrecht, The Netherlands.

Marcus, M. (1980), *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, MA.

Nyberg, E. (1987), "Parsing and and the Acquisition of Word Order", *Proceedings of the Fourth Eastern States Conference on Linguistics*, The Ohio State University, Columbus, OH.

Pollard, C. & Sag, I. (1987) *An Information-based Syntax and Semantics*, CSLI Lecture Notes Number 13, Menlo Park, CA.

Pritchett, B. (1987), *Garden Path Phenomena and the Grammatical Basis of Language Processing*, Harvard University Ph.D. dissertation, Cambridge, MA.

Sheiber, S. (1986) *An Introduction to Unification-based Approaches to Grammar*, CSLI Lecture Notes Number 4, Menlo Park, CA.

Stowell, T. (1981), *Origins of Phrase Structure*, MIT Ph.D. dissertation.

Wehrli, E. (1988), "Parsing with a GB Grammar", in U. Reyle and C. Rohrer (eds.), *Natural Language Parsing and Linguistic Theories*, 177-201, Reidel, Dordrecht, the Netherlands.

The Computational Implementation of Principle-Based Parsers[1]

Sandiway Fong
Robert C. Berwick
Artificial Intelligence Laboratory,
Massachusetts Institute of Technology

Abstract

This paper addresses the issue of how to organize linguistic principles for efficient processing. Based on the general characterization of principles in terms of purely computational properties, the effects of principle-ordering on parser performance are investigated. A novel parser that exploits the possible variation in principle-ordering to dynamically re-order principles is described. Heuristics for minimizing the amount of unnecessary work performed during the parsing process are also discussed.

1 Introduction

Recently, there has been some interest in the implementation of grammatical theories based on the principles and parameters approach (Correa [3], Dorr [4], Johnson [5], Kolb & Thiersch [6], and Stabler [10]). In this framework, a fixed set of universal principles parameterized according to particular languages interact deductively to account for diverse linguistic phenomena. Much of the work to date has focused on the not inconsiderable task of formalizing such theories. The primary goal of this paper is to explore the computationally-relevant properties of this framework. In particular, we address the hitherto largely unexplored issue of how to organize linguistic principles for efficient processing. More specifically, this paper examines if, and how, a parser can re-order principles to avoid doing unnecessary work. Many important questions exist: for example, (1) What effect, if any, does principle-ordering have on the amount of work needed to parse a given sentence? (2) If the effect of principle-ordering is significant, then are some orderings much better than others? (3) If so, is it possible to predict (and explain) which ones these are?

By characterizing principles in terms of the purely computational notions of "filters" and "generators", we show how how principle-ordering can be utilized to minimize the amount of work performed in the course of parsing. Basically, some principles, like Move-α (a principle relating 'gaps' and 'fillers') and Free Indexing (a principle relating referential items) are "generators" in the sense that they build more hypothesized output structures than their inputs. Other principles, like the θ-Criterion which places restrictions on the assignment of thematic relations, the Case Filter which requires certain noun phrases to be

[1]The work of the first author is supported by an IBM Graduate Fellowship. R.C. Berwick is supported by NSF Grant DCR-85552543 under a Presidential Young Investigator's Award.

marked with abstract Case, and Binding Theory constraints, act as filters and weed-out ill-formed structures.

A novel, logic-based parser, the Principle-Ordering Parser (PO-PARSER), was built to investigate and demonstrate the effects of principle-ordering. The PO-PARSER was deliberately constructed in a highly-modular fashion to allow for maximum flexibility in exploring alternative orderings of principles. For instance, each principle is represented separately as an atomic parser operation. A structure is deemed to be well-formed only if it passes all parser operations. The scheduling of parser operations is controlled by a dynamic ordering mechanism that attempts to eliminate unnecessary work by eliminating ill-formed structures as quickly as possible. (For comparison purposes, the PO-PARSER also allows the user to turn off the dynamic ordering mechanism and to parse with a user-specified (fixed) sequence of operations.)

Although we are primarily interested in exploiting the (abstract) computational properties of principles to build more efficient parsers, the PO-PARSER is also designed to be capable of handling a reasonably wide variety of linguistic phenomena. The system faithfully implements most of the principles contained in Lasnik & Uriagereka's [7] textbook. That is, the parser makes the same grammaticality judgements and reports the same violations for ill-formed structures as the reference text. Some additional theory is also drawn from Chomsky [1] and [2]. Parser operations implement principles from Theta Theory, Case Theory, Binding Theory, Subjacency, the Empty Category Principle, movement at the level of Logical Form as well in overt syntax, and some Control Theory. This enables it to handle diverse phenomena including parasitic gaps constructions, strong crossover violations, passive, raising, and super-raising examples.

2 The Principle Ordering Problem

This section addresses the issue of how to organize linguistic principles in the PO-PARSER framework for efficient processing. More precisely, we discuss the problem of how to order the application of principles to minimize the amount of 'work' that the parser has to perform. We will explain why certain orderings may be better in this sense than others. We will also describe heuristics that the PO-PARSER employs in order to optimize the the ordering of its operations.

But first, is there a significant performance difference between various orderings? Alternatively, how important an issue is the principle ordering problem in parsing? An informal experiment was conducted using the PO-PARSER described in the previous section to provide some indication on the magnitude of the problem. Although we were unable to examine all the possible orderings, it turns out that order-of-magnitude variations in parsing times could be achieved merely by picking a few sample orderings.[2]

[2]The PO-PARSER has about twelve to sixteen parser operations. Given a set of one dozen operations, there are about 500 million different ways to order these operations. Fortunately, only about half a million of these are actually valid, due to logical dependencies between the various operations. However, this is still far too many to test exhaustively. Instead, only a few well-chosen orderings were tested on a number of sentences from the reference. The procedure

2.1 Explaining the Variation in Principle Ordering

The variation in parsing times for various principle orderings that we observed can be explained by assuming that overgeneration is the main problem, or bottleneck, for parsers such as the PO-PARSER. That is, in the course of parsing a single sentence, a parser will hypothesize many different structures. Most of these structures, the ill-formed ones in particular, will be accounted for by one or more linguistic filters. A sentence will be deemed acceptable if there exists one or more structures that satisfy every applicable filter. Note that even when parsing grammatical sentences, overgeneration will produce ill-formed structures that need to be ruled out. Given that our goal is to minimize the amount of work performed during the parsing process, we would expect a parse using an ordering that requires the parser to perform extra work compared with another ordering to be slower.

Overgeneration implies that we should order the linguistic filters to eliminate ill-formed structures as quickly as possible. For these structures, applying any parser operation other than one that rules it out may be considered as doing extra, or unnecessary, work (modulo any logical dependencies between principles).[3] However, in the case of a well-formed structure, principle ordering cannot improve parser performance. By definition, a well-formed structure is one that passes all relevant parser operations: Unlike the case of an ill-formed structure, applying one operation cannot possibly preclude having to apply another.

2.2 Optimal Orderings

Since some orderings perform better than others, a natural question to ask is: Does there exist a 'globally' optimal ordering? The existence of such an ordering would have important implications for the design of the control structure of any principle-based parser. The PO-PARSER has a novel 'dynamic' control structure in the sense that it tries to determine an ordering-efficient strategy for every structure generated. If such a globally optimal ordering could be found, then we can do away with the run-time overhead and parser machinery associated with calculating individual orderings. That is, we can build an ordering-efficient parser simply by 'hardwiring' the optimal ordering into its control structure. Unfortunately, no such ordering can exist.

The impossibility of the globally optimal ordering follows directly from the "eliminate unnecessary work" ethic. Computationally speaking, an optimal ordering is one that rules out ill-formed structures at the earliest possible opportunity. A *globally* optimal ordering would be one that always ruled out every

involved choosing a default sequence of operations and 'scrambling' the sequence by moving operations as far as possible from their original positions (modulo any logical dependencies between operations).

[3]In the PO-PARSER for example, the Case Filter operation which requires that all overt noun phrases have abstract Case assigned, is dependent on both the inherent and structural Case assignment operations. That is, in any valid ordering the filter must be preceded by both operations.

possible ill-formed structure without doing any unnecessary work. Consider the following three structures (taken from Lasnik's book):

(1) a. *John$_1$ is crucial $[_{CP}[_{IP}$ t_1 to see this]]
b. *$[_{NP}$John$_1$'s mother $][_{VP}$ likes himself$_1$]
c. *John$_1$ seems that he$_1$ likes t_1

Example (1) violates the Empty Category Principle (ECP). Hence the optimal ordering must invoke the ECP operation before any other operation that it is not dependent on. On the other hand, example (1b) violates a Binding Theory principle, 'Condition A'. Hence, the optimal ordering must also invoke Condition A as early as possible. In particular, given that the two operations are independent, the optimal ordering must order Condition A before the ECP and vice-versa. Similarly, example (1c) demands that the 'Case Condition on Traces' operation must precede the other two operations. Hence a globally optimal ordering is impossible.

2.3 Heuristics for Principle Ordering

The principle-ordering problem can be viewed as a limited instance of the well-known conjunct ordering problem (Smith & Genesereth [9]). Given a set of conjuncts, we are interested in finding all solutions that satisfy all the conjuncts simultaneously. The parsing problem is then to find well-formed structures (i.e. solutions) that satisfy all the parser operations (i.e. conjuncts) simultaneously. Moreover, we are particularly interested in minimizing the cost of finding these structures by re-ordering the set of parser operations.

This section outlines some of the heuristics used by the PO-PARSER to determine the minimum cost ordering for a given structure. The PO-PARSER contains a dynamic ordering mechanism that attempts to compute a minimum cost ordering for every phrase -ucture generated during the parsing process.[4] The mechanism can be subd ied into two distinct phases. First, we will describe how the dynamic ordering mechanism decides which principle is the most likely candidate for eliminating a given structure. Then, we will explain how it makes use of this information to re-order parser operation sequences to minimize the total work performed by the parser.

2.3.1 Predicting Failing Filters

Given any structure, the dynamic ordering mechanism attempts to satisfy the "eliminate unnececessary work" ethic by predicting a "failing" filter for that

[4]In their paper, Smith & Genesereth drew a distinction between "static" and "dynamic" ordering strategies. In static strategies, the conjuncts are first ordered, and then solved in the order presented. By contrast, in dynamic strategies the chosen ordering may be revised between solving individual conjuncts. Currently, the PO-PARSER employs a dynamic strategy. The ordering mechanism computes an ordering based on certain features of each structure to be processed. The ordering may be revised after certain operations (e.g. movement) that modify the structure in question.

structure. More precisely, it will try to predict the principle that a given structure violates on the basis of the simple structure cues. Since the ordering mechanism cannot know whether a structure is well-formed or not, it assumes that all structures are ill-formed and attempts to predict a failing filter for every structure. In order to minimize the amount of work involved, the types of cues that the dynamic ordering mechanism can test for are deliberately limited. Only inexpensive tests such as whether a category contains certain features (e.g. ±anaphoric, ±infinitival, or whether it is a trace or a non-argument) may be used. Any cues that may require significant computation, such as searching for an antecedent, are considered to be too expensive. Each structure cue is then associated with a list of possible failing filters. (Some examples of the mapping between cues and filters are shown below.) The system then chooses one of the possible failing filters based on this mapping.[5]

(2)

Structure cue	*Possible failing filters*
trace	Empty Category Principle, and Case Condition on traces
intransitive	Case Filter
passive	Theta Criterion Case Filter
non-argument	Theta Criterion
+anaphoric	Binding Theory Principle A
+pronominal	Binding Theory Principle B

The correspondence between each cue and the set of candidate filters may be systematically derived from the definitions of the relevant principles. For example, Principle A of the Binding Theory deals with the conditions under which antecedents for anaphoric items, such as "each other" and "himself", must appear. Hence, Principle A can only be a candidate failing filter for structures that contain an item with the +anaphoric feature. Other correspondences may be somewhat less direct: for example, the Case Filter merely states that all overt noun phrase must have abstract Case. Now, in the PO-PARSER the conditions under which a noun phrase may receive abstract Case are defined by two separate operations, namely, Inherent Case Assignment and Structural Case Assignment. It turns out that an instance where Structural Case Assignment will not assign Case is when a verb that normally assigns Case has passive morphology. Hence, the presence of a passive verb in a given structure may cause an overt noun phrase to fail to receive Case during Structural Case Assignment — which, in turn may cause the Case Filter to fail.[6]

[5]Obviously, there are many ways to implement such a selection procedure. Currently, the PO-PARSER uses a voting scheme based on the frequency of cues. The (unproven) underlying assumption is that the probability of a filter being a failing filter increases with the number of occurrences of its associated cues in a given structure. For example, the more traces there are in a structure, the more likely it is that one of them will violate some filter applicable to traces, such as the Empty Category Principle (ECP).

[6]It is possible to automate the process of finding structure cues simply by inspecting the closure of the definitions of each filter and all dependent operations. One method of deriving

The failing filter mechanism can been seen as an approximation to the Cheapest-first heuristic in conjunct ordering problems. It turns out that if the cheapest conjunct at any given point will reduce the search space rather than expand it, then it can be shown that the optimal ordering must contain that conjunct at that point. Obviously, a failing filter is a "cheapest" operation in the sense that it immediately eliminates one structure from the set of possible structures under consideration.

Although the dynamic ordering mechanism performs well in many of the test cases drawn from the reference text, it is by no means foolproof. There are also many cases where the prediction mechanism triggers an unprofitable re-ordering of the default order of operations. (We will present one example of this in the next section.) A more sophisticated prediction scheme, perhaps one based on more complex cues, could increase the accuracy of the ordering mechanism. However, we will argue that it is not cost-effective to do so. The basic reason is that, in general, there is no *simple* way to determine whether a given structure will violate a certain principle.[7] That is, as far as one can tell, it is difficult to produce a cheap (relative to the cost of the actual operation itself), but effective approximation to a filter operation. For example, in Binding Theory, it is difficult to determine if an anaphor and its antecedent satisfies the complex locality restrictions imposed by Principle A without actually doing some searching for a binder. Simplifying the locality restrictions is one way of reducing the cost of approximation, but the very absence of search is the main reason why the overhead of the present ordering mechanism is relatively small.[8] Hence, having more sophisticated cues may provide better approximations, but the tradeoff is that the prediction methods may be almost as expensive as performing the real operations themselves.

2.3.2 Logical Dependencies and Re-ordering

Given a candidate failing filter, the dynamic ordering mechanism has to schedule the sequence of parser operations so that the failing filter is performed as early

cues is to collect the negation of all conditions involving category features. For example, if an operation contains the condition "`not (Item has_feature intransitive)`", then we can take the presence of an intransitive item as a possible reason for failure of that operation. However, this approach has the potential problem of generating too many cues. Although, it may be relatively inexpensive to test each individual cue, a large number of cues will significantly increase the overhead of the ordering mechanism. Furthermore, it turns out that not all cues are equally useful in predicting failure filters. One solution may be to use "weights" to rank the predictive utility of each cue with respect to each filter. Then an adaptive algorithm could be used to "learn" the weighting values, in a manner reminiscent of Samuels [8]. The failure filter prediction process could then automatically eliminate testing for relatively unimportant cues. This approach is currently being investigated.

[7] If such a scheme can be found, then it can effectively replace the definition of the principle itself.

[8] We ignore the additional cost of re-ordering the sequence of operations once a failing filter has been predicted. The actual re-ordering can be made relatively inexpensive using various tricks. For example, it is possible to "cache" or compute (off-line) common cases of re-ordering a default sequence with respect to various failing filters, thus reducing the cost of re-ordering to that of a simple look-up.

as possible. Simply moving the failing filter to the front of the operations queue is not a workable approach for two reasons.

Firstly, simply fronting the failing filter may violate logical dependencies between various parser operations. For example, suppose the Case Filter was chosen to be the failing filter. To create the conditions under which the Case Filter can apply, both Case assignment operations, namely, Inherent Case Assignment and Structural Case Assignment, must be applied first. Hence, fronting the Case Filter will also be accompanied by the subsequent fronting of both assignment operations — unless, of course, they have already been applied to the structure in question.

Secondly, the failing filter approach does not take into account the behaviour of "generator" operations. A generator may be defined as any parser operation that always produces one output, and possibly more than one output, for each input. For example, the operations corresponding to $\bar{X}$ rules, Move-α, Free Indexing and LF Movement are the generators in the PO-PARSER. (Similarly, the operations that we have previously referred to as "filters" may be characterized as parser operations that, when given N structures as input, pass N and possibly fewer than N structures.) Due to logical dependencies, it may be necessary in some situations to invoke a generator operation before a failure filter can be applied. For example, the filter Principle A of the Binding Theory is logically dependent on the generator Free Indexing to generate the possible antecedents for the anaphors in a structure. Consider the possible binders for the anaphor "*himself*" in "*John thought that Bill saw himself*" as shown below:

(3) a. *John$_i$ thought that Bill$_j$ saw himself$_i$
b. John$_i$ thought that Bill$_j$ saw himself$_j$
c.*John$_i$ thought that Bill$_j$ saw himself$_k$

Only in example (3b), is the antecedent close enough to satisfy the locality restrictions imposed by Principle A. Note that Principle A had to be applied a total of three times in the above example in order to show that there is only one possible antecedent for "*himself*". This situation arises because of the general tendency of generators to overgenerate. But this characteristic behaviour of generators can greatly magnify the extra work that the parser does when the dynamic ordering mechanism picks the wrong failing filter. Consider the ill-formed structure "**John seems that he likes t*" (a violation of the principle that traces of noun phrase cannot receive Case.) If however, Principle B of the Binding Theory is predicted to be the failure filter (on the basis of the structure cue "*he*"), then Principle B will be applied repeatedly to the indexings generated by the Free Indexing operation. On the other hand, if the Case Condition on Traces operation was correctly predicted to be the failing filter, then Free Indexing need not be applied at all. The dynamic ordering mechanism of the PO-PARSER is designed to be sensitive to the potential problems caused by selecting a candidate failing filter that is logically dependent on many generators.[9]

[9]Obviously, there are many different ways to accomplish this. One method is to compute

2.4 Linguistic Filters and Determinism

In this section we describe how the characterization of parser operations in terms of filters and generators may be exploited further to improve the performance of the PO-PARSER for some operations. More precisely, we make use of certain computational properties of linguistic filters to improve the backtracking behaviour of the PO-PARSER. The behaviour of this optimization will turn out to complement that of the ordering selection procedure quite nicely. That is, the optimization is most effective in exactly those cases where the selection procedure is least effective.

We hypothesize that linguistic filters, such as the Case Filter, Binding Conditions, ECP, and so on, may be characterized as follows:

> (4) **Hypothesis**: Linguistic filters are side-effect free conditions on configurations

In terms of parser operations, this means that filters should never cause structure to be built or attempt to fill in feature slots.[10] Moreover, computationally speaking, the parser operations corresponding to linguistic filters should be deterministic. That is, any given structure should either fail a filter or just pass. A filter operation should never need to succeed more than once, simply because it is side-effect free.[11] By contrast, operations that we have characterized as generators, such as Move-α and Free Indexing, are not deterministic in this sense. That is, given a structure as input, they may produce one or more structures as output.

the "distance" of potential failure filters from the current state of the parser in terms of the number of generators yet to be applied. Then the failing filter will be chosen on the basis of some combination of structure cues and generator distance. Currently, the PO-PARSER uses a slightly different and cheaper scheme. The failure filter is chosen solely on the basis of structure cues. However, the fronting mechanism is restricted so that the chosen filter can only move a limited number of positions ahead of its original position. The original operation sequence is designed such that the distance of the filter from the front of the sequence is roughly proportional to the number of (outstanding) operations that the filter is dependent on.

[10]So far, we have not encountered any linguistic filters that require either structure building or feature assignment. Operations such as θ-role and Case assignment are not considered filters in the sense of the definition given in the previous section. In the PO-PARSER, these operations will never fail. However, definitions that involve some element of 'modality' are potentially problematic. For example, Chomsky's definition of an *accessible SUBJECT*, a definition relevant to the principles of Binding Theory, contains the following phrase "... *assignment to α of the index of β would not violate the (i-within-i) filter* $*[_{\gamma_i} \ldots \delta_i \ldots]$". A transparent implementation of such a definition would seem to require some manipulation of indices. However, Lasnik (p.58) points out that there exists an empirically indistinguishable version of *accessible SUBJECT* without the element of modality present in Chomsky's version.

[11]It turns out that there are situations where a filter operation (although side-effect free) could succeed more than once. For example, the linguistic filter known as the "Empty Category Principle" (ECP) implies that all traces must be "properly governed". A trace may satisfy proper government by being either "lexically governed" or "antecedent governed". Now consider the structure $[_{CP}$ *What*$_1$ *did you* $[_{VP}$ *read* $t_1]]$. The trace t_1 is both lexically governed (by the verb *read*) and antecedent governed (by its antecedent *what*). In the PO-PARSER the ECP operation can succeed twice for cases such as t_1 above.

Given the above hypothesis, we can cut down on the amount of work done by the PO-PARSER by modifying its behaviour for filter operations. Currently, the parser employs a backtracking model of computation. If a particular parser operation fails, then the default behaviour is to attempt to re-satisfy the operation that was called immediately before the failing operation. In this situation, the PO-PARSER will only attempt to re-satisfy the preceding operation if it happens to be a generator. When the preceding operation is a filter, then the parser will skip the filter and, instead, attempt to resatisfy the next most recent operation and so on.[12] For example, consider the following calling sequence:

$$\begin{tikzcd} \boxed{G_1} \arrow[r, dotted] & \boxed{G_2} \arrow[r] & \boxed{F_1} \arrow[r] & \boxed{F_2} \arrow[r] & \boxed{F_3} \end{tikzcd}$$

Suppose that a structure generated by generator G_2 passes filters F_1 and F_2, but fails on filter F_3. None of the three filters could have been the cause of the failure by the side-effect free hypothesis. Hence, we can skip trying to resatisfy any of them and backtrack straight to G_2.

Note that this optimization is just a limited form of dependency-directed backtracking. Failures are traced directly to the last generator invoked, thereby skipping over any intervening filters as possible causes of failure. However, the backtracking behaviour is limited in the sense that the most recent generator may not be the cause of a failure. Consider the above example again. The failure of F_3 need not have been caused by G_2. Instead, it could have been caused by structure-building in another generator further back in the calling sequence, say G_1. But the parser will still try out all the other possibilities in G_2 first.

Consider a situation in which the principle selection procedure performs poorly. That is, for a particular ill-formed structure, the selection procedure will fail to immediately identify a filter that will rule out the structure. The advantages of the modified mechanism over the default backtrack scheme will be more pronounced in such situations — especially if the parser has to try several filters before finding a "failing" filter. By contrast, the behaviour of the modified mechanism will resemble that of the strict chronological scheme in situations where the selection procedure performs relatively well (i.e. when a true failing filter is fronted). In such cases, the advantages, if significant, will be small. (In an informal comparison between the two schemes using about eighty sentences from the reference text, only about half the test cases exhibited a noticeable decrease in parsing time.)

[12] This behaviour can be easily simulated using the 'cut' predicate in Prolog. We can route all calls to filter operations through a predicate that calls the filter and then cuts off all internal choice points. (For independent reasons, the PO-PARSER does not actually use this approach.)

3 Conclusions: The Utility of Principle-Ordering

From our informal experiments with the PO-PARSER, we have found that dynamic principle-ordering can provide a significant improvement over any fixed ordering. We have found that speed-ups varying from three- or four-fold to order-of-magnitude improvements are possible in many cases.[13]

The control structure of the PO-PARSER forces linguistic principles to be applied one at a time. Many other machine architectures are certainly possible. For example, we could take advantage of the independence of many principles and apply principles in parallel whenever possible. However, any improvement in parsing performance would come at the expense of violating the minimum (unnecessary) work ethic. Lazy evaluation of principles is yet another alternative. However, principle-ordering would still be an important consideration for efficient processing in this case. Finally, we should also consider principle-ordering from the viewpoint of scalability. The experience from building prototypes of the PO-PARSER suggests that as the level of sophistication of the parser increases (both in terms of the number and complexity of individual principles), the effect of principle-ordering also becomes more pronounced.

References

[1] Chomsky, N.A., *Lectures on Government and Binding: The Pisa Lectures.* 1981. Foris Publications.

[2] Chomsky, N.A., *Knowledge of Language: Its Nature, Origin, and Use."* 1986. Prager.

[3] Correa, N., "Syntactic Analysis of English with respect to Government-Binding Grammar," Ph.D Dissertation, 1988. Syracuse University.

[4] Dorr, B.J., "UNITRAN: A Principle-Base Approach to Machine Translation," M.I.T. A.I. Technical Report No. 1000. 1987.

[5] Johnson, M., "Knowledge as Language," *ms.* M.I.T. Brain and Cognitive Sciences.

[6] Kolb, H.P., & C. Thiersch, "Levels and Empty Categories in a Principles and Parameters Approach to Parsing," *ms.* 1988. Tilburg University.

[7] Lasnik, H. & J. Uriagereka, *A Course in GB Syntax: Lectures on Binding and Empty Categories.* 1988. M.I.T. Press.

[8] Samuels, A.L., "Some Studies in Machine Learning Using the Game of Checkers. II — Recent Progress," *IBM Journal.* November 1967.

[9] Smith, D.E., & M.R. Genesereth, "Ordering Conjunctive Queries," *Artificial Intelligence* **26** (1985) 171–215.

[10] Stabler, E.P., Jr. "The Logical Approach to Syntax: Foundations, Specifications and Implementations of Theories of Government and Binding." *ms.* 1989. University of Western Ontario.

[13]Obviously, the speed-up obtained will depend on the number of principles present in the system and the degree of 'fine-tuning' of the failure filter selection criteria.

A Probabilistic Parsing Method for Sentence Disambiguation

T. Fujisaki, F. Jelinek, J. Cocke, E. Black, T. Nishino+

IBM Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights, N.Y. 10598
+Tokyo Denki University

1. Introduction

Constructing a grammar which can parse sentences selected from a natural language corpus is a difficult task. One of the most serious problems is the unmanageably large number of ambiguities. Pure syntactic analysis based only on syntactic knowledge will sometimes result in hundreds of ambiguous parses. Martin [15] reported that his parser generated 455 ambiguous parses for the sentence:

List the sales of products produced in 1973 with the products produced in 1972.

Through the long history of work in natural language understanding, semantic and pragmatic constraints have been known to be indispensable for parsing. These should be represented in some formal way and be referred to during or after the syntactic analysis process. AI researchers have been exploring the use of semantic networks, frame theory, etc. to describe both factual and intuitive knowledge for the purpose of filtering out meaningless parses and to aid in choosing the most likely interpretation. The SHRDLU system [22] by Winograd successfully demonstrated the possibility of sophisticated language understanding and problem solving in this direction. However, to represent semantic and pragmatic constraints, which are usually domain sensitive, in a well-formed way is a very difficult and expensive task. To the best of our knowledge, no one has ever succeeded in doing so except in relatively small restricted domains.

Furthermore, there remains a basic question as to whether it is possible to formally encode all of the syntactic, semantic and pragmatic information needed for disambiguation in a definite and deterministic way. For example, the sentence

Print for me *the sales of stair carpets.*

seems to be unambiguous; however, in the ROBOT system pure syntactic analysis of this sentence resulted in two ambiguous parses, because the "ME" can be interpreted as an abbreviation of the state of Maine[9]. Thus, this simple example reveals the necessity of pragmatic constraints for the disambiguation task. Readers may claim that the system which would generate the second interpretation is too lax and that a human would never be perplexed by the case. However, a reader's view would change if he were told that the the sentence below had been issued previous to the sentence above.

Print for ca *the sales of stair carpets.*

Knowing that the speaker inquired about the business in California in the previous queries, it is quite natural to interpret "me" as the state of Maine in this context. A problem of this sort usually calls for the introduction of an appropriate discourse model to guide the parsing. Even with a sophisticated discourse model beyond anything available today, it would be impossible to take account all previous sentences: The critical previous sentence may always be just beyond the capacity of the discourse stack.

Thus it is quite reasonable to think of a parser which disambiguates sentences by referring to statistics which encode various characteristics of the past discourse, the task domain, and the speaker. For instance, the probability that the speaker is referring to states and the probability that the speaker is abbreviating a name, are useful in disambiguating the example. If the probabilities of the above are both statistically low, one could simply neglect the interpretation of the state of "Maine" for "me". Faced with such a situation, we propose, in this paper, to employ probability as a device to quantify language ambiguities. In other words, we will propose a hybrid model for natural language processing which comprises linguistic expertise, i.e. grammar knowledge, and its probabilistic augmentation for approximating natural language. With this framework, semantic and pragmatic constraints are expected to be captured implicitly in the probabilistic augmentation.

Section 2 introduces the basic idea of the probabilistic parsing modeling method and Section 3 presents the experimental results when this modeling method is applied to parsing problems of English sentences and of Japanese noun compound words. Detailed description of the method are given elsewhere.

2. Probabilistic Context-free Grammar

2.1 Extension to Context-free Grammar

A probabilistic context-free grammar is an augmentation of a context-free grammar [5]. Each of the grammar and lexical rules $\{r\}$, having a form of $\alpha \rightarrow \beta$, is associated with a conditional probability $Pr(r) = Pr(\beta \mid \alpha)$. This conditional probability denotes the probability that a non-terminal symbol α, having appeared in the sentential form during the sentence derivation process, will be replaced with a sequence of terminal and non-terminal symbols β. Obviously $\sum_{\beta} Pr(\beta \mid \alpha) = 1$ holds. Processes of sentence generation from a sentence symbol S by a probabilistic context-free grammar will be carried out in an identical manner to the usual non-probabilistic context-free grammar. But the advantage of the probabilistic grammar is that the probability can be computed for each of the derivation trees, which enables us to quantify sentence ambiguities as described below.

The probability of a derivation tree t can be computed as a product of conditional probabilities of the rules which are employed for deriving that tree t.

$$Pr(t) = \prod_{r \in D(t)} Pr(r)$$

Here r denotes a rule of the form $\alpha \rightarrow \beta$, and $D(t)$ denotes an ordered set of the rules which are employed for deriving the tree t. The next figure explains how the probability of a derivation tree t can be computed as a product of rule probabilities.

$$\begin{aligned} Pr(t) = \; & Pr(NP.VP.ENDM \mid S) \times \\ & Pr(DET.N \mid NP) \times \\ & Pr(\textbf{the} \mid \text{det}) \times \\ & Pr(\textbf{boy} \mid N) \times \\ & Pr(V.NP \mid VP) \times \\ & Pr(\textbf{likes} \mid V) \times \\ & Pr(DET.N \mid NP) \times \\ & Pr(\textbf{that} \mid \text{det}) \times \\ & Pr(\textbf{girl} \mid N) \times \\ & Pr(. \mid ENDM) \end{aligned}$$

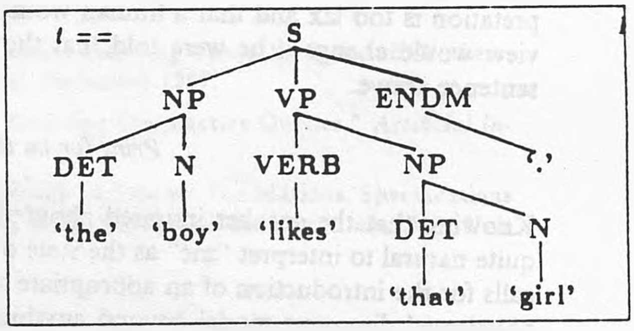


Fig. 1 Probability of a Derivation Tree

An ambiguous grammar allows many different derivation trees to coexist for sentences. From the viewpoint of sentence parsing, we say that a sentence is ambiguous when more than two parsed trees, say $t_1, t_2, \ldots$ are derived from the parsing process. Having a device to compute probability for a derivation tree as shown above, we can handle sentence ambiguity in a quantitative way. Namely, when a sentence s is parsed ambiguously into derivation trees $t_1, t_2, \ldots$ and a probability $Pr(t_i)$ is computed for each derivation tree t_i, the sum of the probabilities $\sum_i Pr(t_i)$ can be regarded as the probability that a particular sentence s will happen to be generated among other infinite possibilities. More interesting is the ratio denoting relative probabilities among ambiguous derivation trees:

$$\frac{Pr(t_j)}{\sum_k Pr(t_k)}$$

We can assume that it should denote the "likelihood" of each derivation tree. For example, consider the following English sentence *"Reply envelopes are enclosed for your convenience."* The sentence is ambiguous because it can be parsed in two different ways; the first being in the imperative mode, and the second in the declarative.

t_1: "Reply (that) envelopes are enclosed for your convenience." $\Rightarrow \dfrac{Pr(t_1)}{(Pr(t_1) + Pr(t_2))}$

t_2: "Reply envelopes (A kind of envelopes) are enclosed for your convenience." $\Rightarrow \dfrac{Pr(t_2)}{(Pr(t_1) + Pr(t_2))}$

These correspond to two different parsed trees, t_1 and t_2. By computing $Pr(t_1) + Pr(t_2)$, we can estimate the probability that the specific sentence *"Reply envelopes are ... "* is generated from among an infinite number of possible sentences. On the other hand, $Pr(t_1)/(Pr(t_1) + Pr(t_2))$ and $Pr(t_2)/(Pr(t_1) + Pr(t_2))$ give measures of likelihood for interpretations t_1 and t_2.

2.2 Estimation of Rule Probabilities from Data

The Forward / Backward algorithm, described in [11], popularly used for estimating transition probabilities for a given hidden-Markov-model, can be extended so as to estimate rule probabilities of a probabilistic context free grammar in the following manner.

Assume a Markov model, whose states correspond to possible sentential forms which appear in a sentence parsing process of a context free grammar. Then each transition between two states of the Markov model corresponds to an application of a context-free rule that maps one sentential form into another. For example, the state $NP.VP$ can be reached from the state S by applying the rule $S \rightarrow NP.VP$ to a start symbol S, the state $ART.NOUN.VP$ can be reached from the state $NP.VP$ by applying the rule $NP \rightarrow ART.NOUN$ to the first NP of the sentential form $NP.VP$, and so on. Since each rule corresponds to a state transition between two states, parsing a set of sentences given as training data will enable us to count how many times each transition is traversed. In other words, it tells how many times each rule is fired when the given set of sentences is generated. For example, the transition from the state S to the state $NP.VP$ may happen most frequently because the rule $S \rightarrow NP.VP$ is commonly used in almost every declarative sentence; while the transition from the state $ART.NOUN.VP$ to the state every.$NOUN.VP$ may happen 103 times; etc. In a context-free-grammar, each replacement of a non-terminal symbol occurs independently of the context. Therefore, counts of all transitions between states $\alpha.A.\beta$ to $\alpha.B.C.\beta$, with arbitrary α and β, should be tied together.

Counting the transitions in such a way for thousands of sentences will enable us to estimate the rule probabilities $\{Pr(\beta \mid \alpha)\}$ which are the probabilities that left hand side non-terminal symbols α will be replaced with right hand side patterns β. The actual iteration procedure to estimate these probabilities from N sentences $\{B^i\}$ is shown below.

1. Make an initial guess of $\{Pr(\beta \mid \alpha)\}$ such that $\sum_\beta Pr(\beta \mid \alpha) = 1$ holds.
2. Parse each output sentence B^i. Assume that grammar is ambiguous and that more than one derivation path exists which generate the given sentence B^i. In such cases, we denote D^i_j as the j-th derivation path for the ith-sentence.
3. Compute the probability of each derivation path D^i_j in the following way:

$$Pr(D^i_j) = \prod_{r \in D^i_j} Pr(r)$$

This computes $Pr(D^i_j)$ as a product of the probabilities of the rules $\{r\}$ which are employed to generate that derivation path D^i_j .

4. Compute the Bayes *a posteriori* estimate of the count $C^i_\alpha(\beta)$ which represents how many times the rule $\alpha \rightarrow \beta$ was used for generating the sentence B^i .

$$C^i_\alpha(\beta) = \sum_j \left(\frac{Pr(D^i_j)}{\sum_k Pr(D^i_k)} \times n^i_j(\alpha, \beta) \right)$$

Here, $n^i_j(\alpha, \beta)$ denotes the number of times the rule $\alpha \rightarrow \beta$ is used on the derivation path D^i_j .

5. Normalize the count so that the total count for rules with same left hand side non-terminal symbol α becomes 1.

$$f_\alpha(\beta) = \sum_i \frac{C^i_\alpha(\beta)}{\sum_\gamma C^i_\alpha(\gamma)}$$

6. Replace $\{Pr(\beta \mid \alpha)\}$ with $\{f_\alpha(\beta)\}$ and repeat from step 2.

Through this process, the $\{Pr(\beta \mid \alpha)\}$ will approach the real transition probability[2,10]. This algorithm has been proven to converge [3].

2.3 Parsing Procedure which computes Probabilities

To find the most-likely parse, that is, the parse tree which has the highest probability from among all the candidate parses, requires a lot of time if we calculate probabilities separately for each ambiguous parse. The following is a parsing procedure based on the Cocke-Kasami-Young [1] bottom-up parsing algorithm which can accomplish this task very efficiently. By using it, the most-likely parse tree for a sentence will be obtained while the normal bottom-up parsing process is performed. It gives the maximum probability $\mathrm{Max}_j Pr(t_j)$ as well as the total probability of all parses $\sum_j Pr(t_j)$ at the same time.

The Cocke-Kasami-Young parsing algorithm maintains a two-dimensional table called the Well-Formed-Substring-Table (WFST). An entry in the table, *WFST(i,j)* , corresponds to a substring(i,j), j words in length, starting at the i-th word, of an input sentence [1]. The entry contains a list of triplets. An application of a rule $\alpha \rightarrow \beta\gamma$ will add an entry (α, β, γ) to the list. This triplet shows that a sequence of $\beta.\gamma$ which spans substring(i,j) is replaced with a non-terminal symbol α. (β: is the pointer to another *WFST* entry that corresponds to the left subordinate structure of α and γ : is the pointer to the right subordinate structure of α.)

In order to compute probabilities of parse trees in parallel to this bottom-up parsing process, the structure of this WFST entry is modified as follows. Instead of having an one-level flat list of triplets, each entry of WFST was changed to hold a two-level list. The top-level of the two-level list corresponds to a left hand side non-terminal symbol, called as LHS symbol hereinafter. All combinations of left and right subordinate structures are kept in the sub-list of the LHS symbol. For instance, an application of a rule $\alpha \rightarrow \beta\gamma$ will add (β, γ) to the sub-list of α.

In addition to the sub-list, a LHS symbol is associated with two variables - *MaxP* and *SumP*. These two variables keep the maximum and the total probabilities of the LHS symbol of all possible right hand side combinations. MaxP and SumP can be computed in the process of bottom-up chart parsing. When a rule $\alpha \rightarrow \beta\gamma$ is applied, $MaxP$ and $SumP$ are computed as:

$$MaxP(\alpha) = \max_{\beta,\gamma}(Prob(\alpha \rightarrow \beta\gamma) \times MaxP(\beta) \times MaxP(\gamma))$$

$$SumP(\alpha) = \sum_{\beta,\gamma}(Prob(\alpha \rightarrow \beta\gamma) \times SumP(\beta) \times SumP(\gamma))$$

This procedure is similar to that of Viterbi algorithm[4] and maintains the maximum probability and the total probability in $MaxP$ and $SumP$ respectively. $MaxP/SumP$ gives the maximum relative probability of the most-likely parse.

3. Experiments

To demonstrate the capability of the modeling method, a few trials were made to disambiguate corpora of highly ambiguous phrases. Two of these experiments will be briefly described below. Details can be found elsewhere.

3.1 Disambiguation of English Sentence Parsing

As the basis of this experiment, the grammar developed by Prof. S. Kuno in the 1960's for the machine translation project at Harvard University [13,14,18] was used with some modification. The set of grammar specifications in the Kuno grammar, which are in Greibach Normal form, were translated into a form which is more favorable to our method. The 2118 original rules were reformulated into 7550 rules in Chomsky normal form[1].

Training sentences were chosen from two corpora. One corpus is composed of articles from *Datamation* and *Reader's Digest* (average sentence length in words 10.85, average number of ambiguities per sentence 48.5) and the other from business correspondence (average sentence length in words 12.65, average number of ambiguities per sentence 13.5). A typical sentence from the latter corpus is shown below:

It was advised that there are limited opportunities at this time.

The 3582 sentences from the first corpus, and 624 sentences from the second corpus that were successfully parsed were used to train the 7550 grammar rules besides some lexical rules in each corpus.

Once the probabilities for rules are thus obtained, they can be used to disambiguate sentences by the procedure described in section 2.3.

```
    SENTENCE
      PRONOUN      ( we )
      PREDICATE
        AUXILIARY    ( do )
        INFINITE VERB PHRASE
          ADVERB TYPE1 ( not )
  (A) 0.356 INFINITE VERB PHRASE
   :      VERB TYPE IT1( utilize )
   :      OBJECT
   :        NOUN          ( outside )
   :        ADJ CLAUSE
   :          NOUN          ( art )
   :          PRED. WITH NO OBJECT
   :            VERB TYPE VT1 ( services )
  (B) 0.003 INFINITE VERB PHRASE
   :      VERB TYPE IT1( utilize )
   :      OBJECT
   :        PREPOSITION  ( outside )
   :     NOUN OBJECT
   :       NOUN       ( art )
   :     OBJECT
   :       NOUN         ( services )
(C) 0.641 INFINITE VERB PHRASE
   :   VERB TYPE IT1( utilize )
   :   OBJECT
   :     NOUN        ( outside )
   :     OBJECT MASTER
   :       NOUN        ( art )
   :       OBJECT MASTER
   :         NOUN          ( services )
  PERIOD
    ADVERB TYPE1 ( directly )
    PRD          ( . )
```

Fig. 2 Parse Tree for "We do not utilize . . ."

Figure 2 shows the parsing result for the sentence *"we do not utilize outside art services directly."*, which turned out to have three ambiguities.

As shown in the figure, ambiguities come from the three distinct substructures, (A), (B) and (C), for the phrase *"utilize outside art services."*. The derivation (C) corresponds to the most common interpretation while in (A) *"art"* and *"outside"* are regarded respectively as subject and object of the verb *"services"*. In (B), *"art service"* is regarded as an object of the verb *"utilize"* and *"outside"* is inserted as a preposition. The numbers 0.356, 0.003 and 0.641 signify the relative probabilities of the three interpretations. As shown in this case, the correct parse (the third one) gets the highest relative probability, as was expected.

Some of the resultant probabilities obtained through the iteration process for each of the grammar rules and the lexical rules are shown below.

Rules for "IT6"[1]

(0.11054)	IT6 → BELIEVE	--(a)
(0.10685)	IT6 → KNOW	--(b)
(0.08562)	IT6 → FIND	
(0.07628)	IT6 → THINK	
(0.03525)	IT6 → CALL	
(0.03280)	IT6 → REALIZE	
.	.	

Rules for "IT3"[2]

(0.16055)	IT3 → GET
(0.12447)	IT3 → MAKE
(0.11988)	IT3 → HAVE
(0.08132)	IT3 → SEE
(0.06477)	IT3 → KEEP
(0.06363)	IT3 → BELIEVE
.	.

Rules for "SE"[3]

(0.21602)	SE → AAA 4X VX PD	---(c)
(0.15296)	SE → PRN VX PD	---(d)
(0.15229)	SE → NNN VX PD	
(0.11965)	SE → AV1 SE	
(0.04730)	SE → PRE NQ SE	
(0.04457)	SE → NNN AC VX PD	
(0.02616)	SE → AV2 SE	
.	.	

Rules for "VX"[4]

(0.19809)	VX → VT1 N2
(0.10704)	VX → PRE NQ VX
(0.08790)	VX → VI1
(0.07500)	VX → AUX BV
(0.05455)	VX → AV1 VX
.	.

Fig. 3 Rule probabilities estimated by iteration

Numbers in the parentheses on the left of each rules denote probabilities estimated from the iteration process described in the section 3.3. For example, the probabilities that the words **believe**, and **know** have the part of speech **IT6** are shown as 11.1\% and 10.7\% on lines (a) and (b) respectively. Line (c) shows that a sequence AAA (article and other adjective etc.) 4X (subject noun phrase), VX(predicate) and PD (period or post sentential modifiers followed by period) forms a sentence (SE) with probability 21.6\%. Line (d), on the other hand, shows that a sequence PRN

1 Infinite form of a mono-transitive verb which takes a noun-clause object
2 Infinite form of a complex-transitive verb which takes an object and an objective compliment
3 sentence
4 predicate

(pronoun), VX and PD forms a sentence (SE) with probability 15.3 %. In such ways, the probability findings convey useful information for language analysis.

Table 1 summarizes the experiments. Test 1 corresponds to the corpus of articles from *Datamation* and *Reader's Digest*, while Test 2 derived from the business correspondence. In both cases, the base Kuno grammars were successfully augmented by probabilities.

a.	Corpus	test1	test2
b.	Number of sentences used for training	3582	624
c.	Number of sentences checked manually	63	21
d.	Number of sentences with no correct parse	4	2
e.	Number of sentences where highest prob. was given to the most natural parse	54	18
f.	Number of sentences where highest prob. was not given to the right one	5	1

Table 1. Summary of English sentence parsing

3.2 Disambiguation of Japanese Noun Compound Word Parsing

Analyzing structures of noun compound words is difficult because noun compound words usually do not have enough structural clues for syntactic parsing[17]. Particularly in the Japanese language, noun compound words consist only of a few types of components, and pure syntactic analysis will result in many ambiguous parses. Some kind of mechanism which can handle inter-word analysis of constituent words is needed to disambiguate them.

We applied our probabilistic modeling method for disambiguating parsing of Japanese noun compound words. It was done by associating rule probabilities to basic construction rules of noun compound words. In order to make rule probabilities sensitive to inter-word relationship of component words, words were grouped into finer categories $\{N_1, N_2, N_3, \ldots N_m\}$. The base rules were replicated for each combination of right hand side word categories. Since we assumed that the right-most word of the right hand side inherits the category from the left hand side parent, a single $N \rightarrow NN$ rule was replicated to $m \times m$ rules. For these $m \times m$ rules, separate probabilities were prepared and estimated. The method described in the section 2.2 was used to estimate these probabilities from noun compound words actually observed in text.

Once probabilities for rules were estimated, the parsing procedure described in the section 2.3 was used to compute relative probability of each parse tree i.e. the likelihood of the parse tree among others.

In this experiment, we categorized words by a conventional clustering technique which groups words according to neighboring word patterns. For example, "oil" and "coal" belong to the same category in our method because they frequently appear in similar word patterns such as " ~ burner", " ~ consumption", " ~ resources". 31,900 noun compound words picked from abstracts of technical papers [12] were used for this categorization process. Twenty eight categories were obtained through this process for 1000 high-frequency 2-character kanji primitive words, 8 categories for 200 prefix single-character words, and 10 categories for 400 suffix single-character words[16]. Base rules deriving from different combination of these 46 word categories resulted in 5582 separate rules. These base rules are displayed below.

<word> → <2 character kanji primitive word>

<word> → <word> <word>

< word > → *< prefix single character word >* *< word >*

< word > → *< word >* *< suffix single character word >*

5582 conditional probabilities of these rules were estimated from 28,568 noun compound words.

After training was successfully done, 153 noun compound words were randomly chosen, parsed by the procedure shown in the section 3.3 and the parse trees were examined by hand. The check was made whether the correct parse is given the highest probabilities. Among the 153 test words, 22 was uniquely parsed and 131 test words were parsed with more than two alternative parse trees. Among 131, in 92 cases, the right parses were given the highest probabilities.

Show below are parsing results for two noun compound words.

word 1: 中(medium) 規模(scale) 集積(integrated) 回路(circuit)

word 2: 小(small) 規模(scale) 電力(electricity) 会社(company)

(Word order is the same both in English and in Japanese).

For both of these cases, 5 alternative parse trees were given. Obtained parse trees were computed with relative probabilities, the likelihood, among other alternative parses. In the first sentence, the 5-th parse tree, which is the most natural, got the highest probability 0.43. In the second case, the 3rd parse tree, which is the most natural, got the highest probability 1.0.

word 1 "medium scale integrated circuit"			
	structure of parsed tree shown in bracket notation	meaning implied from structure	prob.
1	medium [[scale integrated] circuit]	a medium-size "scale-integrated-circuit"	0.17
2	medium [scale [integrated circuit]]	a medium-sized integrated circuit which is scale (?)	0.04
3	[medium scale] [integrated circuit]	an integrated-circuit of medium-scale	0.19
4	[medium [scale integrated]] circuit	a medium-size circuit which is scale-integrated	0.17
5	[[medium scale] integrated] circuit	a circuit which is medium-scale integrated	0.43
case 2 "small scale electricity company"			
1	small [[scale electricity] company]	a small company which serves scale-electricity	0.0
2	small [scale [electricity company]]	a company which is small, serves electricity, and is something to do with scale	0.0
3	[small scale] [electricity company]	a company which serves electricity and which is small scale	1.0
4	[small [scale electricity]] company	a company which services small scale-electricity	0.0

5	[[small scale] electricity] company	a company which services small scale electricity (micro electronics?)	0.0

4. Concluding Remarks

N-gram modeling technique [20] has been proven to be a powerful and effective method for language modeling. It has successfully been used in several applications such as speech recognition, text segmentations, character recognition and others.[11,6,7,19,21] At the same time, however, it has proved to be difficult to approximate language phenomena precisely enough when context dependencies expand over a long distance. A direct means to remedy the situation is (a) to increase N of N-gram or (b) to increase the length of basic units from a character to a word or to a phrase. If the vocabulary size is M, however, the statistics needed for maintaining the equivalent precision in the N-gram model increase in proportion to M^N. The situation is similar in (b). Increasing the length of the basic unit causes an exponential increase in vocabulary size. Hence an exponential increase of the required statistics volume follows in (b) as well. This shows that the N-gram model faces a serious data gathering problem when a task has a long-context dependency. Obviously, the parsing of sentences creates this sort of problem.

On the other hand, the method introduced here aims to remedy this problem by combining a probabilistic modeling procedure with linguistic expertise. In this hybrid approach [7,8], linguistic expertise provides the framework of a grammar, and the probabilistic modeling method augments the grammar quantitatively.

Since the probabilistic augmentation process is completely automatic, it is not necessary to rely on human endeavor which tends to be expensive, inconsistent, and subjective. Also the probabilistic augmentation of a grammar is adaptable for any set of sentences.

These two important features make the method useful for various problems of natural language processing. Besides its use for sentence disambiguation demonstrated in the section 3.4, the method can be used to customize a given grammar to a particular sub-language corpus. Namely, when a grammar designed for a general-corpus is applied to this method, the rules and the lexical entries which are used less frequently in the corpus will automatically be given low or zero probabilities. Alternately, the rules and the lexical entries which require more refinement will be given high probabilities, thus the method helps us to tune a grammar in a top-down manner. The method is also useful for improving performance of top-down parsing when used for obtaining hints for reordering rules according to the rule probabilities.

In this way, although all possible uses have not been explored the method proposed in this paper has enormous potential application, and the author hopes that a new natural language processing paradigm may emerge from it.

Use of probability in natural language analysis may seem strange, but it is in effect a only simple generalization of common practice: Namely, the usual top-down parsing strategy forces a true or false (1 or 0) decision, i.e. to choose one alternatives from others on every non-deterministic choice point.

And most importantly, by use of the proposed method a grammar can be probabilistically augmented objectively and automatically from a set of sentences picked from an arbitrary corpus. On the other hand, the representation of semantic and pragmatic constraints in the form of usual semantic networks, frame theory, etc., requires a huge amount of subjective human effort.

Acknowledgement

The authors are indebted to Dr. B. Mercer, and Dr. L. Bahl of the IBM Thomas J. Watson Research Center, for their valuable technical suggestions and comments.

Prof. S. Kuno of Harvard University has not only given the senior author permission to use the Kuno grammar, but also much valuable linguistic advice.

References

[1] A.V. Aho, J.D. Ullman, *The Theory of Parsing, Translation and Compiling*, Vol. 1, Prentice-Hall, 1972

[2] J.K. Baker, *Trainable Grammars for Speech Recognition*, internal memo, 1982

[3] L.E. Baum, *A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions in Markov Chains*, Vol. 41, No. 1, The Annals of Mathematical Statistics, 1970

[4] G.D. Forney, Jr., *The Viterbi Algorithm*, Proc. of IEEE, Vol. 61, No. 3, 1973

[5] K.S. Fu, *Syntactic Methods in Pattern Recognition* , Vol. 112, Mathematics in Science and Engineering, Academic Press, 1974

[6] T. Fujisaki, *A Scheme of Separating and Giving Phonetic Transcriptions to Kanji-kana Mixed Japanese Documents by Dynamic Programming*,(in Japanese), Natural Language Workshop Report NL28-5, Inf. Proc. Soc. of Japan, 1981

[7] T.Fujisaki, *Handling of Ambiguities in Natural Language Processing* (in Japanese), Doctoral dissertation, Dept. of Information Science, Univ. of Tokyo, 1985

[8] T. Fujisaki, *An Approach to Stochastic Parsing*, Proc. of COLLING84, 1984

[9] L. Harris *Experience with ROBOT in 12 Commercial Natural Language Database Query Applications*, Proc. of IJCAI, 1979.

[10] F. Jelinek, *Notes on the Outside Inside Algorithm*, internal memo, 1983

[11] F. Jelinek, et. al, *Continuous Speech Recognition by Statistical Methods*, Proc. of the IEEE, Vol. 64, No. 4, 1976

[12] MT for Electronics, *Science and Technology Database*(in tape form), Japanese Information Center for Science and Technology, Vol. 26

[13] S. Kuno, *The Augmented Predictive Analyzer for Context-free Languages and Its Relative Efficiency*, CACM, Vol. 9, No. 11, 1966

[14] S. Kuno, A.G. Oettinger, *Syntactic Structure and Ambiguity of English*, Proc. of FJCC, 1963

[15] W.A. Martin, et al., *Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results*, MIT LCS Report TR-261, 1979

[16] T. Nishno, T. Fujisaki, *Probabilistic Parsing of Kanji Compound Words* (in Japanese), J. of Inf. Proc. Soc. of Japan, Vol 29, No.11, 1988

[17] T. W. Finin., *Constraining the interpretation of nominal compounds in a limited cpmtext*, In R. Grishman and R. Kittredge, editors, Analysing Language in a Restricted Domian, Lawrence Erlbaum Assoc., Hillsdale, 1986

[18] A.G. Oettinger, *Report No. NSF-8: Mathematical Linguistics and Automatic Translation*, The Computation Laboratory, Harvard Univ., 1963

[19] J. Raviv, *Decision Making in Markov Chains Applied to the Problem of Pattern Recognition*, IEEE, Trans. Information Theory, Vol. IT-3, No. 4, 1967

[20] C.E. Shannon, *Prediction and Entoropy of Printed English*, Bell Sys. Tech. J., Vol. 30, 1951

[21] K. Takeda, T. Fujisaki, *Segmentation of Kanji Primitive Word by a Stochastical Method*(in Japanese), J. of Inf. Proc. Soc. of Japan, Vol 28, No.9, 1987

[22] T. Winograd, *Understanding Natural Language* New York, Academic Press, 1972

[23] T. Winograd, *Language as a Cognitive Precess* Vol. 1 Syntax, Addison Wesley, 1983

A Sequential Truncation Parsing Algorithm based on the Score Function

Keh-Yih Su[*], Jong-Nae Wang[**], Mei-Hui Su[**] and Jing-Shin Chang[***]

[*]Department of Electrical Engineering
National Tsing Hua University, Hsinchu, Taiwan, R.O.C.

[**]BTC R&D Center
28 R&D Road II, 2F
Science-Based Industrial Park, Hsinchu, Taiwan, R.O.C.

[***]Institute of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

Key Words: Machine Translation, Parsing Strategy, Score Function, Sequential Truncation Parsing Algorithm

ABSTRACT

In a natural language processing system, a large amount of ambiguity and a large branching factor are hindering factors in obtaining the desired analysis for a given sentence in a short time. In this paper, we are proposing a sequential truncation parsing algorithm to reduce the searching space and thus lowering the parsing time. The algorithm is based on a score function which takes the advantages of probabilistic characteristics of syntactic information in the sentences. A preliminary test on this algorithm was conducted with a special version of our machine translation system, the ARCHTRAN, and an encouraging result was observed.

Motivation

In a natural language processing system, the number of possible analyses associated with a given sentence is usually large due to the ambiguous nature of natural languages. But, it is desirable that only the best one or two analyses are translated and passed to the post-editor in order to reduce the load of the post-editor. Therefore, in a practical machine translation system, it is important to obtain the best (in probabilistic sense) syntax tree having the best semantic interpretation within a reasonably short time. This is only possible with an intelligent parsing algorithm that can truncate undesirable analyses as early as possible.

There are several methods to accelerate the parsing process [Su 88b], one of which is to decrease the size of the searching space. This can be accomplished with a scored parsing algorithm that truncates unlikely paths as early as possible [Su 87a, 87b] and hence decreases the parsing time.

As for the searching strategy for the scored parsing algorithm, it may be either parallel or sequential. But in our system, a time limit is used to stop the parsing process when a sentence is taking too long to parse because its length or because it has a very complicated structure. Therefore, the sequential searching strategy is better for us than the parallel approach because we are likely to have some complete syntax trees to work with even if the parsing was suspended abnormally when its time expires. On the other hand, the parallel approach will not have this advantage because none of the on-going paths have traversed to the end.

In this paper, we are proposing a sequential truncation algorithm for parsing sentences efficiently. This algorithm employs the score function we proposed in [Su 88a]. However, this algorithm is different from the one proposed in [Su 87a, 87b], which described a parallel truncation algorithm for scored parsing. Here, we are adopting a sequential truncation method. While we are using this sequential approach, a large speed-up in the parsing time has been observed.

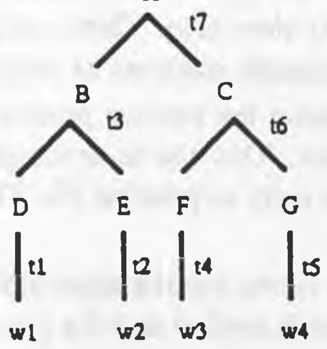
Definition of the Score Function

In a scored parsing system, the best analysis is selected base on its score. Several scoring mechanisms have been proposed in the literatures [Robi 83, Benn 85, Gars 87, Su 88a]. The one we adopt is the score function based on the conditional probability we proposed in [Su 88a]. How to select the best analysis of a sentence is now converted into the problem of finding the semantic interpretation (*Sem$_i$*), the syntactic structure (*Syn$_j$*) and the lexical categories (*Lex$_k$*) that maximize the conditional probability of the following equation,

$$\begin{aligned} &SCORE(Sem_i, Syn_j, Lex_k) \\ &\quad = P(Sem_i, Syn_j, Lex_k | w_1 \ldots w_n) \\ &\quad = P(Sem_i | Syn_j, Lex_k, w_1 \ldots w_n) * P(Syn_j | Lex_k, w_1 \ldots w_n) * P(Lex_k | w_1 \ldots w_n) \\ &\quad = SCORE_{sem}(Sem_i) * SCORE_{syn}(Syn_j) * SCORE_{lex}(Lex_k) \quad , \end{aligned} \tag{1}$$

where w_1 to w_n stands for the words in the given sentence and the last three product terms are semantic score, syntactic score and lexical score respectively. Since we are using just the syntactic information in our current implementation, we will focus only on the syntactic aspect of this score function (i.e. *SCORE$_{syn}$(Syn$_j$)*, which can be approximated by $SCORE_{syn}(Syn_j) \approx P(Syn_j | Lex_k) = P(Syn_j | v_1 \ldots v_n)$, where v_1 to v_n are the lexical categories corresponding to w_1 to w_n).

To show the mechanism informally, first refer to the syntax tree in Fig. 1. shown here with its reduction sequences (produced with a bottom-up parsing algorithm), where L_i is i-th phrase level consists of terminals and nonterminals. The transition from a phrase level L_i to the next phrase level L_{i+1} corresponds to a reduction or derivation of a nonterminal at time t_i.



L8 - { A, }
L7 - { B, C }
L6 - { B, F, G }
L5 - { B, F, W4 }
L4 - { B, W3, W4 }
L3 - { D, E, W3, W4 }
L2 - { D, W2, W3, W4 }
L1 - { W1, W2, W3, W4 }

Fig. 1 Different Phrase Levels for a bottom-up Parsing

The syntax score of the tree in Fig. 1 can be formulated as the following conditional probability equation, where l_i and r_i are the left and right contexts of the reducing symbols:

$$\begin{aligned} SCORE_{syn}(Syn_A) & \\ &= P(L_8, L_7 ... L_2 | L_1) \\ &= P(L_8 | L_7 ... L_2, L_1) * P(L_7 | L_6 ... L_1) * ... * P(L_2 | L_1) \\ &\approx P(\{A\} | \{l_7, B, C, r_7\}) * P(\{C\} | \{l_6, F, G, r_6\}) * ... * P(\{D\} | \{l_1, w_1, r_1\}) \end{aligned} \quad (2)$$

Eq. 2 can be further reduced to the following equation if only one left and one right context symbol are considered where "$\emptyset$" is the null symbol.

$$\begin{aligned} SCORE_{syn}(Syn_A) & \\ &\approx P(\{A\} | \{\emptyset, B, C, \emptyset\}) * P(\{C\} | \{B, F, G, \emptyset\}) * ... * P(\{D\} | \{\emptyset, w_1, w_2\}) \end{aligned} \quad (3)$$

If we want to calculate the score at the point where a word is just being fetched (compact multiple reductions and one shift into one step), the $SCORE_{syn}(Syn_A)$ can also be approximated into the following equation.

$$\begin{aligned} SCORE_{syn}(Syn_A) & \\ &= P(L_8, L_7 ... L_2 | L_1) \\ &= P(L_8, L_7, L_6 | L_5, L_4 ... L_1) * P(L_5 | L_4, L_3 ... L_1) * P(L_4, L_3 | L_2, L_1) * P(L_2 | L_1) \\ &\approx P(L_8, L_7, L_6 | L_5) * P(L_5 | L_4) * P(L_4, L_3 | L_2) * P(L_2 | L_1) \\ &\approx P(L_8 | L_5) * P(L_5 | L_4) * P(L_4 | L_2) * P(L_2 | L_1) \end{aligned} \quad (4)$$

Two assumptions were made in formulating Eq. 2–4. First, it is assumed that the forming of phrase level i is only dependent on its immediate lower phrase level, since most information percolated from other lower levels is contained in that level. And second, a reduction is only locally context sensitive to its left or right context at each phrase level. This assumption is also supported in other systems as well [Marc 80, Gars 87].

A simulation based solely on this syntactic score was conducted and reported in [Su 88a] with a full-path searching algorithm. The result shows that the correct syntactic structures of over 85% of the test sentences were successfully picked when a total of three local left and right context symbols were consulted.

The Sequential Truncation Algorithm

Using the score function defined in the previous section, we will present the idea of sequential truncation algorithm with Fig. 2.

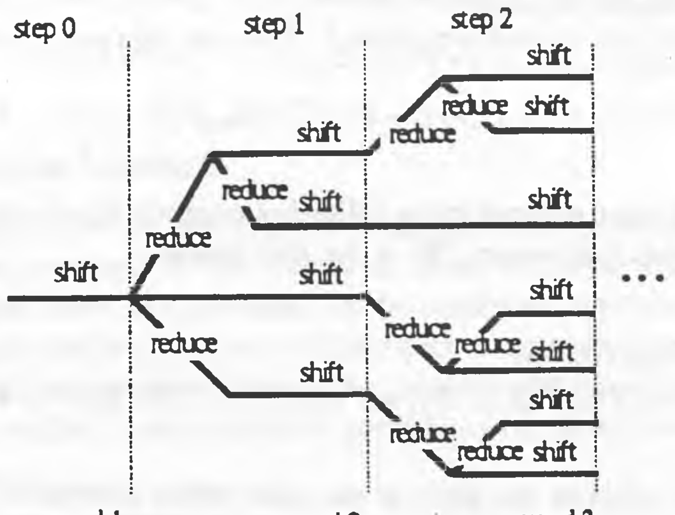


Fig. 2 The searching tree

Each path in Fig. 2 corresponds to a possible derivation of a given sentence. The parser will use the depth-first strategy to traverse the searching tree. But during the searching process, the parser compares the score of each path accumulated so far with a running threshold $C(\alpha_i)$ (a detailed definition will be given in the following section) at each step i when the next word is fetched. If the score of the path is less than the running threshold $C(\alpha_i)$, it will be truncated, i.e. blocked, and the next path will be tried. This process continues until we get the first complete parse tree (i.e. when the whole sentence is reduced to a S node). After we obtain the first complete parse tree, a lower bound for the scores is acquired. The parser will continue to traverse other pathes, but from now on, the score of each path will also be compared with the final accumulated score of the first complete parse tree in addition to be compared with the running threshold. This additional comparison is similar to the branch and bound strategy employed in many AI applications [Wins 84] and it will accelerate the parsing process further. The whole process is shown in the flow chart in Fig. 3. If the test fails in either case, this path will be truncated. Continuing in this manner, we may get a second complete parse tree which has a final score higher than the first one. In this case, we will replace the lower bound with the final score of the second parse tree and repeat the whole process until the end of the entire searching process.

If all the paths are blocked without arriving at any complete parse tree, we can adopt one of two possible strategies. First, we could loosen the running thresholds, i.e. lowering the $C(\alpha_i)$, and try the deepest path gone so far again. Second, we can process this sentence in fail-soft mode. The fail-soft mechanism will skip and discard the current state and attempts to continue the parsing at some later point.

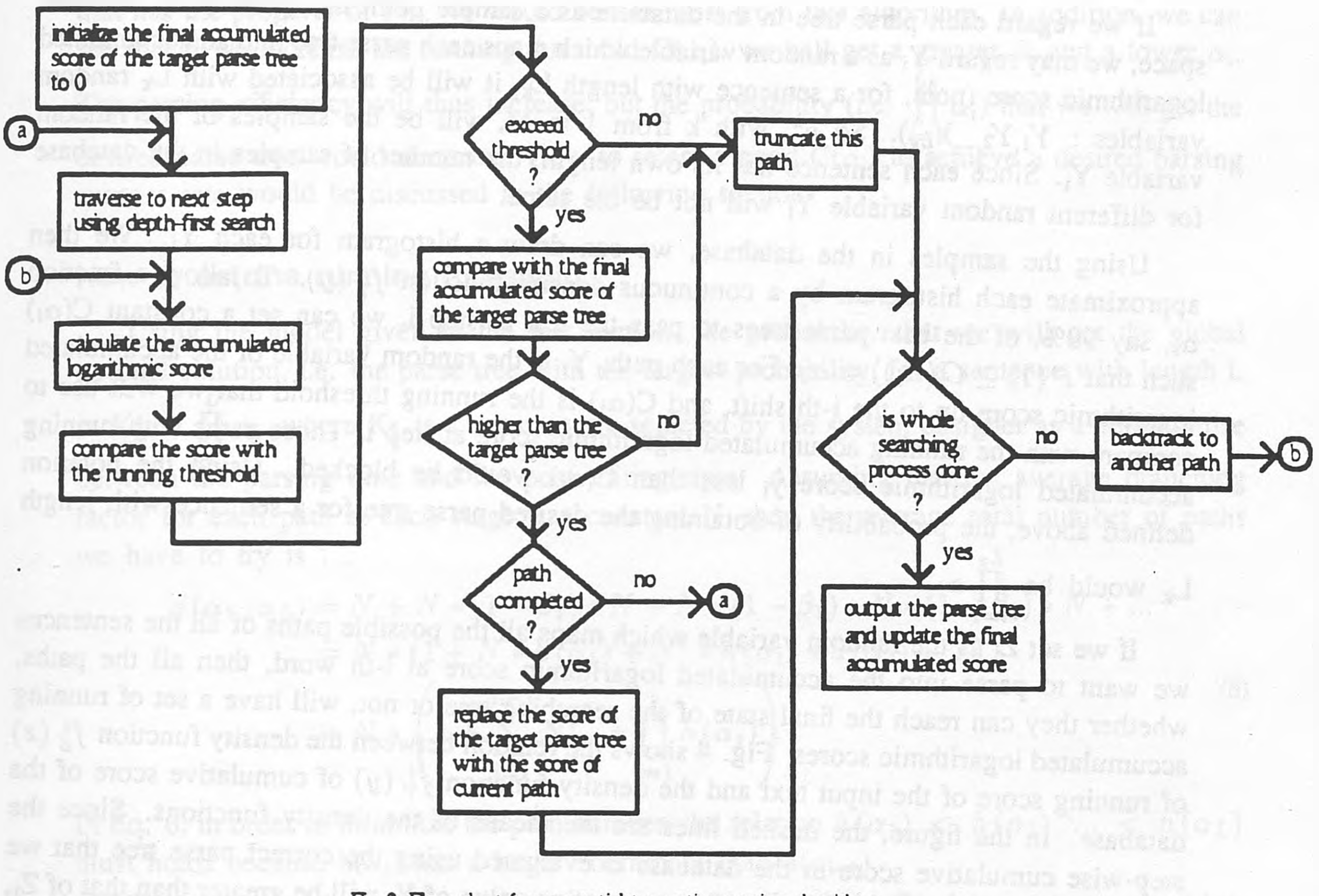

initialize the final accumulated score of the target parse tree to 0
a
traverse to next step using depth-first search
b
calculate the accumulated logarithmic score
compare the score with the running threshold
exceed threshold ?
no
yes
truncate this path
compare with the final accumulated score of the target parse tree
higher than the target parse tree
no
yes
path completed ?
no
a
yes
replace the score of the target parse tree with the score of current path
is whole searching process done ?
no
backtrack to another path
b
yes
output the parse tree and update the final accumulated score

Fig. 3 Flow chart for sequential truncation parsing algorithm

The effectiveness of the sequential truncation algorithm depends on the distribution of scores of the database and the input sentences. As we can see, for each syntax tree can be expressed as the product of a sequence of conditional probability as shown in Eq. 4. Each term in the product corresponds to a transition between two "shift" actions and is evaluated immediately after a "shift". Taking the logarithm on both sides of Eq. 4, we get the following equation where X_i denotes a sequence of phrase levels at i-th step and L is the length of the sentence.

$$log\,(SCORE_{syn}\,(Syn)) = \sum_{i=1}^{L} log\ P(X_i|X_{i-1}) \qquad (5)$$

If we define $y_j = \sum_{i=1}^{j} log\ \ P(X_i|X_{i-1})$, then y_j denotes the accumulated logarithmic score up to the j-th word which is also the j-th shift of the sentence.

Suppose we have M sentences with their correct parse trees in the database. For each parse tree, we can evaluate y_j by using the logarithmic score function defined before. So for the k-th sentence in the database, we obtain a sequence $y_1^k,\ y_2^k,\ \cdots y_{L_k}^k$, where y_j^k denotes the accumulated logarithmic score of the k-th sentence and L_k denotes the length of the k-th sentence.

If we regard each parse tree in the database as a sample point in a probability outcome space, we may regard Y_i as a random variable which maps each parse tree into an accumulated logarithmic score (note, for a sentence with length L_k, it will be associated with L_k random variables : $Y_1, Y_2, \ldots Y_{L_k}$). So y_i^k, with k from 1 to M, will be the samples of the random variable Y_i. Since each sentence has its own length, the number of samples in the database for different random variable Y_i will not be the same.

Using the samples in the database, we can draw a histogram for each Y_i. We then approximate each histogram by a continuous density function $f_Y^i(y)$. To allow a fraction α_i, say 99%, of the best parse trees to pass the test at step i, we can set a constant $C(\alpha_i)$ such that $P(Y_i \geq C(\alpha_i)) = \alpha_i$. For each path, Y_i is the random variable of the accumulated logarithmic score up to the i-th shift, and $C(\alpha_i)$ is the running threshold that we will use to compare with the running accumulated logarithmic score at step i. Those paths with running accumulated logarithmic score y_i less than $C(\alpha_i)$ would be blocked. Using the notation defined above, the probability of obtaining the desired parse tree for a sentence with length L_k would be $\prod_{i=1}^{L_k} \alpha_i$.

If we set Z_i as the random variable which maps all the possible paths of all the sentences we want to parse into the accumulated logarithmic score at i-th word, then all the paths, whether they can reach the final state of the searching tree or not, will have a set of running accumulated logarithmic scores. Fig. 4 shows the relation between the density function $f_Z^i(z)$ of running score of the input text and the density function $f_Y^i(y)$ of cumulative score of the database. In the figure, the dashed lines are the means of the density functions. Since the step-wise cumulative score in the database is evaluated using the correct parse tree that we have selected, we would expect that the expectation value of Y_i will be greater than that of Z_i, that is, $E[Y_i] > E[Z_i]$; and the variance of Y_i is less than that of Z_i, that is $Var[Y_i] < Var[Z_i]$.

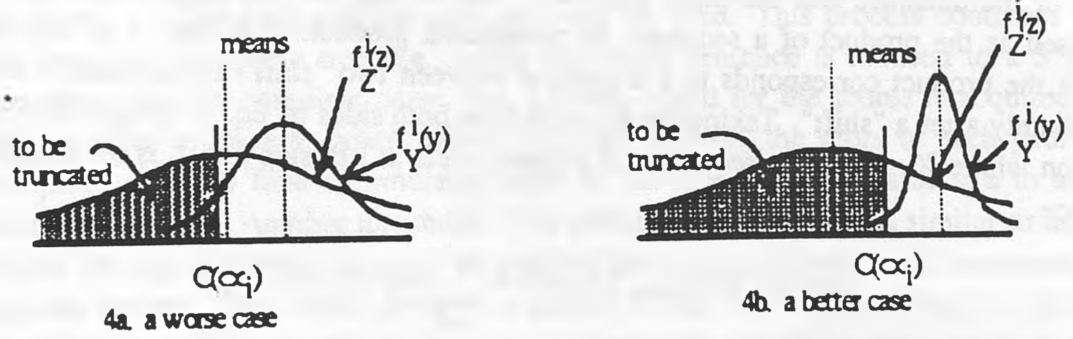


Fig. 4 Relationship between the running score of the input text and the cummulative score of the database

Let β_i denotes $F_Z^i(C(\alpha_i))$, where $F_Z^i(z)$ is the cumulated distribution function of Z_i, then β_i is the probability that a path will be truncated at the i-th step of the searching tree. By using this sequential truncation method, the searching space would then be approximately reduced to $\prod_{i=1}^{L_k} (1 - \beta_i)$, which is a small portion of the original searching space generated by a full path searching algorithm. Therefore the efficiency of parsing is increased. Since β_i in Fig. 4a is less than that in Fig. 4b, which correspond to the situation that has a large expectation difference (E[Y$_i$]-E[Z$_i$]) and a small variance ratio (Var[Y$_i$]/Var[Z$_i$]), the underlying grammar that has the property of Fig. 4b would benefit most from this algorithm. In addition, we can see that if we increase the running threshold C(α_i), we will get a greater β_i and a lower α_i. The parsing efficiency will thus increase, but the probability (i.e. $\prod_{i=1}^{L_k} \alpha_i$) that we will get the desired parse tree would decrease. How to select a good C(α_i) to achieve a desired parsing success rate would be discussed in the following section.

How to set the running threshold

Using the model given in the last section, the probability that we will get the global optimal solution, i.e. the parse tree with the largest probability, for a sentence with length L is $K_L = \prod_{i=1}^{L} \alpha_i$, where K$_L$ is a constant pre-selected by the system designer as a compromise between the parsing time and the post-editing time. Assuming that the average branching factor for each path at each stage is a constant N, then the average total number of paths we have to try is :

$$\begin{aligned} g(\alpha_1 ... \alpha_L) &= N + N*(1-\beta_1)*N + N*(1-\beta_1)*N*(1-\beta_2)*N + ... \\ &= N*(1 + N*h(\alpha_1) + N^2*h(\alpha_1)*h(\alpha_2) + ...) \\ &= N*\left(1 + \sum_{i=1}^{L-1} N^i \ * \prod_{j=1}^{i} h(\alpha_j)\right) \end{aligned} \tag{6}$$

In Eq. 6, in order to minimize the path number, the relation $h(\alpha_1) < h(\alpha_2) \ ... < h(\alpha_L)$ must holds because h(α_i) has a larger coefficient than h(α_{i+1}).

The problem of selecting an appropriate running threshold C(α_i) is now converted into one of minimizing g(α_1...α_L) under the constraint of $\prod_{i=1}^{L} \alpha_i = K_L$. Taking the logarithm on both sides, we get $\sum_{i=1}^{L} log\ \alpha_i \ = \ log\ K_L$. Then the Lagrange multiplier λ is used to get $g^*(\alpha_1 ... \alpha_L) = g(\alpha_1 ... \alpha_L) + \lambda * \sum_{i=1}^{L} log\ \alpha_i$. Taking the partial derivative of g* with respects to α_1...α_L, we will get the following equations :

$$\frac{\partial g^*}{\partial \alpha_1} = 0, \quad \frac{\partial g^*}{\partial \alpha_2} = 0, \quad ... \quad \frac{\partial g^*}{\partial \alpha_L} = 0, \quad and \quad \sum_{i=1}^{L} log\ \alpha_i \ = \ log\ K_L \tag{7}$$

There are (L+1) variables, which are α_1...α_L, and λ, and (L+1) equations. So, α_1...α_L can be solved by the numerical method. Since α_i is usually very close to 1, we can linearize the function h(α_i) in the region around α_i=1 and approximate by $h(\alpha_i) \approx a*\alpha_i + b$. In this way, we can substitute h(α_i) in the above equation by $a*\alpha_i + b$ to simplify the calculation.

During our derivation, we have assumed that the average branching factor at each stage is a constant N. This constraint can be relaxed by assuming the average branching factor at i-th stage to be N$_i$. In this way, we will get a more complicated expression for g(α_1...α_L), but it can still be solved in the same way.

The running threshold C(α_i) can now be computed off-line by selecting different K$_L$ for different sentence length L. We will call this set of C(α_i) the "static running threshold", because once they are computed, they will not be changed during the sentence parsing. However, if we arrive at a complete parse tree with much higher final accumulated running score than the final accumulated running threshold, then even if a path can pass all the accumulated running thresholds it might still be discarded when it is being compared with the final accumulated running score. So, the running threshold should be adjusted to reflect a high final accumulated running score. Therefore, it would be better if the running threshold is changed to $C'(\alpha_i)=C(\alpha_i)+\Delta C(\alpha_i)$, where $\Delta C(\alpha_i)$ is set to $\gamma * (y_i^* - C(\alpha_i))$, where $0<\gamma<1$ and y_i^* is the accumulated logarithmic score of the current best parse tree at the i-th step, and γ is a tunning constant pre-selected by the system designer. $C'(\alpha_i)$ is then the "dynamic running threshold". Using the dynamic running threshold, the efficiency of parsing would be further improved.

If it so happen that all the pathes are blocked before any complete parse tree is formed, we can find the deepest path (let us assuming it to be at the j-th step) among the blocked ones and continue it with a lowered running threshold of $C'(\alpha_j)=y'_j$, where y'_j is the score of this path at the j-th step. Since the procedure to lower the running threshold is quite complicated and uses up memory space in run time, it might be better just invoke the fail-soft mechanism for sentences whose paths are all blocked.

Testing

We completed two preliminary testings of truncation algorithm with special versions of our English-Chinese MT system and a database of 1430 sentences.

In the first experiment, the sentence parsing time needed by a charted parser that uses bottom-up parsing with top-down filtering is compared with the time needed by the same charted parser with truncation mechanism. From the test, we found that the average sentence parsing time by the charted parser with truncation is improved by a factor of four. For some sentences, the improvement can go as high as a factor of twenty. This result is encouraging because minimizing parsing time is critical to a practical MT system.

Nevertheless, we noted that our output quality has degraded slightly. By this, we mean that the best selected tree produced by the charted parser with no truncation is not among the trees produced by the charted parser with truncation. Exploring this problem further, we discovered that the chart [Wino 83] used during parsing is in conflict with the truncation mechanism. The reason for having chart is to be able to store all subtrees that were parsed in previous path traversal. So, when we backtrack to the next path and arrive at the same range of inputs, the same subtrees can be used again without reparsing. However, the idea behind the truncation mechanism is to discard subtree in the context in which it has low probability. Therefore, if we adopt the truncation mechanism during parsing, not every subtree between a string of inputs is successfully constructed and stored into the chart. For example, in Fig. 5, there are two possible subtrees between b and c when the pathes in the block A are expanded.

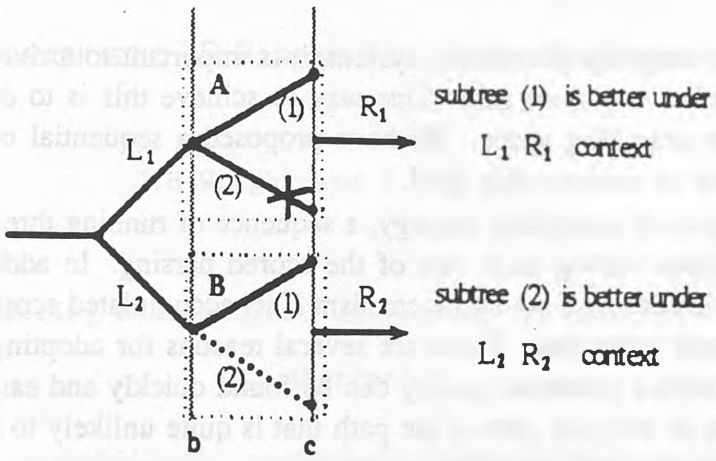


Fig. 5. Chart with truncation mechanism

In Fig. 5, one of the subtrees is discarded and the other is stored into the chart. There are two reasons why a subtree may be discarded. First, it might be caused by a natural language's constraints on the context dependency. Second, a subtree might be discarded because of its small running accumulated score (and thus truncated by the truncation mechanism.) Either will leave us a chart with incomplete subchart. So, this will result in the best possible tree being missing as a side-effect of using this chart. For instance, in Fig. 5, the best tree might be the second subtree with the left context of L_2 and with the right context of R_2 (i.e., its probability is the highest.) But, since the path expansion starting from the left context of L_1 has the second subtree discarded because its probability under the context of L_1 and R_1 is small, the best tree will never be formed. Therefore, with a chart having incomplete subcharts, the possibility of obtaining the best tree is determined by the pathes traversed before.

One solution to this incompatibility problem is to mark the sections of the chart that are complete. Hence, if an incomplete subchart is encountered again, it will be reparsed. On the other hand, if a complete set of chart is encountered, the subtrees can be copied directly from the chart. Another solution is to suspend the truncation mechanism when a set is being tried the first time. And if subtrees are copied directly from the chart, the truncation mechanism resumes its normal function. In this way, it is guaranteed that every subchart in the chart is complete. Both of these solutions increase our sentence parsing time as the overhead. This compromise, however, is unavoidable if the advantages of using chart are to be maintained.

In the second experiment, we converted the charted parser for the first experiment into one with sequential searching strategy and without the use of the chart. Similar sentence parsing test is conducted for this chartless parser but with a smaller analyses grammar. The result shows that the total parsing time for this parser with truncation mechanism added is better than the same parser without truncation by the factor of three.

From the positive results of the above two experiments, we have shown the inclusion of the sequential truncation algorithm is advantageous for a MT system. In addition, we have also shown the feasibility of harmonize the use of chart and the truncation algorithm. Currently, we are in the process of resolving the incompatibility problem between the chart and the truncation mechanism and constructing a working system with this solution.

Conclusion

In a natural language processing system, it is important to arrive at a good analysis for a sentence in a relatively short time. One way to achieve this is to decrease the parsing time by reducing the searching space. We have proposed a sequential truncation algorithm with a score function to achieve this goal.

In this sequential truncation strategy, a sequence of running thresholds are used to bound the searching space during each step of the scored parsing. In addition, a path can also be blocked by the branch-and-bound mechanism if its accumulated score is lower than that of an already completed parse tree. There are several reasons for adopting this strategy. First, the first parse tree with a moderate quality can be found quickly and easily. Second, the running threshold serves to truncate part of the path that is quite unlikely to lead to the best analysis, and thus greatly reduces the searching space.

We have made a pilot test on the truncation mechanism with a charted parser that adopts bottom-up parsing with top-down filtering. With a database of 1430 sentences, the result indicates an average improvement in the sentence parsing time by the factor of four (for some sentences the improvement goes as high as a factor of twenty). However, we also discovered an incompatibility problem between the use of chart and the truncation mechanism. In another pilot test we conducted on the truncation mechanism, the sentence parsing time is tested for a chartless parser that adopts sequential parsing strategy. The result shows an improvement in parsing time by a factor of three for the inclusion of the truncation mechanism. These encouraging results demonstrate a great promise for the sequential truncation strategy.

As our current research topic, we shall resolve the incompatibility problem between the chart and the truncation algorithm and include the solution into our working MT system, the ARCHTRAN.

References

[Benn 85] Bennett, W.S. and J. Slocum, "The LRC Machine Translation System," *Computational Linguistics*, vol. 11, No. 2-3, pp. 111-119, ACL, Apr.-Sep. 1985.

[Gars 87] Garside, Roger, Geoffrey Leech and Geoffrey Sampson (eds.), *The Computational Analysis of English : A Corpus-Based Approach*, Longman , New York, 1987.

[Marc 80] Marcus, M.P., *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, MA, 1980.

[Robi 82] Robinson, J.J., "DIAGRAM : A Grammar for Dialogues," *CACM*, vol. 25, No. 1, pp. 27-47, ACM, Jan. 1982.

[Su 87a] Su, K.-Y., J.-S. Chang, and H.-H. Hsu, "A Powerful Language Processing System for English-Chinese Machine Translation," *Proc. of 1987 Int. Conf. on Chinese and Oriental Language Computing*, pp.260-264, Chicago, Ill, USA, 1987.

[Su 87b] Su, K.-Y., J.-N. Wang, W.-H. Li, and J.-S. Chang, "A New Parsing Strategy in Natural Language Processing Based on the Truncation Algorithm", *Proc. of Natl. Computer Symposium (NCS)*, pp. 580-586, Taipei, Taiwan. 1987.

[Su 88a] Su, K.-Y. and J.-S.Chang, "Semantic and Syntactic Aspects of Score Function," *Proc. COLING-88*, vol. 2, pp. 642-644, 12th Int. Conf. on Comput. Linguistics, Budapest, Hungary, 22-27 Aug. 1988.

[Su 88b] Su, K.-Y., "Principles and Techniques of Natural Language Parsing : A Tutorial," *Proc. of ROCLING-I*, pp.57–61, Nantou, Taiwan. Oct. 1988.

[Wino 83] Winograd, Terry, *Language as a Cognitive Process*, Addison-Wesley, Reading, MA., USA, 1983.

[Wins 84] Winston, P.H., *Artificial Intelligence*, Addison-Wesley, Reading, MA., USA, 1984.

Probabilistic LR Parsing for Speech Recognition

J.H.Wright and E.N.Wrigley

Engineering Mathematics, University of Bristol, U.K.

Abstract

An LR parser for probabilistic context-free grammars is described. Each of the standard versions of parser generator (SLR, canonical and LALR) may be applied. A graph-structured stack permits action conflicts and allows the parser to be used with uncertain input, typical of speech recognition applications. The sentence uncertainty is measured using entropy and is significantly lower for the grammar than for a first-order Markov model.

1. INTRODUCTION

1.1 *Background*

The automatic recognition of continuous speech requires more than signal processing and pattern matching: a model of the language is needed to give structure to the utterance. At sub-word level, hidden Markov models [1] have proved of great value in pattern matching. The focus of this paper is modelling at the linguistic level. Markov models are adaptable and can handle potentially any sequence of words [2]. Being probabilistic they fit naturally into the context of uncertainty created by pattern matching. However, they do not capture the larger-scale structure of language and they do not provide an interpretation. Grammar models capture more of the structure of language, but it can be difficult to recover from an early error in syntactic analysis and there is no watertight grammar.

A systematic treatment of uncertainty is needed in this context, for the following reasons:

(1) some words and grammar rules are used more often than others;

(2) pattern matching (whether by dynamic time warping, hidden Markov modelling or multi-layer perceptron [3]) returns a degree of fit for each word tested, rather than an absolute discrimination; a number of possible sentences therefore arise;

(3) at the end of an utterance it is desirable that each of these sentences receive an overall measure of support, given all the data so that the information is used efficiently.

The type of language model which is the focus of this paper is the probabilistic context-free grammar (PCFG). This is an obvious enhancement of an ordinary CFG, the probability information initially intended to capture (1) above, but as will be seen this opens the way to satisfying (2) and (3). An LR parser [4,5] is used with an adaptation [6] which enlarges the scope to include almost any practical CFG. This adaptation also allows the LR approach to be used with uncertain input [7], and this approach enables a grammar model to interface with the speech recognition front end as naturally as does a Markov model.

1.2 *Probabilistic Context-Free Grammars*

A "probabilistic context-free grammar (PCFG)" [8-10] is a 4-tuple $<N,T,R,S>$ where N is a nonterminal vocabulary including the start symbol S, T is a terminal vocabulary, and R is a set of production-rules each of which is a pair of form $<A \rightarrow \alpha,\ p>$, with $A \in N$, $\alpha \in (N \cup T)^*$, and p a probability. The probabilities associated with all the rules having a particular nonterminal on the LHS must sum to one. A probability is associated with each derivation by multiplying the probabilities of those rules used, in keeping with the context-freeness of the grammar.

A very simple PCFG can be seen in figure 1: the symbols in uppercase are the nonterminals, those in lowercase are the terminals (actually preterminals) and λ denotes the null string.

2. LR PARSING FOR PROBABILISTIC CFGs

The LR parsing strategy can be applied to a PCFG if the rule-probabilities are driven down into the parsing action table by the parser generator. In addition, one of the objectives of using the parser in speech recognition is for providing a set of prior probabilities for possible next words at successive stages in the recognition of a sentence. The use of these prior probabilities will be described in section 3.1. In what follows it will be assumed that the grammars are non-left-recursive, although null rules are allowed.

2.1 *SLR Parser*

The first aspect of parser construction is the closure function. Suppose that I is an SLR kernel set consisting of LR(0) items of the form

$$<A \rightarrow \alpha \cdot \beta,\ p>$$

The item probability p can be thought of as a posterior probability of the item given the terminal string up to that point. The computation of closure(I) requires that items

$$<B \rightarrow \cdot \gamma_r,\ p_B p_r>$$

be added to the set for each rule $<B \rightarrow \gamma_r,\ p_r>$ with B on the LHS, provided $p_B p_r$ exceeds some small probability threshold ϵ, where p_B is the total probability of items with B appearing after the dot (in the closed set).

New kernel sets are generated from a closed set of items by the goto function. If all the items with symbol $X \in (N \cup T)$ after the dot in a set I are

$$<A_k \rightarrow \alpha_k \cdot X \beta_k,\ p_k> \quad \text{for } k=1,\ldots,n_X, \quad \text{with } p_X = \sum_{k=1}^{n_X} p_k$$

then the new kernel set corresponding to X is

$$\{<A_k \rightarrow \alpha_k X \cdot \beta_k,\ p_k/p_X> \quad \text{for } k=1,\ldots,n_X\}$$

and goto(I,X) is the closure of this set. The set already exists if there is another set which has the same number of elements, an exact counterpart for each dotted item, and a probability for each item that differs from that for its counterpart in the new set by at most ϵ.

Starting from an initial state I_0 consisting of the closure of

$\{<S' \rightarrow \cdot S,\ 1>\}$

where S' is an auxiliary start symbol, this process continues until no further sets are created. They can then be listed as $I_0, I_1, \ldots$.

Each state set I_m generates state m and a row in the parsing tables "action" and "goto". The goto table simply contains the numbers of the destination states, as for the deterministic LR algorithm, but the action table also inherits probabilistic information from the grammar.

(1) For each terminal symbol b, if there are items in I_m such that the total $p_b > \epsilon$, and the shift state n is given by $goto(I_m, b) = I_n$, then

$action[m,b] = <shift\text{-}to\text{-}n,\ p_b>$

(2) For each nonterminal symbol B, if $p_B > \epsilon$ and $goto(I_m, B) = I_n$ then

$goto[m,B] = n$

(3) If $<S' \rightarrow S\cdot,\ p> \in I_m$ then $action[m,\$] = <accept,\ p>$

(4) If $<B \rightarrow \gamma\cdot,\ p> \in I_m$ where $B \neq S'$ then

$action[m, FOLLOW(B)] = <reduce\text{-}by\ B \rightarrow \gamma,\ p>$

For the very simple grammar shown in figure 1 the parsing tables turn out as shown in figure 2, with shift-reduce optimisation [4,5] applied. The probability of each entry is underneath.

The range of terminal symbols which can follow a B-reduction is given by the set FOLLOW(B) which can be obtained from the grammar by a standard algorithm [4]. For a probabilistic grammar, the probability p attached to the reduce item cannot be distributed over those entries because when the tables are compiled it is not determined which of those terminals can actually occur next in that context, so the probability p is attached to the whole range of entries.

The probability associated with a shift action is the prior probability of that terminal occurring next at that point in the input string (assuming no conflicts). Completing the set of prior probabilities involves following up each reduce action using local copies of the stack until shift actions block all further progress. The reduce action probability must be distributed over the shift terminals which emerge. This is done by allocating this probability to the entries in the action table row for the state reached after the reduction, in proportion to the probability of each entry. Some of these entries may be further reduce actions in which case a similar procedure must be followed, and so on.

2.2 *Canonical LR Parser*

For the canonical LR parser each item possesses a lookahead distribution:

$<A \rightarrow \alpha\cdot\beta,\ p,\ \{P(a_i)\}_{i=1,\ldots,|T|}>$

The closure operation is more complex than for the SLR parser, because of the propagation of lookaheads through the non-kernel items. The items to be added to a kernel set to close it take the form

$$\langle B \rightarrow \cdot\gamma_r,\ p_B p_r,\ \{P_B(a_j)\}_{j=1,\ldots,|T|}\rangle$$

so that all the items with B after the dot are then

$$\langle A_k \rightarrow \alpha_k \cdot B\beta_k,\ p_k,\ \{P_k(a_i)\}_{i=1,\ldots,|T|}\rangle \quad \text{for } k=1,\ldots,n_B$$

and

$$P_B(a_j) = \sum_{k=1}^{n_B} \frac{p_k}{p_B} \sum_{i=1}^{|T|} P^F(\beta_k a_i, a_j) P_k(a_i)$$

where $P^F(\beta_k a_i, a_j)$ is the probability of a_j occurring first in a string derived from $\beta_k a_i$, which is easily evaluated. A justification of this will be published elsewhere. The lookahead distribution is copied to the new kernel set by the goto function.

The first three steps of parsing table construction are essentially the same as for the SLR parser. In step (4), the item in I_m takes the form

$$\langle B \rightarrow \gamma\cdot,\ p,\ \{P(a_i)\}_{i=1,\ldots,|T|}\rangle \quad \text{where } B \neq S'$$

The total probability p has to be distributed over the possible next input symbols a_i, using the lookahead distribution:

$$\text{action}[m, a_i] = \langle\text{reduce-by } B \rightarrow \gamma,\ pP(a_i)\rangle$$

for all i such that $pP(a_i) > \epsilon$. The prior probabilities during parsing action can now be read directly from the action table.

2.3 *LALR Parser*

Merging the states of the canonical parser which differ only in lookaheads for each item causes the probability distribution of lookaheads to be lost, so for the LALR parser the LR(1) items take the form

$$\langle A \rightarrow \alpha\cdot\beta,\ p,\ L\rangle \quad \text{where } L \subseteq T.$$

The preferred method for generating the states as described in [4] can be adapted to the probabilistic case. Reduce entries in the parsing tables are then controlled by the lookahead sets, with the prior probabilities found as for the SLR parser.

2.4 *Conflicts and Interpretation*

An action conflict arises whenever the parser generator attempts to put two (or more) different entries into the same place in the action table, and there are two ways to deal with them. The first approach is to resolve each conflict [11]. This is a dubious practice in the probabilistic case because there is no clear basis for resolving the probabilities of the actions in conflict. The second approach is to split the stack and pursue all options, conceptually in parallel. Tomita [6] has devised an efficient enhancement of the LR parser which operates in this way. A graph-structured stack avoids duplication of effort and maintains (so far as possible) the speed and compactness of the parser. With this approach the LR algorithm can handle almost any practical CFG, and is highly suited to probabilistic grammars, the main distinction being that a probability becomes attached to each branch.

The generation and action of the probabilistic LR parser can be supported by a Bayesian interpretation. This is in keeping with the further adaptation of the algorithm to deal with uncertain input.

3. UNCERTAIN INPUT DATA

3.1 *Prediction and Updating Algorithm*

The situation envisaged for applications of the probabilistic LR parser in speech recognition is depicted in figure 3. The parser forms part of a linguistic analyser whose purpose is to maintain and extend those partial sentences which are compatible with the input so far. With each partial sentence there is associated an overall probability and partial sentences with very low probability are suspended. It is assumed that the pattern matcher returns likelihoods of words, which is true if hidden Markov models are used. Other methods of pattern matching return measures which it is assumed can be interpreted as likelihoods, perhaps via a transformation.

let Γ_s^m (s=1,2,...) represent partial sentences up to stage m (the stage denoted by a superscript). let D^m represent the data at stage m, and $\{D\}^m$ represent all the data up to stage m. Each branch Γ_s^m predicts words a_j^m (perhaps via the LR parser) with probability $P(a_j^m|\Gamma_s^{m-1})$, so the total prior probability for each word a_j^m is

$$P(a_j^m|\{D\}^{m-1}) = \sum_s P(a_j^m|\Gamma_s^{m-1})P(\Gamma_s^{m-1}|\{D\}^{m-1})$$

Using Bayes' theorem the posterior probabilities of the words are

$$P(a_j^m|\{D\}^m) = \frac{P(D^m|a_j^m)P(a_j^m|\{D\}^{m-1})}{\sum_i P(D^m|a_i^m)P(a_i^m|\{D\}^{m-1})}$$

where $P(D^m|a_j^m)$ is the likelihood. If we define the extended branch Γ_{sj}^m as $\Gamma_s^{m-1}\&a_j^m$ then after some manipulation the probability of this is

$$P(\Gamma_{sj}^m|\{D\}^m) = \frac{P(a_j^m|\Gamma_s^{m-1})P(\Gamma_s^{m-1}|\{D\}^{m-1})}{P(a_j^m|\{D\}^{m-1})} P(a_j^m|\{D\}^m) \qquad (1)$$

This shows that the posterior probability of a_j^m is distributed over the extended partial sentences in proportion to their root sentences's contribution to the total prior probability of that word. If $P(\Gamma_{sj}^m|\{D\}^m)<\epsilon$ then the branch is suspended. The next set of prior probabilities can now be derived and the cycle continues.

These results are derived using the following independence assumptions:

$$P(a_i^k|a_j^m,D^m) = P(a_i^k|a_j^m) \quad \text{and} \quad P(D^m|a_j^m,D^k) = P(D^m|a_j^m)$$

which decouple the data at different stages.

Figure 4 shows successive likelihoods, entered by hand for a (rather contrived) illustration using the grammar in figure 1. At the end the two viable sentences (with probabilities) are

"pn tv det n pron tv pn" (0.897)
"det n pron tv pn tv pn" (0.103)

Notice that the string which maximises the likelihood at each stage,

"pn tv pron tv pron tv pn"

might correspond to a line of poetry but is not a sentence in the language.

The graph-structured stack approach of Tomita [6] is used for non-deterministic input. Each path through the stack graph corresponds to one or more partial sentences and the probability $P(\Gamma_s^m | \{D\}^m)$ has to be associated with each partial sentence Γ_s^m.

3.2 *Entropy of the Partial Sentences*

Despite the pruning the number of partial sentences maintained by the parser tends to grow with the length of input. It seems sensible to base the measure of complexity upon the probabilities of the sentences rather than their number, and the obvious measure is the entropy of the distribution. The discussion here will assume that the proliferation of sentences is caused by input uncertainty rather than by action conflicts. This is likely to be the dominant factor in speech applications.

The sentence entropy $\mathcal{H}_S^m$ is defined as

$$\mathcal{H}_S^m = - \sum_{s,j} P(\Gamma_{sj}^m | \{D\}^m) \log P(\Gamma_{sj}^m | \{D\}^m)$$

where natural logarithms are used. A related measure called "perplexity" [12], defined as

$$\mathcal{P}_S^m = \exp(\mathcal{H}_S^m)$$

is the equivalent (in entropy) number of equally-likely sentences. Substituting for $P(\Gamma_{sj}^m | \{D\}^m)$ from equation (1) leads to

$$\mathcal{H}_S^m = - \sum_j P(a_j^m | \{D\}^m)[\log P(a_j^m | \{D\}^m) - h_j^m]$$

where

$$h_j^m = - \sum_s P(\Gamma_s^{m-1} | a_j^m, \{D\}^{m-1}) \log P(\Gamma_s^{m-1} | a_j^m, \{D\}^{m-1})$$

is the entropy contributed by the sentences at stage m-1 predicting word a_j^m. The quantities h_j^m can be evaluated with the prior probabilities.

It can be shown that the sentence entropy has an upper bound as a function of the likelihoods:

$$\mathcal{H}_S^m \leqslant \log \sum_j \exp(h_j^m)$$

withequality when $P(D^m | a_j^m) \propto \dfrac{\exp(h_j^m)}{P(a_j^m | \{D\}^{m-1})}$.

The constant of proportionality does not matter. Figure 5(a) shows this upper bound for the grammar in figure 1, and it can be seen that the perplexity is equivalent to 35 equally-likely sentences after 10 words.

The upper bound is very pessimistic because it ignores the discriminative power of the pattern matcher. This could be measured in various ways but it is convenient to define a "likelihood entropy" $\mathcal{H}_L^m$ as

$$\mathcal{H}_L^m = -\sum_j \frac{P(D^m|a_j^m)}{\sum_i P(D^m|a_i^m)} \log \frac{P(D^m|a_j^m)}{\sum_i P(D^m|a_i^m)}$$

and the "likelihood perplexity" is then $\mathcal{P}_L^m = \exp(\mathcal{H}_L^m)$.

The maximum sentence entropy subject to a fixed likelihood entropy can be found by simulation. Sets of random likelihoods with a given entropy can be generated from sets of independent uniform random numbers by raising these to an appropriate power. Permuting these so as to maximise the sentence entropy greatly reduces the number of sample runs needed to get a good result. These likelihoods are then fed into the parser and the procedure repeated to simulate the recognition process. The sentence entropy is maximised over a number of such runs.

The likelihoods which produce the upper bound line shown in figure 5(a) have a perplexity which is approximately constant at 6.6. This line is reproduced almost exactly by the above simulation procedure, using a fixed $\mathcal{P}_L$ of 6.6 with 30 sample runs.

The simulation method is easily adapted to compute the average sentence entropy over the sample runs. For this it is preferable to average the entropy and then convert to a perplexity rather than average the measured perplexity values. This process provides an indication of how the parser will perform in a typical case, assuming a fixed likelihood perplexity as a parameter (although this could be varied from stage to stage if required).

Figure 5(a) shows how the average compares with the maximum for a fixed $\mathcal{P}_L$ of 6.6, and how the sentence perplexity is reduced when the likelihoods are progressively more constrained ($\mathcal{P}_L = 5.0$, 3.0 and 2.0).

3.3 *Comparison with Inferred Markov Model*

Markov models have some advantages over grammar models for speech recognition in flexibility and ease of use but a major disadvantage is their limited memory of past events. For an extended utterance the number of possible sentences compatible with a Markov model may be much greater than for a grammar model, for the same data. Demonstrating this in the present context requires the derivation of a first-order Markov model from a probabilistic grammar [13].

The uncertainty algorithm of section 3.1 will operate largely unchanged with the prior probabilities obtained from the transition probabilities rather than from the LR parser. Figure 5(b) contains results corresponding to those in (a), for the Markov model inferred from the grammar in figure 1. The upper bound reaches 409 after 10 words, for a likelihood perplexity of approximately 6.3, reducing to 37 for the average (after 30 sample runs). This falls with the likelihood perplexity but is higher than for the grammar model. The sentence perplexity for the grammar is twice that for the inferred Markov model after from six to nine words depending on $\mathcal{P}_L$. This comparison is reproduced for other grammars considered.

References

1. S E Levinson, L R Rabiner and M M Sondhi, "An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition", BSTJ vol 62, pp1035-1074, 1983.

2. R Garside, G Leech and G Sampson (eds), "The Computational Analysis of English, a Corpus-Based Approach", Longman, 1987.

3. H Bourland and C J Wellekens, "Speech Pattern Discrimination and Multilayer Perceptrons", Computer Speech and Language, vol 3, pp1-19, 1989.

4. A V Aho, R Sethi and J D Ullman, "Compilers: Principles, Techniques and Tools", Addison-Wesley, 1985.

5. N P Chapman, "LR Parsing, Theory and Practice", Cambridge University Press, 1987.

6. M Tomita, "Efficient Parsing for Natural Language", Kluwer Academic Publishers, 1986.

7. J H Wright and E N Wrigley, "Linguistic Control in Speech Recognition", Proceedings of the 7th FASE Symposium, pp545-552, 1988.

8. P Suppes, "Probabilistic Grammars for Natural Languages", Synthese, vol 22, pp95-116, 1968.

9. W J M Levelt, "Formal Grammars in Linguistics and Psycholinguistics, volume 1", Mouton, 1974.

10. C S Wetherall, "Probabilistic Languages: A Review and Some Open Questions", Computing Surveys vol 12, pp361-379, 1980.

11. S M Shieber, "Sentence Disambiguation by a Shift-Reduce Parsing Technique", Proc. 21st Annual Meeting of Assoc. for Comp. Linguistics, pp113-118, 1983.

12. L R Bahl, J Jelinek and R L Mercer, "A Maximum Likelihood Approach to Continuous Speech Recognition", IEEE Trans. on Pattern Analysis and Machine Intelligence, vol PAMI-5, pp179-190, 1983.

13. J H Wright, "Linguistic Modelling for Application in Speech Recognition", Proceedings of the 7th FASE Symposium, pp391-398, 1988.

(1) S $\rightarrow$ NP VP, 1.0	(5) REL $\rightarrow$ pron VP, 0.3
(2) NP $\rightarrow$ pn, 0.4	(6) VP $\rightarrow$ iv, 0.5
(3) NP $\rightarrow$ det n REL, 0.6	(7) VP $\rightarrow$ tv NP, 0.5
(4) REL $\rightarrow$ λ, 0.7	

Figure 1: A simple probabilistic grammar.

STATE	ACTION							GOTO			
	pn	det	n	pron	iv	tv	$	S	NP	REL	VP
0	sr2 0.4	s1 0.6						s2	s3		
1			s4 1.0								
2							acc 1.0				
3					sr6 0.5	s5 0.5					sr1
4				s6 0.3	r4 ⟵	r4 0.7	r4 ⟶			sr3	
5	sr2 0.4	s1 0.6							sr7		
6					sr6 0.5	s5 0.5					sr5

Figure 2: SLR and LALR parsing tables for the grammar in figure 1.

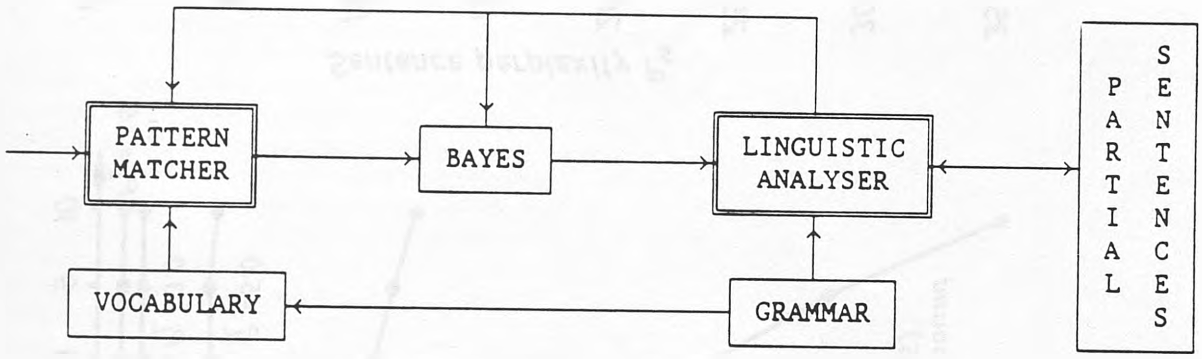


Figure 3: Linguistic control block diagram for speech recognition.

TERMINAL	STAGE (m)							
	1	2	3	4	5	6	7	8
pn	0.9			0.3	0.4		0.9	
det	0.2		0.4					
n		0.2		0.5				
pron			0.8		0.7			
iv								
tv		0.8	0.1	0.9		0.8		
$								1.0

Figure 4: Likelihoods for illustration of uncertainty algorithm.

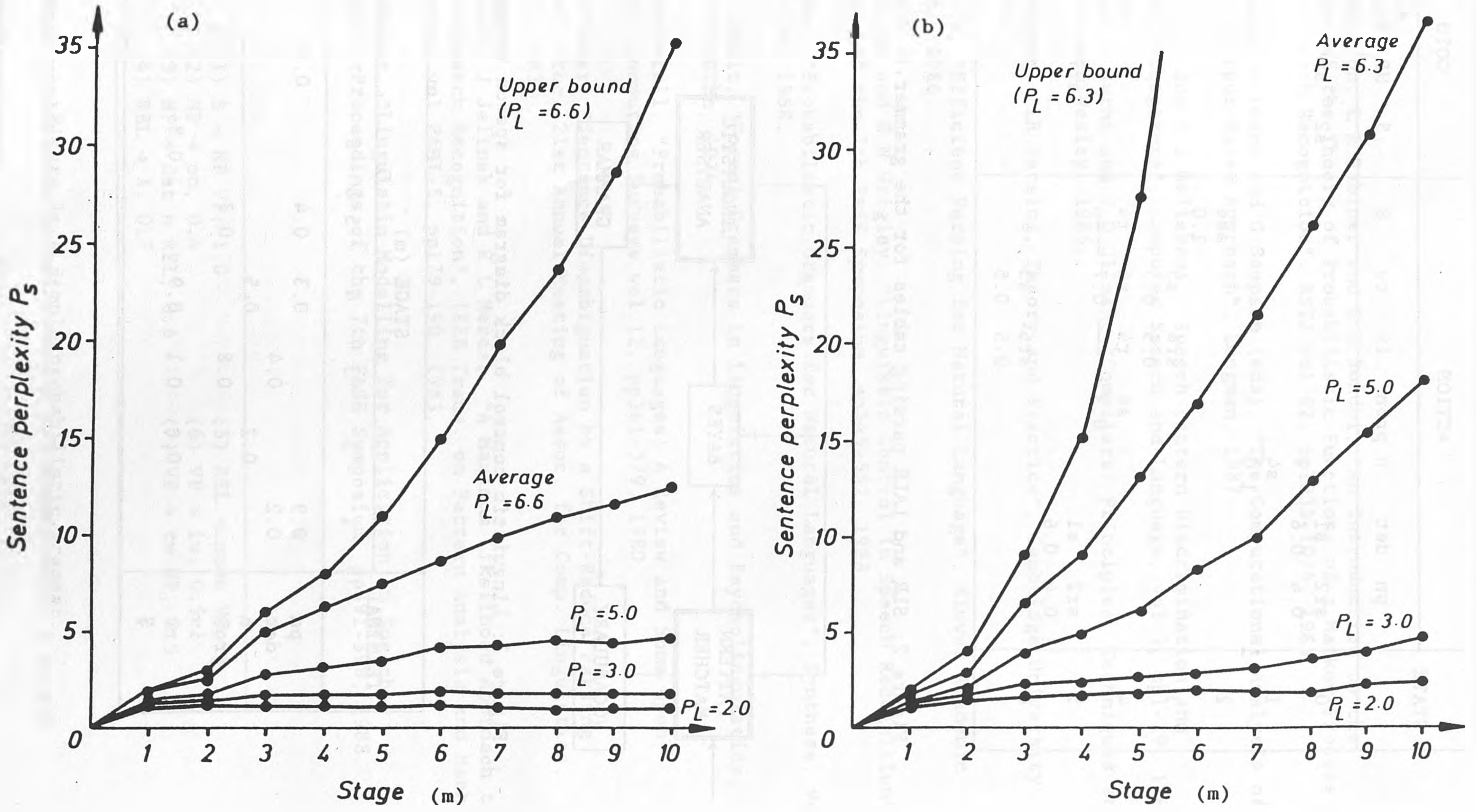


Figure 5: Sentence perplexities for (a) grammar, (b) Markov model.

PARSING SPEECH FOR STRUCTURE AND PROMINENCE

Dieter Huber

Department of Computational Linguistics
University of Göteborg
and
Department of Information Theory
Chalmers University of Technology
S-412 96 Gothenburg
Sweden

International Workshop on Parsing Technologies
Carnegie Mellon University
Pittsburgh, Pennsylvania
August 28-31, 1989

INTRODUCTION

The purpose of parsing natural language is essentially to assign to a linear input string of symbols a formalized structural description that reflects the underlying linguistic (syntactic and/or semantic) properties of the utterance and can be used for further information processing.

In most practical applications, this *delinearization* [4] is acheived by some kind of recursive pattern matching strategy which accepts texts in standard orthographic writing, i.e. composed of discrete symbols (the letters and signs of some specified alphabet) and blocks of symbols (words separated by blanks) as input, and rewrites them step by step, in accordance with (1) a lexicon and (2) a finite set of production rules defined in a formal grammar, into a *parse tree* or a *bracketed string*. This approach is commonly restricted to the domain of the sentence as maximal unit of linguistic processing, thus adhering to the traditional view that larger units like paragraphs, texts and discourse, are formed by mere juxtaposition of autarchic, independently parsed sentences.

Clearly, this kind of procedure developed for parsing written language material is not immediately applicable to speech processing purposes. For one, natural human speech does not normally present itself in the acoustical medium as a simple linear string of discrete, well demarcated and easily identifiable symbols, but constitutes a continuously varying signal which incorporates virtually unlimited allophonic variations, reductions, elisions, repairs, overlapping segmental representations, grammatical deficiencies, and potential ambiguities at all levels of linguistic description. There are no "blanks" and "punctuation marks" to define words or indicate sentential boundaries in the acoustic domain. Syntactic structures at least in spontaneous speech are often fragmentary or highly irregular, and cannot be easily defined in terms of established grammatical theory [26]. Last not least, important components of the total message are typically encoded and transmitted by nonverbal and even nonvocal means of communication [18].

On the other hand, human speakers organize and present their speech output in terms of well defined and clearly delimited *chunks* rather than as an unstructured, amorphous chain of signals. This division into chunks is represented among other parameters in the time course of voice fundamental frequency (F_0) where it appears as a sequence of coherent *intonation units* optionally delimited by pauses and/or periods of laryngealization [19], and containing at least one salient pitch movement [9],[20]. Human listeners are able to perceive these units as "natural groups" forming a kind of *performance structure* [12], which reflects the *information structure* of the utterance [14] and is used to decode the intended meaning of the transmitted message. This involves (1) chopping up the message into *information units* in accordance with the speaker's and listener's shared state of knowledge, (2) organizing these units both internally and externally in terms of given and new information, and (3) selecting one or at the most two elements in each unit as points of prominence within the message.

SYSTEM OVERVIEW

While written language input is generally presented to the parser with both the *terminal symbols* (i.e. words) and the *starting symbols* or *roots* (i.e. sentences) clearly delineated and set off from each other by spaces and/or punctuation marks, thus imposing the parsing algorithm with the task to identify som kind of intermediate structure(s) representation composed of *variables* from a finite set of *non-terminal symbols* or *categories* (i.e. the phrase structure, constituent structure, functional structure, etc), essentially the reverse applies when parsing connected speech input. That is, the continuously varying speech signal is presented to the analysis with some kind of intermediate structure(s) representation either immediately observable (e.g. the voiced-unvoiced distinction between individual speech sounds) or readily deducible (e.g. the prosodic structure expressed in patterns of intonation and accentuation) without prior knowledge of higher-level linguistic information, thus leaving the parser with the task to recognize (or rather support the recognition of) both the individual words and the full sentences.

This reverse relationship between text parsing on one side and speech parsing on the other is illustrated schematically in figure 1. It must be appreciated in this context that the intermediate structure(s) representations in text *versus* speech parsing are neither identical nor necessarily isomorphical.

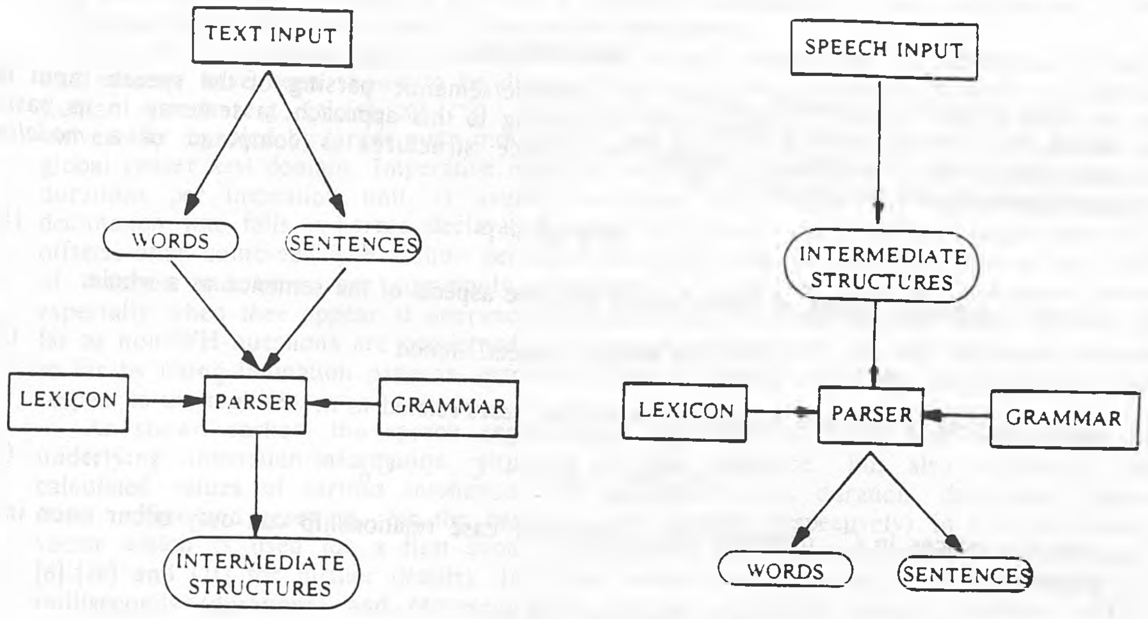


Figure 1 Parsing NL text *versus* parsing connected speech

The speech parsing algorithm presented in this study is thus initiated by a data-driven, left-right speech segmentation stage that exploits the prosodically cued *chunking* present in the acoustical speech signal and uses it to perform automatic, speaker-independent segmentation of continuous speech into functionally defined intonation/information units. For this purpose, two global declination lines are computed by the *linear regression* method, which approximate the trends in time of the peaks (topline) and valleys (baseline) of F_0 across the utterance. Computation is reiterated every time the *Pearson Product Moment Correlation Coefficient* drops below a preset level of acceptability. Segmentation is thus performed without prior knowledge of higher-level linguistic information, with the termination of one unit being determined by the general resetting of the intonation contour wherever in the utterance it may occur.

Earlier studies in the correlations between prosody and grammar have shown that the intonation units thus established time-align in a clearly defined way with units of linguistic structure that can be described in probabilistic terms with respect to three interlacing levels of analysis: constituent structure, linear word count and duration [1],[20]. Furthermore, once the extent of an intonation unit has been established both in the time and in the frequency domain, areas of prominence can easily be detected as overshooting or undershooting F_0 excursions that provide valuable points of departure for further linguistic analysis and island parsing strategies.

A detailed description of the segmentation algorithm together with an evaluation of its performance on three medium sized Swedish texts read by four native speakers (two female, two male) is presented in [21]. Problems of classification by means of hierarchically organized, non-parametric, multiple-hypothesis classifiers are discussed in [6]. A statistical evaluation and coarse classification of the time-alignment between the intonation units established by our segmentation algorithm and features of linguistic structure at the level of a complete sentence (S), clause (C), noun phrase (SUB), verb phrase (VP), adverbial modifier (ADV) and parenthetical construction (PAR) can be found in [20] and [21].

The present paper deals specifically with design aspects of a parsing algorithm that accepts the output of the speech segmentation stage as input and uses it

1 - to build a *case grammar* representation of the original speech utterance;
2 - to guide the word recognition process by generating expectations resulting from partial linguistic analyses.

In the following sections, the grammar formalism, the lexicon and the parser will be presented as separate modules. Problems of integration with other language models (linguistic and stochastic) will be discussed in the summary.

GRAMMAR

The grammar formalism adopted for syntactic/semantic parsing of the speech input is based on Fillmore's *case grammar* [11]. According to this approach, a sentence in its basic structure (deep structure as opposed to surface structure) is composed of a *modality* component M and the *proposition* P:

$$S \Rightarrow M + P \tag{1}$$

where M defines a series of modes which describe aspects of the sentence as a whole:

$$M \Rightarrow \text{tense. aspect...mood} \tag{2}$$

and P consists of the verb together with various *cases* related to it:

$$P \Rightarrow \text{Verb} + C_1 + C_2 \ldots C_n \tag{3}$$

with the indices in C_i denoting that a particular case relationship can only occur once in a proposition.

Each case is defined according to Simmons [28] as:

$$C \Rightarrow K + NP \tag{4}$$

where K (which may be null) stands for the preposition which introduces the noun phrase and defines its relationship with the verb:

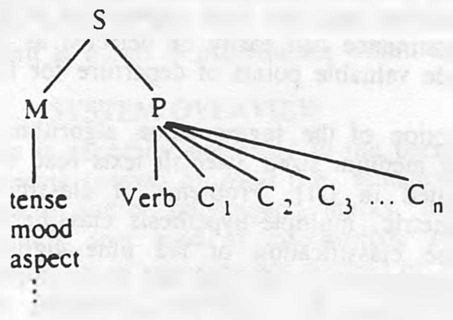
$$K \Rightarrow \text{Prep} \tag{5}$$

and the noun phrase NP is defined as:

$$NP \Rightarrow (\text{Prep})^* + (\text{Det})^* + (\text{Adj}|\text{N})^* + \text{N} + (\text{S}|\text{NP})^* \tag{6}$$

in which the parentheses denote optional elements, the asterix means that the element may be repeated, and the vertical bar indicates alternation.

A full case grammar representation can thus be described as a tree structure in the form:



(7)

Modality

Within the general framework of case grammar, the following modes and their respective possible values have been adopted:

TENSE	- present, past, future
ASPECT	- perfect, imperfect
ESSENCE	- positive, negative, indeterminate
FORM	- simple, emphatic, progressive
MODAL	- can, may, must
MOOD	- declarative, imperative, interrogative
MANNER	- adeverbial
TIME	- adverbial

The modality of the utterance as a whole is ultimately determined by the combination of the individual values assigned to each of the modes listed above.

At least five of these eight modes, i.e. form, mood, essence and the adverbials of time and manner have been shown to be directly reflected in the intonation contours of natural human speech (e.g. [2],[5],[20],[27]). For instance, emphatic pronunciation appears to be universally signaled by larger pitch movements both in the local (emphatic accent) and in the global (wider *key*) domain. Imperative mood, in addition to displaying on the average shorter durations per intonation unit, is usually associated with higher F_0 onsets and steeper declination line falls, whereas declarative mood is typically cued by low, target-value F_0 offsets, often combined with a short period of laryngealization or devoicing. Adverbials, both of manner and time, are commonly processed in terms of separate intonation units, especially when they appear at utterance-final positions. The interrogative mood, at least as far as non-WH-questions are concerned, is signaled intonationally in most languages studied so far by rising intonation patterns, terminally and/or globally (the latter predominantly with respect to the topline).

As shown earlier, the speech segmentation algorithm not only aims to unearth the underlying intonation/information structure of the utterance, but also represents the calculated values of various intonation unit parameters (i.e. duration, declination slope, onset, offset and resetting, for the baselines and toplines respectively) in a 10-parameter vector which is used for a first broad classification and hierarchization (see references [6],[20] and [21] for further details). Individual values are measured in Hz (F_0-values) or milliseconds (durations) and represented in separate probability density functions (PDF) which allows for (1) finer grain, (2) fast computation of average means, standard deviations and modal targets, and (3) direct comparison and categorization of individual intonation unit parameters reflecting *modality* by simple and robust VQ methods.

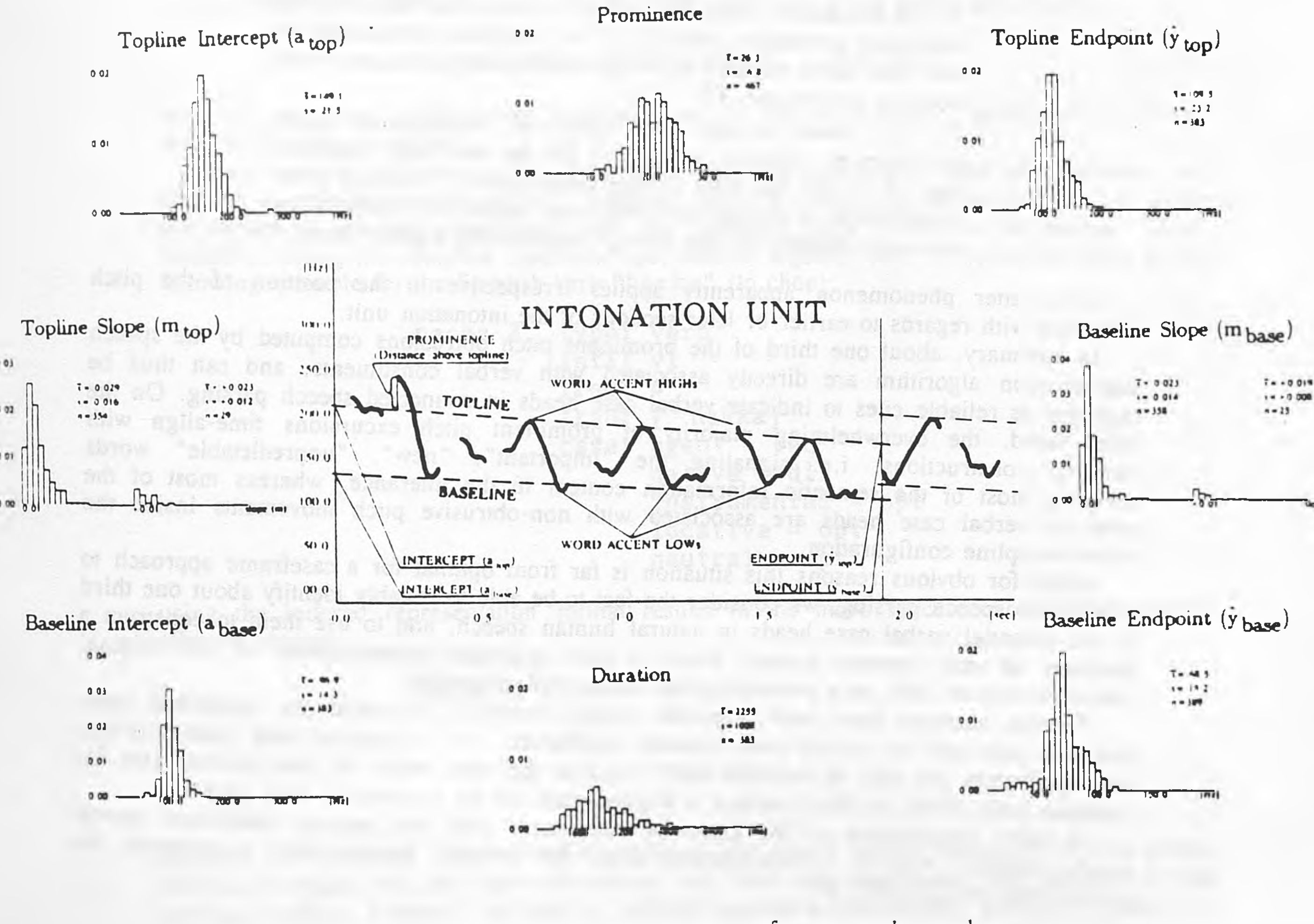


Figure 2 Intonation unit parameters for one male speaker

In summary, modality provides essential information about the propositional content of the utterance. It also provides valuable cues to word order (e.g. interrogative mood is often associated with inverted word order), word structure (e.g. imperative sentences usually lack a lexical expression for the subject, which is commonly understood to be the addressed person), and constituent identity. Determining the modality at an early stage of the parsing process by probabilistic evaluation of the intonational cues specified by the segmentation algorithm thus helps (1) to establish important aspects of the overall meaning of the utterance, and (2) to judge the plausibility of alternative word order hypotheses.

Proposition

In traditional case grammar, the main verb in the proposition constitutes the kernel to which the cases are attached, and the auxiliary verbs contain much of the information about modality. It is thus important to detect and identify the verbal elements of the utterance at an early stage of the parsing process.

It has been shown earlier that once the extent of an intonation unit is established both in the time and in the frequency domain, areas of prominence can easily be spotted as overshooting or undershooting pitch excursions that reach outside the F_0 range defined by the computed baseline-topline configuration. Unfortunately, only a small proportion of these prominent pitch obtrusions (less than one third, i.e. 31.6 %, in our accumulated Swedish material comprising 10440 running words and 704 sentences of read speech recorded by four native speakers) have been found to be directly associated with the verbal constituents in natural human speech, and thus provide an immediate cue for the detection and identification of the *case head*. On the other hand, these verb-prominence coincidences - at least in our Swedish material - have been found to be strongly related:

1 - to prominent pitch obtrusions in the *initial* parts of the individual intonation units (81.7 %), whereas prominence in the final parts appears to be predominantly associated with nominal constituents (77.1 %);

2 - to *lower* average F_0 values of overshooting pitch prominence (typically around 12 Hz for our male speakers and 17-20 Hz for their female counterparts), whereas pitch prominence in connection with focal accent or emphasis on nominals reaches on the average significantly higher values.

This latter phenomenon apparently applies irrespective to the position of the pitch obtrusion with regards to earlier or later sections of the intonation unit.

In summary, about one third of the prominent pitch obtrusions computed by the speech segmentation algorithm are directly associated with verbal constituents, and can thus be regarded as reliable cues to indicate verbal case heads in connected speech parsing. On the other hand, the overwhelming majority of prominent pitch excursions time-align with nominal constructions, i.e. signaling the "important", "new", "unpredictable" words carrying most of the semantic information content in the utterance, whereas most of the potential verbal case heads are associated with non-obtrusive pitch movements inside the baseline-topline configuration.

Albeit for obvious reasons this situation is far from optimal for a caseframe approach to continuous speech parsing, we consider the fact to be able to reliably identify about one third of the potential verbal case heads in natural human speech, and to use them to construct a skeleton of verb kernels around which a case grammar representation of the original utterance can be built, as a promising step in the right direction.

Several attempts have been reported in the literature to extend the traditional case-theoretic approach to include even nominal caseframes, i.e. to construct case grammars that use caseframes not only to describe verbs but also the head nouns of noun phrases (see for instance [15]). Work in this direction is ongoing and will be reported in later papers.

A fuller presentation of the grammar component built for parsing continuous speech input, together with an implementation study for Swedish speech input is prepared for presentation at COLING 90.

LEXICON

The lexicon to be used with the parser is specially designed for speech processing applications (text-to-speech. speech recognition. speech coding. etc) and supports the caseframe approach to continuous speech parsing outlined in this study. Its format is defined as a Swedish monolingual dictionary which contains in addition to the standard entries (head. homograph index. part-of-speech. inflexion code. morphological form classes. etc) also:

1 - a narrow phonetic transcription reflecting standard pronunciation usage:
2 - the textual frequency rating based on a one-million word korpus of Swedish newspaper articles:
3 - an indexed caseframe description for each verb entry.

For the latter purpose. the following reduced set of cases has been adopted from Stockwell. Schachter and Partee [29]. with definitions compiled by the author:

AGENT	- animate instigator of the action
DATIVE	- animate recipient of the action
INSTRUMENTAL	- inanimate object used to perform the action
LOCATIVE	- location or orientation of the action
NEUTRAL	- the thing being acted upon (combining the objective and the factive in Fillmore's original list of cases

A caseframe is thus defined as an ordered array composed of the entire set of cases

```
caseframe = array[agent...neutral]                (8)
```

in which each case can be either required (`req`) or optional (`opt`) or disallowed (`dis`) and must be marked accordingly.

Since several different verbs often share the same particular kind of caseframe. we propose to store the entire set of 3^5 logically possible caseframes as an indexed list. using the indices as pointers (identifiers) with the respective verb entries in the lexicon. Thus. instead of listing the complete caseframe specification together with the lexical entry as in the following example for the Swedish verb "**hacka**" (to chop):

```
hacka 3   type: verb
          infl: v1
          freq: 4
          tran: [ ²hakka]
          case: agent - req
                dative - dis
                instrumental - opt
                locative - opt
                neutral - opt
```

using the indexed representation format results in the more space-economic and search-effective structure:

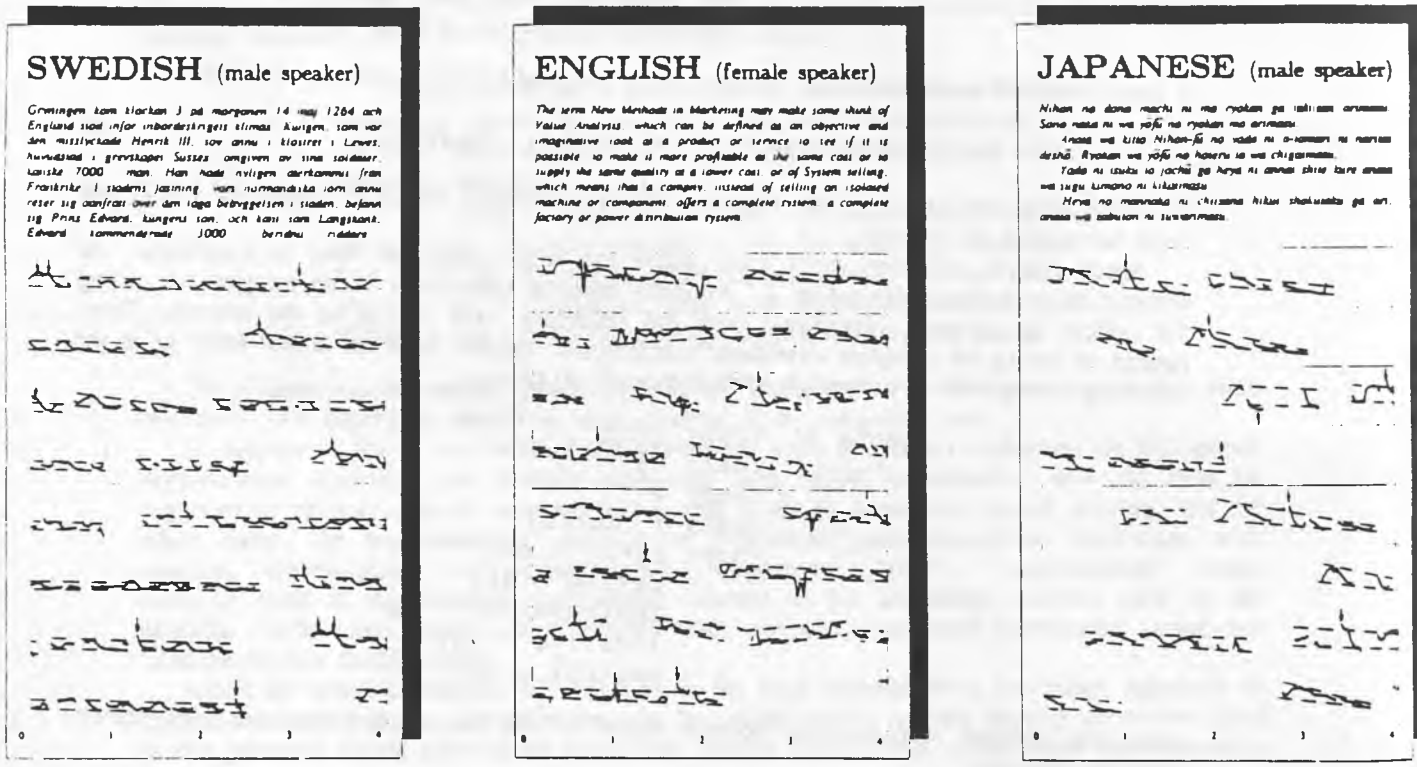
```
hacka 3   type: verb
          infl: v1
          freq: 4
          tran: [ ²hakka]
          case: 97
```

Observe that the entry "`type: verb`" might at first glance appear redundant in view of the fact that to begin with only the verb entries are listed with caseframes. As indicated in the previous section. however. we plan to include caseframe descriptions even for nouns and other nominal constructions, with feature descriptions based on research on valency theory currently conducted at the department of computational linguistics. Further lexical work is also directed towards the extension of individual case states marked as "`req`" or "`opt`" with probabilistic lexical hypotheses derived from KWIC-studies of coherent speech.

PARSER

Given the potentially ungrammatical and often highly fragmentary nature of continuous speech input, the actual parsing of the prosodically segmented utterance is performed following a flexible, multiple-strategy, construction-specific approach as proposed among others by Carbonell & Hayes [8]), Kwasny & Sondheimer [24] and Weischedel & Black [31]. A fundamental objective associated with this kind of approach is to integrate general signal processing and natural language processing techniques (both linguistic and stochastic) in order to fully exploit the combination of partial information obtained at various stages of the analysis.

As shown earlier, the output of the speech segmentation algorithm and input to the parser is a linear sequence of parameter vectors representing the LPC-coefficients and pitch value estimates of the original continuous speech utterance at 16ms-intervals, with the F_0 contour segmented into prosodically defined intonation/information units. Typical prosodic structure representations are exemplified below in figure 3 for three short samples of Swedish (male speaker, high-quality digital recording), English (female speaker, poor-quality analogue recording) and Japanese (male speaker, toll-quality analogue recording) speech.



SWEDISH (male speaker)

Gröningen kom klockan 3 på morgonen 14 maj 1264 och England stod infor inbördeskrigets klimax. Kungen, som var den misslyckade Henrik III, sov ännu i klostret i Lewes, huvudstad i grevskapet Sussex, omgiven av sina soldater, kanske 7000 man. Han hade nyligen återkommit från Frankrike. I stadens [illegible] [illegible] normandiska som ännu reser sig ostfrån över den låga bebyggelsen i staden, befann sig Prins Edvard, kungens son, och kallad som Longshank. Edvard kommenderade 1000 beridna riddare

ENGLISH (female speaker)

The term New Methods in Marketing may make some think of Value Analysis, which can be defined as an objective and imaginative look at a product or service to see if it is possible to make it more profitable at the same cost or to supply the same quality at a lower cost, or of System selling, which means that a company, instead of selling an isolated machine or component, offers a complete system, a complete factory or power distribution system.

JAPANESE (male speaker)

Nihon no dono machi ni mo ryokan ga takusan arimasu. Sono naka ni wa yōfū no ryokan mo arimasu.

Anata wa kitto Nihon-fū no yado ni o-tomari ni naritai deshō. Ryokan wa yōfū no hoteru to wa chigaimasu.

Yado ni tsuku to jochū ga heya ni annai shite kure anata wa sugu kimono ni kikaemasu.

Heya no mannaka ni chiisana hikui shokutaku ga ari, anata wa zabuton ni suwarimasu.

Figure 3 Prosodic structure representation for three short samples of Swedish, English and Japanese speech. Arrows indicate areas of prominence outside the F_0 range defined by the baseline-topline configuration.

The calculated values of the intonation unit parameters duration, declination, onset, offset and resetting, for the baselines and toplines respectively, are stored in a 10-parameter vector and used for a first broad classification and hierarchization of the material.

Once the speech segmentation algorithm has established the extent of an intonation/information unit both in the time and in the frequeny domain, areas of prominence can be easily spotted as overshooting or undershooting pitch excursions reaching outside the F_0 range defined by the computed baseline-topline configuration. Prominence is measured by the Hz-distance above topline or below baseline respectively (compare figure 2).

Based on the probabilistic data for verb-prominence correspondences established in the previous section, the verbal components of the utterance are localized and used as points of departure for further linguistic processing. As shown among others by Waibel [30] for English and Bannert [3] for German, these pitch obtrusions provide the must reliable cue for the automatic detection if *stress* in continuous speech recognition, i.e. marking the "important" words carrying most of the semantic information content in the utterance. In addition, stressed syllables are commonly pronounced with longer durations and better articulation, which qualifies them as "islands of phonemic reliability", generally scoring better recognition rates than the unstressed (reduced, neutralized) parts of the utterance.

Parsing is run in parallel with the acoustic-phonetic classifier, following a hypothesis-driven island parsing strategy, i.e. using the areas of prominence (islands of reliability) as points of departure for inside-out processing. In other words, the classifier first forms a hypothesis about the phonetic identity of the speech segment(s) at the center of prominence. After that, the island is gradually expanded in both directions by verifying neighbour phone candidates using continuously variable hidden Markov models (HMM) [25] based on precompiled allophone/diphone/triphone statistics [16] and bounded by phonological constraints expressed in the form of finite state transition networks as proposed among others by Church [10].

Island expansion proceeds to the beginning and end of the respective intonation/information unit, thus constructing a phone lattice that spans the entire duration of the IU. A word lattice of the input utterance is hypothesized on the basis of information about (1) the most probable number of words predicted for the respective intonation/information unit as derived from the broad classification [21], (2) the language specific knowledge about the phonotactic properties within words and across words defined by the phonology-constrained diphone and triphone models, (3) the expected *case* identities generated by the *caseframe* entries in the lexicon, and (4) the lexical identities listed in a Swedish pronunciation lexicon [17]. Syntactic (including morphological) constraints are only weakly defined in a constituent-based context-free grammar formulation (CFG), which is aimed to permit successful parses even for fragmentary and/or grammatically deficient speech input and is expected to support the pruning of "unpromising" parses at an early stage of the analysis.

It must be appreciated in this context that only about one fifth of all intonation/information units unearthed by the speech segmentation algorithm (18.2% in our Swedish material) align in a simple one-to-one fashion with full sentences, while the majority (81.8% in the Swedish material) aligns with features of constituent structure in the sub-sentence domain. This implies that the overwhelming majority of full sentences (grammatical as well as ungrammatical) contained in continuous speech is processed in terms of several intonation/information units. Empirical study of our accumulated Swedish speech material revealed an average of 2.36 IUs per sentence with three clearly defined modes at 2, 3 and 5 IUs [20]. It must be appreciated in this context that sentences composed of 4 or more intonation/information units typically contain parallel structures such as enumerations, appositions, parentheticals and rhetorical repetitions.

Given the limited number of actually occurring IU-per-sentence constellations represented by the combination of (1) the most probable number of IUs per sentence, (2) the internal properties of each individual IU specified in a 10-parameter vector containing duration, onset, offset, slope and resetting values for the baseline and topline respectively, and (3) the scored lattice of constituent label(s) derived from the coarse-classification procedure, the sub-problem of sentence generation by intonation unit concatenation can be conveniently solved by a finite-state parsing strategy such as proposed by Gibbon [13], i.e. using a *finite-state automation* (FSA) with transition probabilities attached to each arc.

SUMMARY AND CONCLUSIONS

The speech segmentation, classification and hierarchization components have been developed for Swedish speech input. Testing the algorithm for English and Japanese speech input is ongoing and shows promising results. Further research focuses on improvements in the definition of the linguistic description format (*i.e.* incorporating nominal caseframes, attaching probability scores for *cases* in the "opt" state, including lexical hypotheses with the caseframe entries, integrating the case grammar with a functional grammar component, etc).

We like to believe that the approach presented in this study shows promise not only for spoken input parsing in general, but for a number of practical applications in the field of speech processing including telecommunication, interpreting telephony, automatic keyword extraction, and text-to-speech synthesis. Linear regression lines are easily calculated and require only little computational effort, which makes the segmentation algorithm a fast, robust and objective technique for computer speech applications. Modulating voice for increased informativity exploits a natural strategy that human speakers use quite automatically in communicative situations involving channel deficiency (e.g. due to static, transmission noise, or masking effects) and/or different kinds of ambiguity [22]. Prominent pitch excursions (together with greater segmental durations) constitute a universally used feature of language that is employed to signal new *versus* given, contrastive *versus* presupposed, thematic *versus* rhematic information in connected speech utterances [7] and can thus be used as a reliable cue to quickly identify the semantically potent keywords in the message. In addition, the frequency range covered by voice phenomena (intonation, accentuation, laryngealization) lies safely within the normal band limits of telecommunication, which qualifies F_0 as a natural, versatile, and accessible code for human-computer interaction via telephone.

Finally, text-to-speech systems using standard syntactic parsers designed to find "major syntactic boundaries" at which the intonation contour needs to be broken into separate units that help the listener to decode the message, invariably come up with the same two kinds of problems [23]:

1 - they tend to produce not one (the most probable, semantically most plausible) but several alternative parses;
2 - they produce too many boundaries at falsely detected or inappropriate sentence locations.

Perceptual evaluation of these synthesized contours reveals that listeners get distracted and often even plainly confused by too many prosodically marked boundaries, while too few prosodic breaks just sound like as if the speaker simply is talking too fast. These findings not only show that the amount of segmentation and the correspondence between syntactic and prosodic units are dependent on the rate of speech, but that listeners apparently neither expect, nor need, nor even want prosodically cued information about all the potential richness in syntactic structure described by modern syntactic theories, in order to decode the intended meaning of an utterance.

REFERENCES

[1] B Altenberg, "Prosodic Patterns in Spoken English", Lund University Press, 1987

[2] H Altmann, "Zur Problematik der Konstitution von Satzmodi als Formtypen", in: J Meibauer (ed) Satzmodus zwischen Grammatik und Pragmatik, Max Niemeyer Verlag, Tübingen 1987

[3] R.Bannert, "From prominent syllables to a skeleton of meaning: a model of prosodically guided speech recognition", *Proceedings of the XIth International Congress of Phonetic Sciences*, Tallinn (Estonia) 1987

[4] A Barr and E A Feigenbaum, "The Handbook of Artificial Intelligence", Stanford 1981

[5] A Batliner, "Produktion und Prädiktion. Die Rolle intonarischer und anderer Merkmale bei der Bestimmung des Satzmodus", in: H Altmann (ed) Intonationsforschungen, Max Niemeyer Verlag, Tübingen 1988

[6] H Broman, P Brauer, E Eliassen, P Hedelin, D Huber and P Knagenhjelm, "Classification: A Problem of Optimization or Organization?", *Proceedings of the STU-Symposium "Digital Communication"*, Stockholm 1989

[7] G Brown, "Prosodic structure and the given/new distinction", in: A.Cutler and D.R.Ladd (eds), Prosody: Models and Measurements, Springer-Verlag, 1983

[8] J G Carbonell and P J Hayes, "Robust parsing using multiple construction-specific strategies", in: L Bolc (ed), Natural Language Parsing Systems, Springer-Verlag, Berlin 1987

[9] W L Chafe, "Givenness, contrastiveness, definiteness, subjects, topics, and points of view", in: Charles Li (ed), Subject and Topic, Academic Press, New York 1976

[10] K W Church. "Phonological Parsing in Speech Recognition". Kluwer Academic Publishers. 1987

[11] Ch Fillmore. "The case for case". in: E Bach and R T Harms. Universals in Linguistic Theory. Holt. Rinehart and Winston. Chicago 1968

[12] J P Gee and F Grosjean. "Performance structures: a psycnolinguistic and linguistic appraisal". *Cognitive Psychology* **15**. 1983

[13] D Gibbon. "Finite state processing of tone systems". *ACL Proceedings. 1987*

[14] M A K Halliday. "Theme and Information in the English Clause". Oxford University Press. 1976

[15] Ph J Hayes. A G Hauptmann. J G Carbonell and M Tomita. "Parsing spoken language: a semantic caseframe approach". *ACL Proceedings.* 1986

[16] P Hedelin. D Huber and A Leijon. "Probability distribution of allophones. diphones and triphones in phonetic transcriptions of Swedish newspaper text". Chalmers Report **8**. 1988

[17] P Hedelin. A Jonsson and P Lindblad. "Svensk Uttalslexicon (Swedish Pronunciation Lexicon)". Chalmers Report 4. 1989

[18] D Huber. "On the Communicative Function of Voice in Human-Computer Interaction". STIMDI 2. Stockholm 1988

[19] D Huber. "Laryngealization as a Boundary Cue in Read Speech". *Proceedings of the Second Swedish Phonetics Conference.* Lund 1988

[20] D Huber. "Aspects of the Communicative Function of Voice in Text Intonation". PhD Dissertation. Göteborg 1988

[21] D Huber. "A statistical approach to the segmentation and broad classification of continuous speech into phrase-sized information units". *Proceedings ICASSP 89.* Glasgow 1989

[22] D Huber. "Prosodic Contributions to the Resolution of Ambiguity". *Proceedings of the Conference NORDIC PROSODY V.* Åbo (Finland). 1989

[23] D H Klatt. "Review of text-to-speech conversion for English". *Journal of the Acoustical Society of America* **82**(3). 1987

[24] S C Kwasny and N K Sondheimer. "Ungrammaticality and extra-grammaticality in natural language understanding systems". *Proceedings of the 17th Annual Meeting of the Association for Computational Linguistics.* La Jolla. Cal. 1979

[25] S E Levinson. "Continuously variable hidden Markov models for automatic speech recognition". *Computer Speech and Language* **1**. 1986

[26] J Löfström. "Repliker utan Gränser (Boundless Conversational Exchanges)". PhD Dissertation. Göteborg 1989

[27] W Oppenrieder. "Intonarische Kennzeichnung von Satzmodi". in: H Altmann (ed) Intonationsforschungen. Max Niemeyer Verlag. Tübingen 1988

[28] R F Simmons. "Semantic networks: their computation and use for understanding English sentences". in: R C Schank and K M Colby (eds). Computer Models of Thought and Language. W H Freeman & Co. San Francisco 1973

[29] R P Stockwell. P Schachter and B H Partee. "The Major Syntactic Structures of English". Holt. Rinehart and Winston. New York 1973

[30] A Waibel. "Prosodic knowledge sources for word hypothesization in a continuous speech recognition system". *Proceedings ICASSP 87.* Dallas 1987

[31] R M Weischedel and L Black. "Responding to potentially unparseable sentences". *American Journal of Computational Linguistics* **6**. pp.97-109. 1980

Parsing Continuous Speech by HMM-LR Method

Kenji KITA, Takeshi KAWABATA, Hiroaki SAITO

ATR Interpreting Telephony Research Laboratories
Seika-chou, Souraku-gun, Kyoto 619-02, JAPAN

Abstract

This paper describes a speech parsing method called HMM-LR. In HMM-LR, an LR parsing table is used to predict phones in speech input, and the system drives an HMM-based speech recognizer directly without any intervening structures such as a phone lattice. Very accurate, efficient speech parsing is achieved through the integrated processes of speech recognition and language analysis. The HMM-LR method is applied to large-vocabulary speaker-dependent Japanese phrase recognition. The recognition rate is 87.1% for the top candidates and 97.7% for the five best candidates.

1 Introduction

This paper describes a speech parsing method called HMM-LR. This method uses an efficient parsing mechanism, a generalized LR parser, driving an HMM-based speech recognizer directly without any intervening structures such as a phone lattice.

Generalized LR parsing [1] is a kind of LR parsing [2], originally developed for programming languages and has been extended to handle arbitrary context-free grammars. An LR parser is guided by an LR parsing table automatically created from context-free grammar rules, and proceeds left-to-right without backtracking. Compared with other parsing algorithms such as the CYK (Cocke-Younger-Kasami) algorithm [3] or Earley's algorithm [4], a generalized LR parsing algorithm is the most efficient algorithm for natural language grammars.

There have been some applications of generalized LR parsing to speech recognition. Tomita [5] proposes an efficient word lattice parsing algorithm. Saito [6] proposes a method of parsing phoneme sequences that include altered, missing and/or extra phonemes. However, these methods are inadequate because of the information loss due to signal-symbol conversion. The HMM-LR method does not use any intervening structures. The system drives an HMM-based speech recognizer directly for detecting/verifying phones predicted using an LR parsing table.

HMM (Hidden Markov Models) [7] has the ability to cope with the acoustical variation of speech by means of stochastic modeling, and it has been used widely for speech recognition. In HMM, any word models can be composed of phone models. Thus, it is easy to construct a large vocabulary speech recognition system.

This paper is organized as follows. Section 2 describes the LR parsing mechanism. Section 3 describes HMM. Section 4 describes the HMM-LR method. Section 5 describes recognition experiments using HMM-LR. Finally, section 6 presents our conclusions.

2 LR Parsing

2.1 LR Parsing

LR parsing was originally developed for programming languages. It is applicable to a large class of context-free grammars.

The LR parser is deterministically guided by an LR parsing table with two subtables (*action table* and *goto table*). The action table determines the next parser action *ACTION*[*s*,*a*] from the state *s* currently on top of the stack and the current input symbol *a*. There are four kinds of actions, *shift*, *reduce*, *accept* and *error*. *Shift* means shift one word from input buffer onto the stack, *reduce* means reduce constituents on the stack using the grammar rule, *accept* means input is accepted by the grammar, and *error* means input is not accepted by the grammar. The goto table determines the next parser state *GOTO*[*s*,*A*] from the state *s* and the grammar symbol *A*.

The LR parsing algorithm is summarized below.

1. *Initialization.* Set *p* to point to the first symbol of the input. Push the initial state 0 on top of the stack.
2. Consult *ACTION*[*s*,*a*] where *s* is the state on top of the stack and *a* is the symbol pointed to by *p*.
3. If *ACTION*[*s*,*a*] = "*shift s'* ", push *s'* on top of the stack and advance *p* to the next input symbol.
4. If *ACTION*[*s*,*a*] = "*reduce A*$\rightarrow$β", pop |β| symbols off the stack and push *GOTO*[*s'*,*A*] where *s'* is the state now on top of the stack.
5. If *ACTION*[*s*,*a*] = "*accept*", parsing is completed.
6. If *ACTION*[*s*,*a*] = "*error*", parsing fails.
7. Return to 2.

2.2 Generalized LR Parsing

Standard LR parsing cannot handle ambiguous grammars. For an ambiguous grammar, the LR parsing table will have *multiple entries* (*conflicts*). As a general method, a stack-splitting mechanism can be used to cope with multiple entries. Whenever a multiple entry is encountered, the stack is divided into two stacks, and each stack is processed in parallel. Thus, it is possible to use LR parsing to handle an ambiguous grammar which describes natural language.

(1) S	→	NP V
(2) S	→	V
(3) NP	→	N
(4) NP	→	N P
(5) N	→	k o r e
(6) P	→	o
(7) V	→	k u r e
(8) V	→	o k u r e

Fig.1 Example grammar

	e	o	u	k	r	$	S	N	V	P	NP
0		s5		s2			6	1	3		4
1		s7,r3		r3						8	
2		s9	s10								
3						r2					
4		s5		s11					12		
5				s13							
6						acc					
7				r6							
8				r4							
9					s14						
10					s15						
11			s10								
12						r1					
13			s16								
14	s17										
15	s18										
16					s19						
17		r5		r5							
18						r7					
19	s20										
20						r8					

Fig.2 LR parsing table

A simple example grammar is shown in Fig.1, and the LR parsing table, compiled from the grammar automatically, is shown in Fig.2. The left part is the action table and the right part is the goto table. The entry "acc" stands for the action "*accept*", and blank spaces represent "*error*". The terminal symbol "$" represents the end of the input.

3. HMM (Hidden Markov Models)

HMM is effective in expressing speech statistically, so it has been used widely for speech recognition.

Fig.3 shows an example of a phone model. A model has a collection of *states* connected by *transitions*. Two sets of probabilities are attached to each transition. One is a *transition probability* a_{ij}, which provides the probability for taking transition from state i to state j. The other is an *output probability* b_{ijk}, which provides the probability of emitting code k when taking a transition from state i to state j.

The *forward-backward algorithm* [7] can be used to estimate the model's parameters given a collection of training data. After estimating the model's parameters, the *forward algorithm* (*trellis algorithm*) can be used to verify phones as follows.

$$\alpha_i(t) = \begin{cases} 1 & (t = 0 \ \& \ i = 0) \\ 0 & ((t = 0 \ \& \ i \neq 0) \ or \ (t \neq 0 \ \& \ i = 0)) \\ \sum_j (\alpha_j(t\text{-}1) a_{ji} b_{ji}(y_t)) & \end{cases}$$

$\alpha_i(t)$ is the probability that the Markov process is in state i having generated code sequence $y_1, y_2, \ldots, y_t$. The final probability for the phone is given by $\alpha_F(T)$ where F is a final state of the phone model and T is a length of input code sequence.

4. HMM-LR Method

4.1 Basic Mechanism

In standard LR parsing, the next parser action (shift, reduce, accept or error) is determined using the current parser state and next input symbol to check the LR parsing table. This parsing mechanism is valid only for symbolic data and cannot be applied simply to continuous data such as speech.

In HMM-LR, the LR parsing table is used to predict the next phone in the speech. For the phone prediction, the grammar terminal symbols are phones instead of the grammatical category names generally used in natural language processing. Consequently, a lexicon for the specified task is embedded in the grammar.

The following describes the basic mechanism of HMM-LR (see Fig.4). First, the parser picks up all phones which the initial state of the LR parsing table predicts, and invokes the HMM to verify the existence of these predicted phones. During this process, all possible parsing trees are constructed in

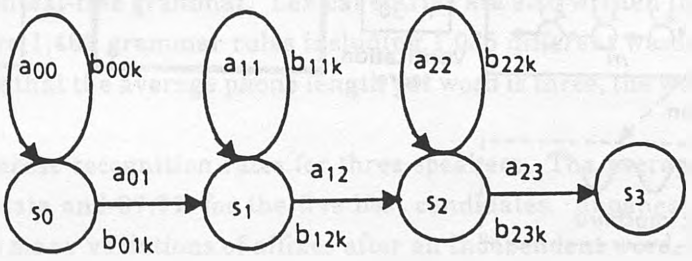


Fig. 3 HMM phone model

parallel. The HMM phone verifier receives a probability array which includes end point candidates and their probabilities, and updates it using an HMM probability calculation process (the forward algorithm). This probability array is attached to each partial parsing tree. When the highest probability in the array is lower than a threshold level, the partial parsing tree is pruned by threshold level, and also by beam-search technique. The parsing process proceeds in this way, and stops if the parser detects an accept action in the LR parsing table. In this case, if the best probability point reaches the end of speech data, parsing ends successfully. A very accurate, efficient parsing method is achieved through the integrated process of speech recognition and language analysis. Moreover, HMM units are phones, and any word models can be composed of phone models, so it is easy to construct a large vocabulary speech recognition system.

4.2 Algorithm

To describe an algorithm for the HMM-LR method, we first introduce a data structure named *cell*. A cell is a structure with information about one possible parsing. The following are kept in the cell:

- LR stack, with information for parsing control.
- Probability array, which includes end point candidates and their probabilities.

The algorithm is summarized below.

1. *Initialization.* Create a new cell C. Push the LR initial state 0 on top of the LR stack of C. Initialize the probability array Q of C;

$$Q(t) = \begin{cases} 1 & t = 0 \\ 0 & 1 \leqq t \leqq T \end{cases}$$

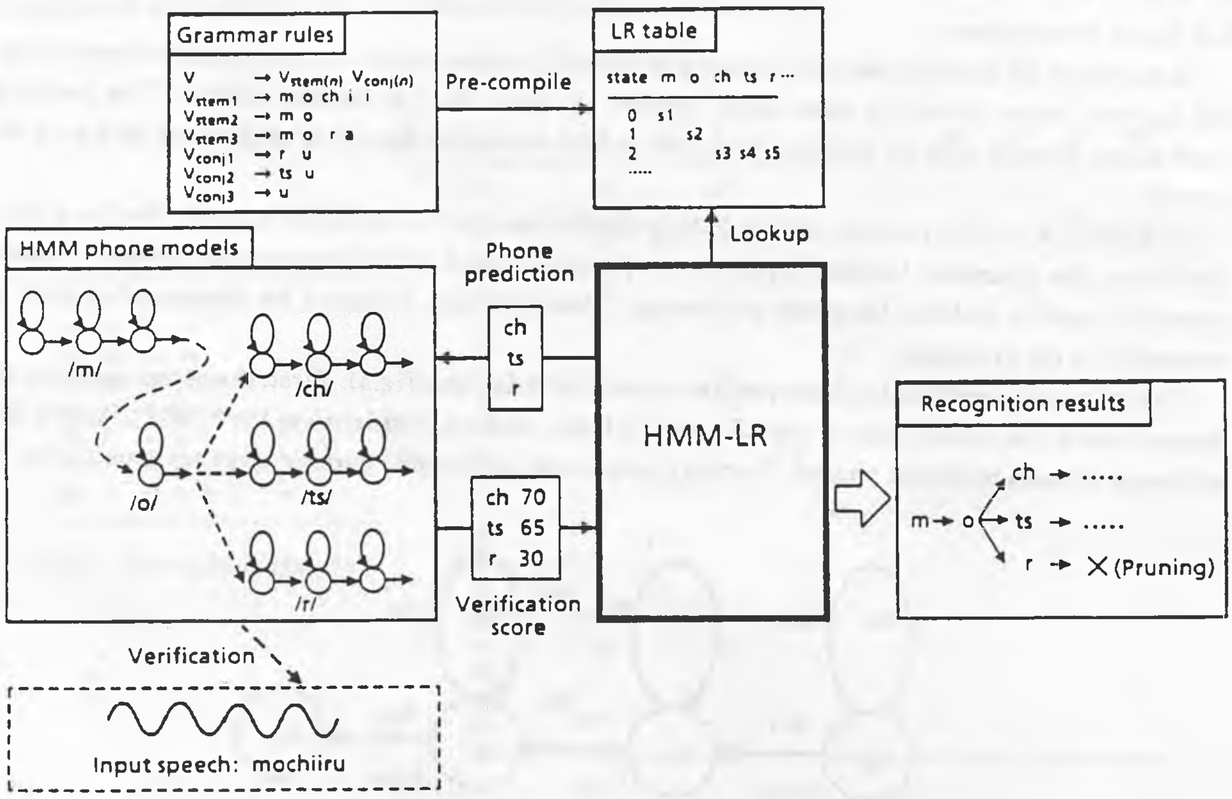


Fig. 4 Basic mechanism of HMM-LR

2. *Ramification of cells.* Construct a set

$$S = \{(C, s, a, x) \mid \exists C, a, x\,(\, C \text{ is a cell } \& \ C \text{ is not accepted } \& \ s \text{ is a state of } C \ \& \ ACTION[s,a] = x \ \& \ x \neq \text{"error"}\}.$$

   For each element $(C, s, a, x) \in S$, do operations below. If a set S is empty, parsing is completed.

3. If $x =$ "*shift s'* ", verify the existence of phone a. In this case, update the probability array Q of the cell C by the following computation.

$$\alpha_i(t) = \begin{cases} Q(t) & (i = 0) \\ 0 & (t = 0 \ \& \ i \neq 0) \\ \Sigma_j\,(\alpha_j(t\text{-}1)a_{ji}b_{ji}(y_t)) & \end{cases}$$

$$Q(t) = \begin{cases} 0 & (t = 0) \\ \alpha_F(t) & \end{cases}$$

   If $max\ Q(i)$ $(i = 1 \ldots T)$ is below a threshold level set in advance, the cell C is abandoned. Else push s' on top of the LR stack of the cell C.

4. If $x =$ "*reduce* $A \rightarrow \beta$", same as standard LR parsing.
5. If $x =$ "*accept*" and $Q(T)$ is larger than a threshold level, the cell C is accepted. If not, cell C is abandoned.
6. Return to 2.

Recognition results are kept in cells. Generally, many recognition candidates exist, and it is possible to rank these candidates using a value $Q(T)$.

The set S constructed in step 2 above is quite large. It is possible to set an upper limit on the number of elements in S by beam-search technique. It is also possible to use *local ambiguity packing* [1] to represent cells efficiently.

5. Experiments

The HMM-LR method is applied to speaker-dependent Japanese phrase recognition. Duration control techniques and separate vector quantization are used to achieve accurate phone recognition. Two duration control techniques are used, one is phone duration control for each HMM phone model and the other is state duration control for each HMM state [8]. Phone duration control is carried out by weighting HMM output probabilities with phone duration histograms obtained from training sample statistics. State duration control is realized by state duration penalties calculated by modified forward-backward probabilities of training samples. In separate vector quantization, spectral features, spectral dynamic features and energy are quantized separately. In the training stage, the output vector probabilities of these three codebooks are estimated simultaneously and independently, and in the recognition stage all the output probabilities are calculated as a product of the output vector probabilities in these codebooks.

The grammar used in the experiments describes a general Japanese syntax of phrases and is written in the form of context-free grammar. Lexical entries are also written in the form of context-free grammar. There are 1,461 grammar rules including 1,035 different words, and perplexity per phone is 5.87. Assuming that the average phone length per word is three, the word perplexity is more than 100.

Table 1 shows the phrase recognition rates for three speakers. The average recognition rate is 87.1% for the top candidate and 97.7% for the five best candidates. Japanese is an agglutinative language, and there are many variations of affixes after an independent word. The problem here is that recognition errors are often mistakes caused by these affixes.

Table 1 Phrase recognition rates

Speaker / Order	MAU	MHT	MNM	Average
1	87.8	85.6	87.8	87.1
2	98.6	93.5	92.8	95.0
3	99.3	96.8	95.0	97.0
4	99.3	97.5	95.7	97.5
5	99.6	97.8	95.7	97.7

6. Conclusion

In this paper, we described a speech parsing method called HMM-LR, which uses a generalized LR parsing mechanism and an HMM-based speech recognizer. The experiment results show that an HMM-LR method is very effective in continuous speech recognition.

An HMM-LR continuous speech recognition system is used as part of the SL-TRANS (Spoken Language TRANSlation) system developed at ATR Interpreting Telephony Research Laboratories.

Acknowledgements

The authors would like to express their gratitude to Dr.Akira Kurematsu, president of ATR Interpreting Telephony Research Laboratories, for his encouragement and support, which made this research possible, and to Mr.Toshiyuki Hanazawa for the HMM program.

References

[1] Tomita, M.: *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems*, Kluwer Academic Publishers (1986).

[2] Aho, A.V., Sethi, R. and Ullman, J.D.: *Compilers, Principles, Techniques, and Tools*, Addison-Wesley (1986).

[3] Aho, A.V. and Ullman, J.D.: *The Theory of Parsing, Tranlation, and Compiling*, Prentice-Hall, Englewood Cliffs (1972).

[4] Earley, J.: *An Efficient Context-Free Parsing Algorithm*, Comm. ACM, Vol.13, No.2, pp.94-102 (1970).

[5] Tomita, M.: *An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition*, Proc. IEEE Int. Conf. Acoust. Speech Signal Process. ICASSP-86, pp.1569-1572 (1986).

[6] Saito, H. and Tomita, M.: *Parsing Noisy Sentences*, Proc. 12th Int. Conf. Comput. Linguist. COLING-88, pp.561-566 (1988)

[7] Levinson, S.E., Rabiner, L.R. and Sondhi, M.M.: *An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition*, Bell Syst. Tech. J., Vol.62, No.4, pp.1035-1074 (1983).

[8] Hanazawa, T., Kawabata, T. and Shikano, K.: *Duration Control Methods for HMM Phoneme Recognition*, The Second Joint Meeting of ASA and ASJ (1988).

Parsing Japanese Spoken Sentences Based on HPSG

Kiyoshi KOGURE

ATR Interpreting Telephony Research Laboratories
Sanpeidani, Inuidani, Seika-Cho, Soraku-gun, Kyoto 619-02, Japan
kogure%atr-la.atr.junet@uunet.uu.net

ABSTRACT

An analysis method for Japanese spoken sentences based on HPSG has been developed. Any analysis module for the interpreting telephony task requires the following capabilities: (i) the module must be able to treat spoken-style sentences; and, (ii) the module must be able to take, as its input, lattice-like structures which include both correct and incorrect constituent candidates of a speech recognition module. To satisfy these requirements, an analysis method has been developed, which consists of a grammar designed for treating spoken-style Japanese sentences and a parser designed for taking as its input speech recognition output lattices. The analysis module based on this method is used as part of the NADINE(Natural Dialogue Interpretation Expert) system and the SL-TRANS (Spoken Language Translation) system.

1. INTRODUCTION

An analysis module for a spoken sentence translation system, or an interpreting telephony system requires the following capabilities:
(i) the module must be able to treat spoken-style sentences; and,
(ii) the module must be able to accept not only strings but also lattice-like structures where the analysis module directly drives a speech recognition module (e.g., a phoneme or word recognition module but not a whole sentence recognition module) or where the analysis module takes as its inputs partial speech recognition results including both correct and incorrect sentence constituents.

To satisfy these requirements, an analysis method has been developed which consists of a grammar framework designed for treating spoken-style Japanese sentences and a unification-based parser designed for taking as its input speech recognition result lattices.

The grammar framework is unification-based lexico-syntactic and is essentially based on HPSG[10] and JPSG[2]. This is because:
(i) a lexico-syntactic approach is modular in the sense that most of the grammatical information is to be specified in descriptions of lexical items; and that it is therefore easy to extend a grammar simply by adding new lexical items to the lexicon or adding new information to lexical items; and
(ii) the JPSG framework can essentially capture constraints between complex predicate constituents and their complements. This capability is important because spoken-style Japanese sentences often have complex predicate constituents.
The grammar framework is extended from these grammatical frameworks by introducing features related to semantic and pragmatic constraints[12].

The parser developed is essentially based on the active chart parsing algorithm[11] because the algorithm is as efficient as Earley's algorithm[1] or any other CFG parsing algorithm and, moreover, has the capability of controlling parsing strategies to avoid exhaustive searches. The parser is extended to treat constraints in Typed Feature Structures (TFS) by using TFSP links (as defined in Section 3).

The analysis method proposed in this paper is used in the analysis module of the NADINE system[4,9] and the NADINE system is used as the machine translation module of the SL-TRANS system. In the SL-TRANS system, input speech is recognized by the Japanese *bunsetsu*[1] phrase recognition module based on the HMM-LR method[8] and the module outputs the sequence of *bunsetsu* phrase lattices, each of which consists of bunsetsu phrase structure candidates. The outputs are filtered by a *bunsetsu* dependency filter module[5] which outputs sentence lattices consisting of fewer *bunsetsu* phrase structure candidates than the HMM-LR produces.

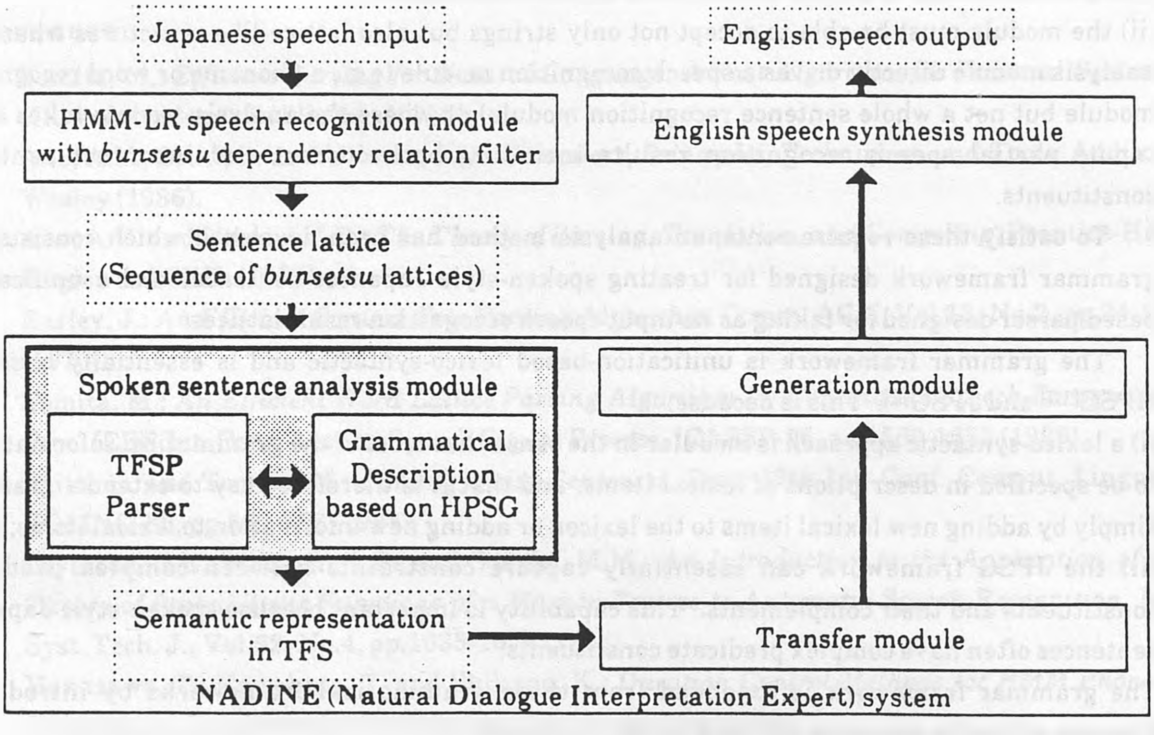
The NADINE system takes as its input a sentence lattice and outputs an English sentence. The analysis module based on this paper's method takes a sentence lattice and outputs typed feature structures which represent syntactic, semantic and pragmatic information of the sentence. Then, the transfer and generation modules output an English sentence.

In this paper, Section 2 describes the grammar framework and Section 3 describes the parser and the analysis method.

2. GRAMMAR FRAMEWORK FOR SPOKEN-STYLE JAPANESE SENTENCES

The grammar built up to analyze spoken-style Japanese sentences is essentially based on HPSG and JPSG. The grammar describes not only syntactic and semantic information but also discourse and pragmatic information in an integrated way by using TFS descriptions.

Resolution of omitted obligatory cases (or zero-pronouns) is very important because



Japanese speech input
HMM-LR speech recognition module with *bunsetsu* dependency relation filter
Sentence lattice (Sequence of *bunsetsu* lattices)
Spoken sentence analysis module
TFSP Parser
Grammatical Description based on HPSG
Semantic representation in TFS
Transfer module
Generation module
English speech synthesis module
English speech output
NADINE (Natural Dialogue Interpretation Expert) system

Fig.1 Overview of the SL-TRANS system (modules related to the analysis module)

1. a basic phonological phrase consisting of a *jiritsugo*-word such as a noun, verb, or adverb followed by zero or more *fuzokugo*-words such as auxiliary verbs, postpositional particles, or sentence final particles.

(i) pronouns referring to the speaker and the hearer seldom appear in spoken-style sentences and these omitted cases make sentences more ambiguous, and
(ii) in order to translate these sentences into natural English sentences, they must be supplemented.
If they are not supplemented, for example, Japanese sentences without agent subject case expressions must often be translated into unnatural English passive sentences (e.g., "*A registration form will be sent*" instead of "*I will send you a registration form*"). In this paper's analysis, such omitted cases are resolved by using constraints on the uses of deictic expressions and their case elements, and so on.

2.1. Treatment of Syntactic and Semantic Information

Spoken-style Japanese sentences often have complex sentence final predicate phrases consisting of main predicates and combinations of auxiliary verbs and sentence final particles. In such a predicate phrase, its head constituent stipulates the properties of the complement occurring just on its left such as its part of speech, conjugational type, and conjugational form. Such stipulations are easily described in the SUBCAT feature value in the head. A SUBCAT feature value is a list of complement constituent specifications.

For example, in the lexical description (1) of the causative auxiliary verb "*seru*", the SUBCAT feature value specifies that the auxiliary takes as its complement a verb phrase with conjugational type CONS (for consonant type) and conjugational form VONG (for voice negative type), and two postpositional phrases (PPs), a PP marked by "*ni*" and a PP marked by "*ga*". Moreover, it specifies that the VP must be located just before the auxiliary and that the relative order between two PPs is free. The SEMF feature, which is a bundle of semantic features, specifies the semantic selectional restrictions and, in the description, the SEMF feature value of the *ga*-PP specifies that the PP must refer to an animate object.

```
[[syn [[morph [[ctype vow][cform aspl-or-infn]]]
      [head [[pos v]
             [modl [[caus +]]]
             ...]]
      [subcat [[first [[syn [[morph [[ctype cons][cform vong]]]
                              [head [[pos v]
                                     [modl [[caus -][deac -] ...]]]]
                              [subcat [[first [[syn [[head [[form ga]
                                                            ...]] ...] ...]
                                               [sem ?causee]]]
                                       [rest end]]]]]
                        [sem ?caused]]]
                [rest (:perm-list [[syn [[head [[form ga] ...]] ...]]
                                   [semf [[human +]]]
                                   [sem ?causer]]
                                  [[syn [[head [[form ni] ...]] ...]]
                                   [sem ?causee]])]]] ...] ...]
 [sem [[relation cause]
       [causer   ?causer]
       [causee   ?causee]
       [caused   ?caused]]]]                                        (1)
```

where "?" is the prefix of the tag and structures denoted by the same tag are token identical, and ":perm-list" is a macro which takes as its arguments a set of typed feature structure descriptions and returns as its value the disjunction of permuted lists made of the set.

Furthermore, the COH feature (Category Of Head) in a complement or adjunct constituent specifies its head constituents. Combinations of COH and SUBCAT features allow flexible grammatical descriptions.

Japanese predicate constituents belong to groups: a member of these groups must, with some exceptions, occur in a strictly one-dimensional sequence; these groups correspond to semantic hierarchies. A new head feature MODL (for modality) has been devised to all and only predicates with grammatically ordered constituents. For example, in the above description (1), the MODL feature value of the first SUBCAT value element specifies that the complement verb phrase should not include any auxiliary verbs.

Besides the predicate constituent order specification, the MODL feature is also used to restrict syntactic and semantic behavior of subordinate (adverbial) phrases. For example, certain formal adverbs (i.e., subordinate conjunctions) require as their complements verb phrases without time or place modifiers. Such requirements reduce ambiguities of adverbial phrase modificands. The MODL feature in conjunction with the SEMF feature contribute to reducing the number of verbal modificand ambiguities.

2.2. Treatment of Pragmatic Constraints on Uses of Expressions

This grammar framework treats discourse or pragmatic constraints on uses of expressions in order to select plausible analysis candidates and to resolve certain kinds of zero-pronouns. An analysis candidate includes not only syntactico-semantic descriptions such as a semantic interpretation (the SEM feature value) but also annotations or a set of conditions under which the interpretation is valid. For example, the sentence

```
Watashi ni  tourokuyoushi     o   o-okuri  itadake        masu   ka
I       DAT registration form ACC HON-send RECEIVE-FAVOR POLITE QUESTION
```

seems to have two analysis candidates corresponding to phrase structures (a) and (b) in Fig.2 (they correspond to "*Could you please send me a registration form?*" and ??"*Could I please send a registration form?*"). However, the analysis candidate corresponding to (b) has the following annotations:

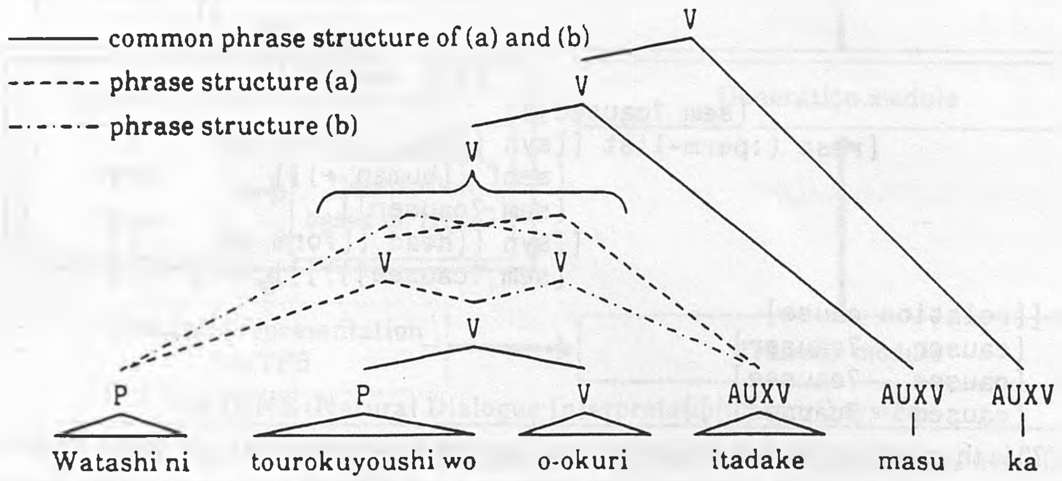


Fig.2 Two derivation trees of the sentence "watashi ni tourokuyoushi o o-okuri itadake masu ka"

```
[[relation             condescend]
 [agent                ?speaker]
 [object               ?subject_sem]
 [comparative-object ?speaker]]
[[relation             express-more-empathy]
 [agent                ?speaker]
 [object               ?subject_sem]
 [comparative-object ?speaker]]
```

(where ?speaker refers to the speaker and ?subject_sem is the semantic representation of the subject of "*itadake*").

Accordingly, these conditions are unnatural (e.g., the speaker expresses more empathy to a person other than himself) but (a) does not have such unnatural conditions. Thus, the analysis (a) is selected as a more plausible candidate than (b).

These annotations are also used for zero-pronoun resolution. In the analysis (a), the subject and indirect object of "*itadake*" are missing. However, (a) has the following annotations:

```
[[relation             condescend]
 [agent                ?speaker]
 [object               ?subject_sem]
 [comparative-object ?indirect-object_sem]]
[[relation             express-more-empathy]
 [agent                ?speaker]
 [object               ?subject_sem]
 [comparative-object ?indirect-object_sem]]
```

and by searching for discourse participants satisfying these conditions, candidates of missing elements can be found.

In order to obtain such annotations, lexical descriptions have PRAG| RESTRS features which include constraints in terms of **RESPECT**, **CONDESCEND**, **POLITE**, **EXPRESS-MORE-EMPATHY** and so on.

Plausibility scores based on these annotations are used in conjunction with other kinds of scores described below to select plausible analysis candidates. Zero-pronoun resolution is applied after parsing. Annotations are used in conjunction with conditions under which utterances of sentences are interpreted as certain types of illocutionary acts, and conditions under which actions in general are rational.

3. FEATURE STRUCTURE PROPAGATION PARSER

3.1. Active Chart Parser with Feature Structure Propagation Links

The active chart parsing algorithm has properties suitable for parsing natural language efficiently. In particular, it has two excellent properties for treating speech recognition result lattices:

(i) it does not limit its inputs to only strings but can accept lattice structures — thus, it can parse speech recognition result lattices directly; and,

(ii) it has the capability of controlling the order of parsing by adapting a method of selecting pending edges from the pending edge list, which works as an agenda. Thus, by adapting a selection method based on certain criteria which, at least, reflects speech recognition result plausibility, plausible parses can be obtained in the early stages without exhaustive search.

However, this second property makes structure sharing difficult in unification-based CFG parsing, or CFG parsing augmented by constraints described in typed feature structures (TFSs). In unification-based parsing, there often exist edges with the same content except for their TFSs. When an active edge is continued with an inactive edge, if there is already an edge with the same contents except for its TFSs as the continuation edge, edge sharing may seem to be able to be achieved by adding the continuation edge's TFSs into the existing edge's. However, this makes parsing incomplete because the existing edge may have been used previously to construct larger edges due to the parsing order freeness and because newly added TFSs are not used to construct larger edges or used as part of larger edges.

In order to solve this problem, the **TFS Propagation parser** (in short, **TFSP parser**) has been developed. The parser is essentially based on active chart parsing and each edge of the parser has a set of TFSs representing syntactic, semantic and pragmatic information of corresponding partial phrase structures. The parser is extended to have special links called **TFS** Propagation links (TFSP links).

A TFSP link in an edge remembers how the TFSs of the edge were previously propagated and specifies how TFSs newly added into the edge should be used. That is, a TFSP link of an active edge points to a continuation edge having as its annotation the inactive edge used to construct the continuation edge. Then, when a TFS is added to an active edge, for each TFSP link of the edge, the TFS is unified with each TFS of the link's inactive edge and then the unification result TFS is added into the link's continuation edge if the unification succeeds. By using TFSP links, new edge creation is necessary only when there is no edge with a certain starting vertex, ending vertex, label and remainder symbol sequence. The TFSP link makes edge structure sharing possible.

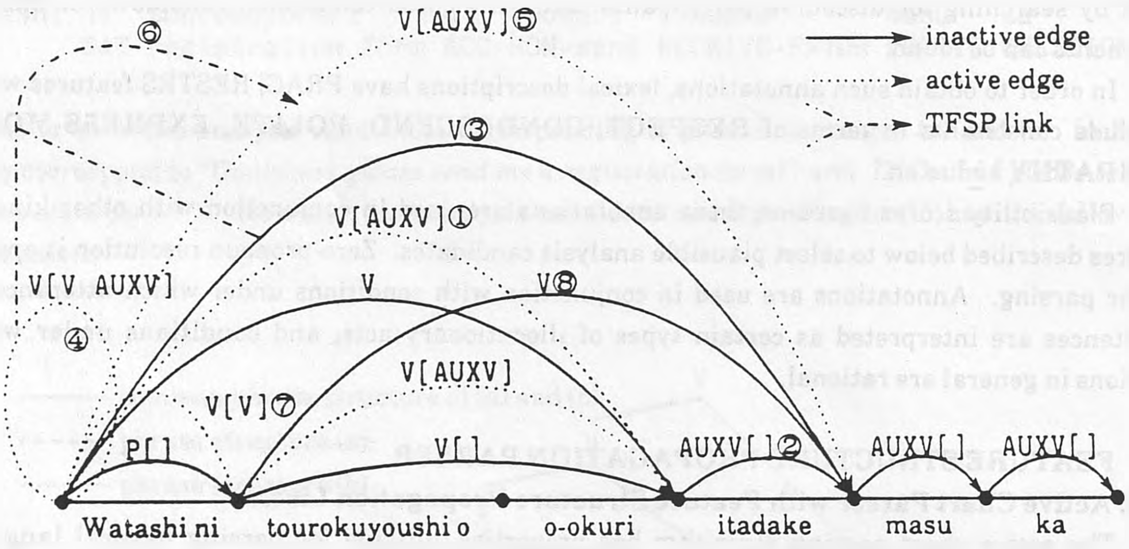


Fig.3 TFSP links

Suppose the case where the inactive edge ③ has been created from the active edge ① and the inactive edge ② and the inactive edge ⑤ has been created from the active edge ④ and the inactive edge ③. The TFSP link ⑥ is created between ④, ③ and ⑤. In this case, when the active edge ⑦ is continued with the inactive edge ⑧, the successful unification result TFSs of ⑦'s and ⑧'s TFSs are added to the edge ③. The edge has a TFSP link and then the newly added TFSs are unified with TFSs in ④ and the successful unification results are propagated to the edge ⑤ as specified by the TFS link ⑥. If there are already TFS links in the edge ⑤, the newly added TFSs are also propagated in the ways specified by these links.

The TFSP link enables the parser to reduce unnecessary edge structure creation and TFS unification. When an active edge is continued with an inactive edge, the continuation edge is meaningful only when it has at least one consistent TFS corresponding to the continuation edge. Therefore, the necessary computation is reduced to finding a pair of active and inactive edge TFSs which are consistent or can be unified. It is not necessary to compute the other pairs' unification after finding a first pair unless TFSs representing whole sentence structures are required later. This is made possible by using TFSP links because they can not only unify TFSs immediately and propagate unification result if desired, but they can also propagate information on how to unify them later. This reduces unnecessary unification computation when the edges are not used as parts of the parses of the whole sentences, especially when the TFSP parser does not need to find all possible parses exhaustively.

The unification method used in the TFSP parser has the following characteristics:
(1) It uses Kasper's disjunctive feature structure unification algorithm[6]. This allows not only for efficient descriptions of each lexical item (such as efficient coding of SUBCAT feature values for treating complement order scrambling and word meanings with conditions for disambiguation), but also packing descriptions of homonyms. Disjunctive lexical descriptions work like Polaroid words[3].
(2) As for the definite feature structure unification algorithm, the incremental copy unification algorithm which allows cyclic structures[7] is adopted to treat cyclic constraints including SUBCAT and COH features.

3.2. Agenda Control Mechanism and Plausibility Score

In order to select the most plausible analysis candidate in the early stages, the TFSP parser selects the pending edge with the best edge score among the pending edge list during parsing, and selects the TFS with the best TFS score among sets of TFSs in complete edges, each of which has as its label the start symbol, as its remainder symbol sequence an empty sequence, as its starting vertex the leftmost vertex of the chart, and as its ending vertex the rightmost vertex of the chart just after parsing finishes. Parsing finishes when a certain number of TFSs have been created with scores better than certain criteria determined by the input sentence length (e.g., the number of *bunsetsu* structures).

The edge score mainly contributes to first obtaining a plausible syntactic structure. The edge score for treating speech recognition result lattices is essentially based on the following:
(a) speech recognition score,
(b) surface string length, and
(c) edge type such as active, inactive, or just-proposed.
When a new edge is created, the edge score is calculated from information on the active edge and the inactive edge. Moreover, when a new TFSP link is created and the links point to an existing continuation edge, the edge score of the continuation is recalculated.

The TFS score mainly contributes to obtaining syntactico-semantically and pragmatically plausible structure and is essentially based on the following:
(d) phrase structure complexity (the number of phrase structure tree nodes),
(e) unfilled complements (the number of elements in SLASH feature value), and
(f) violation of pragmatic constraints on expression usage (the unnatural relationships in the PRAG|RESTRS feature value).

The behavior of the TFSP parser is illustrated by an example. Suppose the case where a speech recognition result lattice includes the following sentence candidates and the nominative postposition "*ga*" has a better speech recognition score than the topic marker "*wa*" (Fig.4). The parser first tries to build up the structure including "*ga*" due to the speech recognition score preference because there are no other differences between structures including "*ga*" and "*wa*". However, the building-up process stops when combining structures corresponding to "*tourokuyoushi ga*" and "*o-okuri*" because of TFS unification failure between SEMF feature values of the verb's subject [[animate +]] and the nominative noun phrase [[animate -]]. Then, the parser adopts the structure containing "*wa*" and analyzes the semantics of the topic noun phrase as playing a semantic object role in the "*okuru*" (sending) relationship.

In this case, the agent subject is missing and the parser outputs as the semantic representation:

```
[[relation  okuru-1]
 [agent     ?subject_sem]
 [recipient ?indirect-object_sem]
 [object    [[parameter ?x]
             [restriction [[relation tourokuyoushi-1]
                           [object   ?x]]]]]]
```

However, the parser also outputs pragmatic constraints on the person referred to by the subject based on the lexical descriptions of the honorific verb "*itashi*" as follows:

```
[[relation           condescend]
 [agent              ?speaker]
 [object             ?subject_sem]
 [comparative-object ?indirect-object_sem]]
```

After parsing, the analysis module searches for the person to whom the speaker can condescend, and if there is no person other than the speaker and the hearer in the discourse of the utterance, the missing subject is analyzed as referring to the speaker. Then, the following semantic representation is obtained:

```
[[relation  okuru-1]
 [agent     ?speaker]
 [recipient ?hearer]
 [object    [[parameter ?x]
             [restriction [[relation tourokuyoushi-1]
                           [object   ?x]]]]]]
```

From this semantic representation, the output sentence "I send you a registration form." is obtained.

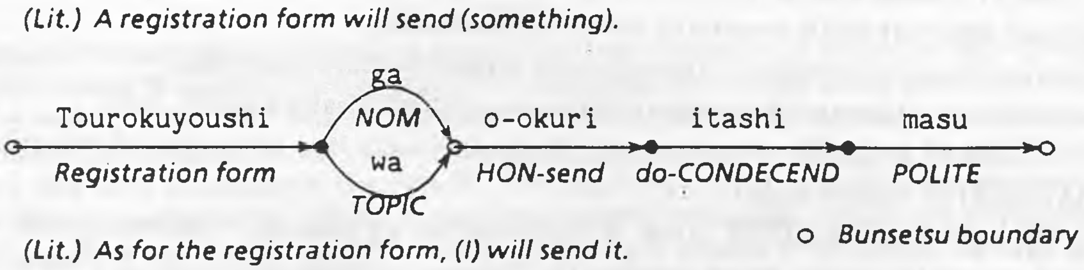


Fig.4 Example of speech recognition result lattice sequence (simplified).

3.3. Experiments

This analysis method is applied to speech recognition results of sentences in 2 task-oriented dialogues about "the secretarial service of the international conference". The HMM-LR speech recognition module with a *bunsetsu* dependency filter outputs for each spoken sentence a sequence of *bunsetsu* phrase lattices. These 2 dialogues consist of 37 sentences. The speech recognition module outputs correct results (i.e., sequences of *bunsetsu* lattices each of which includes the correct *bunsetsu* structure) for 35 sentences. This analysis method is applied to these 35 sentences.

These sentences consists of 76 *bunsetsu* phrases and 112 *bunsetsu* structure candidates. That is, a *bunsetsu* phrase has about 1.47 *bunsetsu* structure candidates.

For this experiment, a grammar was prepared which includes not only lexical items required for accepting correct *bunsetsu* structures in the dialogue, but also all lexical items consisting of all *bunsetsu* structure candidates. The grammar consists of 13 general rules including morphological rules and about 300 lexical entries.

The analysis method obtains correct sentence analysis results for 34 sentences; adequate English sentences are obtained from these correct analysis results. The sentence recognition rate of this method is about 97% and the total sentence recognition rate including the HMM-LR speech recognition module is 92%. The single incorrect analysis result structure, which corresponds to the Japanese sentence "*tourokuyoushi mo o-okuri itashi masu*" (lit. "*I will send you a registration form, too*") instead of "*tourokuyoushi o o-okuri itashi masu*" (lit. "*I will send you a registration form*"), includes as the incorrect speech recognition part only an incorrect modal particle "*mo*" with a higher speech recognition score than the correct case particle "*o*", and the incorrectly recognized structure is perfectly grammatical. In this case, to obtain the correct result requires taking account of the differences in presuppositions derived from these particles and comparing these presuppositions with the context of the utterances.

4. CONCLUSION

In this paper, a new analysis method is proposed for Japanese spoken sentences using a grammar framework for treating spoken-style Japanese sentences and a new parser called the TFSP parser. The grammar framework is essentially based on HPSG and JPSG, and is designed to treat not only syntactic and semantic information but also pragmatic information. Analysis results based on this framework include semantic interpretations of input sentences with annotations on constraints on the uses of these sentences. The TFSP parser has been developed to allow edge structure sharing in unification-based analyses. This method is used as the analysis module of the NADINE system and the SL-TRANS system.

The analysis method is applied to HMM-LR speech recognition result lattices. In parsing lattices, selecting the pending edge with the best score allows the parser to first find plausible candidates. Constraints described in TFSs filter out syntactically or semantically ill-formed structures. The experimental results show that this method is effective in sentence speech recognition. In the experiments, recovering from incorrect recognition requires utterance context understanding including understanding of utterance presuppositions.

ACKNOWLEDGEMENTS

The author is deeply grateful to Dr. Kurematsu, the president of ATR Interpreting Telephony Research Laboratories, Teruaki Aizawa, the head of the Natural Language Understanding

Department, Kei Yoshimoto, and the members of the Natural Language Understanding Department for their constant help and encouragement.

REFERENCE

[1] Earley J.: *An Efficient Context-Free Parsing Algorithm*, Comm. ACM, Vol. 13, No. 2, 1970.

[2] Gunji, T.: *Japanese Phrase Structure Grammar - A Unification-Based Approach*, Dordrecht, Reidel, 1987.

[3] Hirst, G.: *Semantic interpretation and the resolution of ambiguity*, Cambridge University Press, 1987.

[4] Iida, H. et al.: *An Experimental Spoken Natural Dialogue Translation System Using a Lexicon-Driven Grammar*, Proceedings of the European Conference on Speech Communication and Technology, 1989 (to be published).

[5] Kakigahara, K. and Morimoto, T.: *A Study of Bunsetsu Selection Based on the Kakariuke-Dependency*, (in Japanese), IPSG Spring Meeting, 1989.

[6] Kasper, R.: *A Unification Method for Disjunctive Feature Descriptions*, Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics, 1987.

[7] Kato, S. and Kogure, K.: *Efficiency of Feature Structure Unification Methods*, (in Japanese) Proceedings of the Natural Language Working Group of IPSJ, NL64-9, 1987.

[8] Kita, K. et al.: *Parsing Continuous Speech by HMM-LR Method*, Proceedings of the international Workshop on Parsing Technologies, 1989.

[9] Kogure, K. et al.: *A Method of Analyzing Japanese Speech Act Types*, Proceedings of the 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, 1988.

[10] Pollard, C. and Sag, I.: *Information-Based Syntax and Semantics - Volume 1 Fundamentals*, CSLI Lecture Notes, No. 13, 1988.

[11] Winograd, T.: *Language as a cognitive process - Volume 1 Syntax*, Addison-Wesley, 1983.

[12] Yoshimoto, K. and Kogure, K.: *Phrase Structure Grammar for Inter-Terminal Dialog Analysis*, (in Japanese), IPSG Fall Meeting, 1988.

PROBABILISTIC METHODS IN DEPENDENCY GRAMMAR PARSING

Job M. van Zuijlen

BSO/Research
P.O.B. 8348
NL-3503 RH Utrecht
The Netherlands
e-mail: zuijlen@dlt1.uucp

June 1989

ABSTRACT

Authentic text as found in corpora cannot be described completely by a formal system, such as a set of grammar rules. As robust parsing is a prerequisite for any practical natural language processing system, there is certainly a need for techniques that go beyond merely formal approaches. Various possibilities, such as the use of simulated annealing, have been proposed recently and we have looked at their suitability for the parse process of the DLT machine translation system, which will use a large structured bilingual corpus as its main linguistic knowledge source. Our findings are that parsing is not the type of task that should be tackled solely through simulated annealing or similar stochastic optimization techniques but that a controlled application of probabilistic methods is essential for the performance of a corpus-based parser. On the basis of our explorative research we have planned a number of small-scale implementations in the near future.

1. Introduction

Usually a parser is viewed as a program that takes a sentence in a particular language as its input and delivers one or more analyses for that sentence. This is no different in the present prototype of DLT (Distributed Language Translation), a multilingual translation system under development at the Dutch software house BSO. In the prototype, we use an ATN-parser that delivers all syntactic analyses of an input sentence in the source language (SL). Each analysis undergoes structural and lexical transfer resulting in one or more target language (TL) trees.[1]

In order to limit the size of the ATN, we have used Technical English as the basis for our grammar. This type of English has been specially designed for writing technical manuals. It has certain limitations, such as the number of verb forms to be used, the number of elements that may be coordinated, sentence length and the like. Nevertheless, it proves to be very difficult to specify a complete grammar, let alone formulate grammar rules. Moreover, even with such a limited grammar we have to deal with the combinatorial explosion due to the parsing of ambiguous sentences.

[1] In fact, DLT consists of two separate but similar translation processes. The first translates the SL into the IL, DLT's Esperanto-based Intermediate Language; the second translates from the IL into the TL.

A typical complication of a translation system is that, apart from the SL grammar for the parser, we need a grammar for TL generation and a contrastive grammar (metataxis) to link source and target language. Then, there are three dictionaries, one for each language and one for the language pair. Finally, semantic information has to be included. On a prototype scale, it is already difficult to maintain consistency between the various knowledge sources, but for a large-scale industrial version this is almost impossible.

Two recent inventions by members of the DLT research team have contributed to the solution of the complications mentioned previously. Van Zuijlen (1988) has introduced the **Structured Syntactic Network** (SSN) to achieve the compact representation of all dependency-type analyses of a sentence in a single structure. The problem of consistency of knowledge sources has been tackled by Sadler (1989), who has proposed the **Bilingual Knowledge Bank** (BKB), a large structured bilingual corpus. It contains for each sentence the preferred syntactic analysis and translation in the given context, as well as certain other referential and co-referential information. An important structural element is the Translation Unit (TU), a dependency subtree for which there is a non-compositional translation, e.g. expressions like *kick the bucket.*

The introduction of the BKB places the various processes commonly found in a translation system (parsing, structural transfer, semantic evaluation, generation) in a different perspective. We will not deal here with structural transfer and generation but concentrate on the consequences for the parse process, which will be dealt with in a number of sections:

- linguistic theory and representation;
- interfacing parser and BKB;
- corpus-based parsing;
- probabilistic methods.

We conclude with a few remarks about research we have planned for the near future.

2. Linguistic Theory and Representation

The linguistic theory used in DLT is Dependency Grammar, one of the less frequently used formalisms in natural language processing projects (see Schubert (1987) for a discussion on its suitability for machine translation). The dependency grammar of a language describes syntactic relations or **dependencies** between pairs of words. The relation is directed, i.e. one word, the **governor** governs (dominates) the other, the **dependent**. In general, the dependencies range over word classes (syntactic categories) rather than specific words. A useful feature of dependency grammar is that the resulting analysis may be used directly by the semantic component of the translation system, i.e. a single type of representation suffices for all processes in the system.

The syntactic relations in dependency grammar are derived from the **function** of a word in the sentence. For example, *man* is the subject of *walks* in *The man walks*. It is important to realize that dependency grammar is primarily concerned with words; there are no phrasal categories.

A dependency tree has a geometry that is quite different from that of a constituent tree (Figure 1). Notice that in a constituent tree nodes are either phrasal or lexical, but that in a dependency tree nodes are always lexical. The branches of a dependency tree are labeled with syntactic relations. A dependency tree is not ordered, which means that a particular relation is only defined by the governor and the dependent and not by the position of the dependent with respect to other dependents. In the example word order does play a role to identify the subject and the object of the sentence but order is not reflected in the representation.

In order to facilitate the interfacing between the BKB and the parse process (see Section 3), we use an alternative representation, which we will refer to as a **Dependency Link**

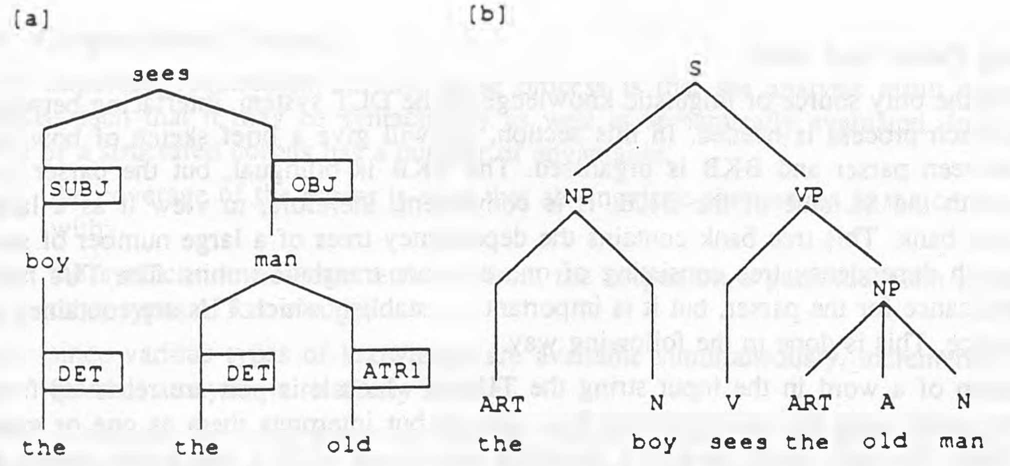


Figure 1. [a] dependency tree and [b] constituent tree for the sentence *The boy sees the old man.*

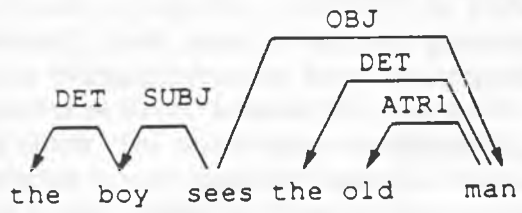


Figure 2. The dependency link representation of *The boy sees the old man.*

Representation (DLR). A dependency link consists of a governor, a dependent and their relation. The link is projective, i.e. it takes the position of governor and dependent with respect to each other into account. We obtain a graphical representation of a DLR by writing down the sentence as a linear string of words and then draw the dependencies as arcs (Dependency Links) connecting the words. Figure 2 shows the dependency link representation of *The boy sees the old man.*

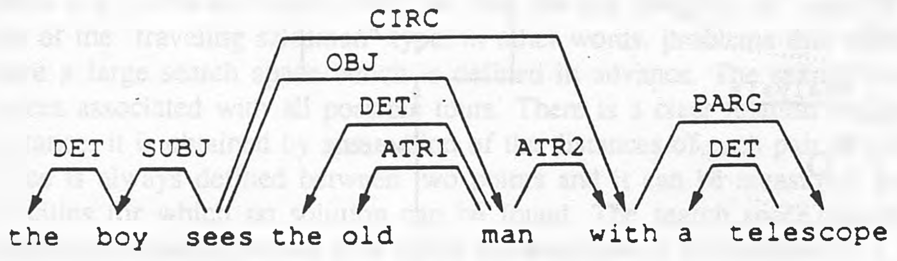


Figure 3. The dependency link representation of *The boy sees the old man with a telescope.*

The DLR shown in Figure 2 has the same representative power as a dependency tree. However, in contrast to a tree, connections in a DLR are by reference and, as a consequence, it is possible to represent directed graphs as well. Graphs are a means to represent multiple analyses of a sentence in a single representation. The ideas behind such a representation for dependency grammar, the SSN, are discussed in Van Zuijlen (1988). The dependency link may be viewed as a common building brick for trees as well as SSNs. This is shown in Figure 3 where we see the two analyses for *The boy sees the old man with a telescope* in a single DLR. By selecting either the link *man*-ATR2-*with* or *sees*-CIRC-*with* we obtain the respective interpretations. The set of dependency links that constitute one interpretation is called an ensemble.

3. Interfacing Parser and BKB

As the BKB is the only source of linguistic knowledge in the DLT system, interfacing between the BKB and each process is needed. In this section, we will give a brief sketch of how the interfacing between parser and BKB is organized. The BKB is bilingual, but the parser has only to deal with the SL side of the BKB. It is convenient, therefore, to view it as a large dependency tree bank. This tree bank contains the dependency trees of a large number of sentences, with each dependency tree consisting of one or more translation units. The TUs have no direct significance for the parser, but it is important to establish which TUs are contained in the input sentence. This is done in the following way.

After recognition of a word in the input string the TUs of which it is part are retrieved from the BKB. The parser does not deal with the TUs directly but interprets them as one or more dependency links. For each word there is a (possibly empty) set of DLs that either govern or depend on the word. By combining DLs into ensembles we obtain dependency trees the **projection** (linearization) of which has to match the input string. So parsing is not carried out by parse tree construction guided by the input string but by matching the input string with the projection of a parse tree synthesized from dependency links (Figure 4).

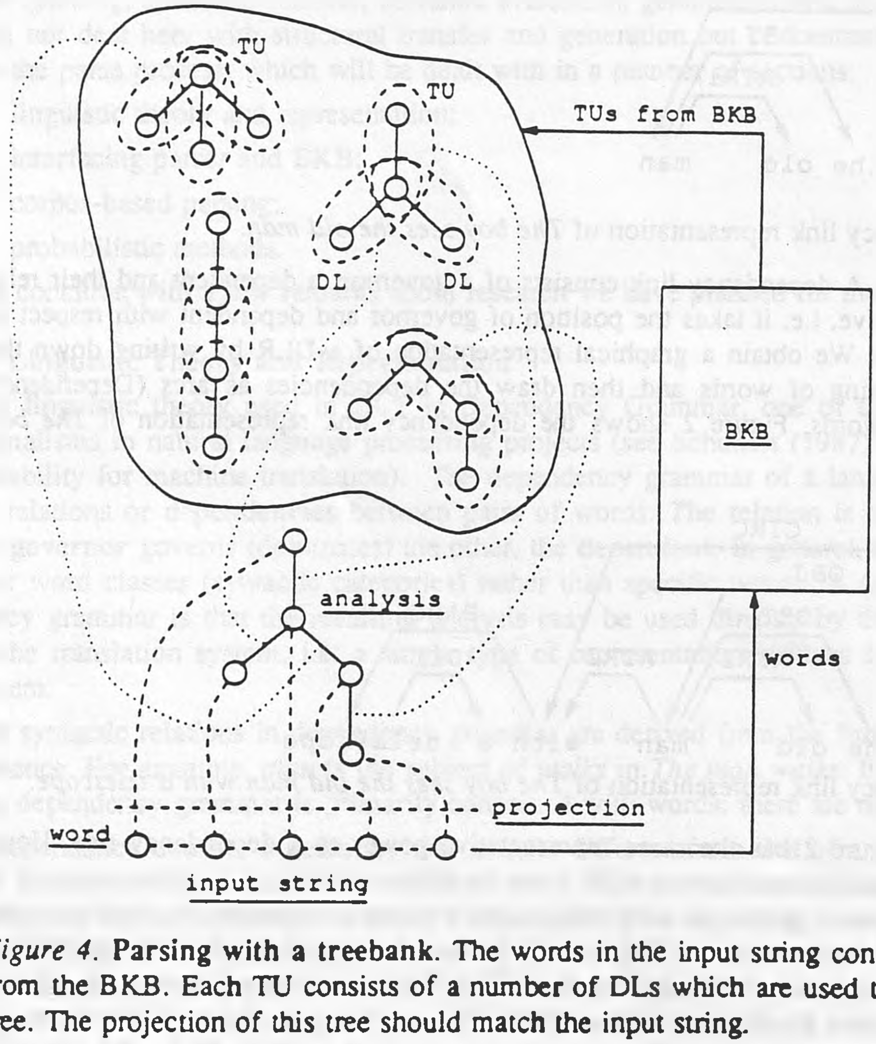


Figure 4. Parsing with a treebank. The words in the input string control the retrieval of TUs from the BKB. Each TU consists of a number of DLs which are used to synthesize an analysis tree. The projection of this tree should match the input string.

The dependency links that are "used" for the analysis (in Figure 4 connected with the analysis tree by dotted lines) select in turn those parts of the TUs retrieved from the BKB that are relevant for the translation of the input string.

4. Corpus-Based Parsing

An important requirement for the parse process is that the analysis result matches with the BKB, such that it may be syntactically as well as semantically evaluated. In that respect the use of a structured corpus has a number of advantages.

(1) the coverage of the parser is such that all linguistic phenomena in the corpus will be dealt with;

(2) the syntactic knowledge retrieved from the corpus on a particular item is consistent with other types of knowledge;

(3) since various types of knowledge are available simultaneously, incremental evaluation of (partial) analyses is relatively simple.

This is evident for input sentences that are literally present in the BKB and for which – in a manner of speaking – direct pattern matching is possible. However, we want to extend the coverage beyond that and, therefore, we have done some explorative research in the field of corpus-based parsing, primarily by reviewing work of others in the light of our specific needs.

Recent work in corpus-based parsing has a common characteristic. A parsed corpus is used as a source of linguistic knowledge and probabilistic methods are used to arrive at an analysis. Basically, parse trees are randomly generated until the optimal parse tree is found with respect to an evaluation measure based on comparison of the parse tree with the corpus. Robustness is guaranteed since, whatever the value of the evaluation, one of the analyses will be better than all others. The search space associated with the investigation of all possible parse trees for a sentence is very large and, therefore, Haigh, Sampson & Atwell (1988) apply simulated annealing in their Annealing Parser for Realistic Input Language (APRIL) as an efficient way to find this optimal parse tree for a complete sentence. Atwell, O'Donoghue & Souter (1989) have developed the Realistic Annealing Parser (RAP) which also uses simulated annealing but works incrementally, thus reducing the search space drastically. Both projects evaluate the resulting trees with corpus information, either in the form of a tree bank (Haigh et al. 1988) or first-order recursive Markov chains (Atwell et al. 1989).

Comparing APRIL and RAP shows that a slightly different approach to the same problem already results in a large reduction of the search space. This justifies the question whether simulated annealing is really a very suitable technique. If we examine the literature on that point (e.g. Aarts and Korst 1989) we find that the problems for which it is successfully applied are of the "traveling salesman" type, in other words, problems that are highly unstructured and have a large search space which is defined in advance. The search space consists of the distances associated with all possible tours. There is a clear relation between a tour and the total distance; it is obtained by summation of the distances of each pair of connected cities. The distance is always defined between two points and it can be measured; there is no configuration of cities for which no solution can be found. The search space may become very large and simulated annealing serves as a means to investigate it efficiently.

At first sight, parsing a language seems to be a similar problem. We have a number of words (cities) and, in the case of a dependency grammar representation, we have to find optimal connections between them. For each connected pair of words we compute the grammaticality of the connection (distance) by comparing it with the linguistic information we have available. Here the problem starts. The "syntactic distance" cannot be calculated straightforwardly but has to be approximated on a probabilistic basis, e.g. by counting the number of occurrences of the particular relation in a corpus. If the relation never occurs it is not possible to say anything sensible about the distance. We might assign a default value to it, but we have no certainty that it contributes to an optimal solution. This in contrast with the "traveling salesman" problem where a long distance between two points does not exclude the connection from being part of the optimal solution.

The temporary acceptance of "odd" constructions in simulated annealing parsers is motivated by the fact that during the search of a new solution the current solution is changed by means of a number of primitive modifications which may lead to intermediary results which are not well-formed. The acceptance of these results doesn't depend on their leading to a solution which may be evaluated by comparing it with the linguistic information available but on a stochastic function that states the probability with which a "bad" result is to be accepted. What is missing is the observation that language is structured and enables predictions on the basis of available partial information. So instead of a random walk (or unguided city tour) it is possible to select those transformations that are most likely to lead to an optimal solution.[2]

A corpus is very useful to make such predictions and if we intend to use the same corpus for the evaluation of the solutions we have the certainty that we only generate those solutions that are verifiable.

Again we may observe a difference with the "traveling salesman" problem. The latter has a predefined solution space and it is easy to specify primitive transformations that will lead from one solution to the other. In the case of parsing the solution space is not predefined but has to be generated on the basis of the linguistic information available. This is either a set of grammar rules or a tree bank based on a parsed corpus.

Souter (1989) discusses how difficult it is to express the grammatical information contained in a such corpus in a limited number of rules. In fact, thousands of rules are needed, many of which are only applied once or twice. He observes a close resemblance between a rule-frequency curve and the more familiar word-frequency curve (Zipf 1936). These findings support the idea that the usual grammar with a few hundred rules is not very adequate and may contain "gaps". Also, our experience with the DLT prototype has made clear to us that a rule-based approach has unacceptable limitations. Still, we are not convinced that it is necessary to apply statistical optimization all the time when a corpus is used to find the correct analysis. When dealing with input that is covered by the corpus the latter may be viewed as large set of rules and a solution will be found in a straightforward, efficient manner. Nevertheless, there is room for probabilistic methods and in the next section we will discuss some applications.

5. Probabilistic Methods

It should be clear from the discussion in the previous section that probabilism is only useful when it is applied in a controlled way. For the parse process in a BKB-based DLT system there are three application areas:

- handling input errors and unusual input;
- restricting the number of analyses;
- ordering of alternatives.

We will discuss each of these areas in the following subsections.

5.1. Incorrect and Unusual Input

As far as the parser is concerned incorrect and unusual input relate to input for which no acceptable solution can be found by straightforward matching with the BKB. The main difference is that if the input is incorrect the user should be consulted for clarification. If the input is unusual a solution should preferably be found without asking. The border between the two is

[2] In RAP (Atwell et al. 1989) the rate of convergence is improved by introducing a bias towards the transformation of low-valued parts of the tree.

determined by the fact whether it is possible to find a **single** analysis that matches with the BKB.

The ability to process deviant input is a requirement for any robust parser. In RAP and APRIL this is achieved by always generating a parse tree, even if the result is implausible. For our application this will not do. Each analysis should match with the BKB, otherwise translation is not possible. If such an analysis cannot be obtained the parser should try and find out what is wrong and, if necessary, consult the user – preferably by making some sensible suggestions.

5.1.1. Input Errors

Input errors may be of various types which ask for different approaches. However, a general principle is that we need to know what the "correct" version is in order to say something sensible about the deviations. This is a severe requirement, but if an error has only local consequences and if there is enough surrounding context it should be possible to determine the cause of the deviation.

Since error analysis may need a combined effort of different knowledge sources, the BKB approach seems to be ideal for intelligent error handling. Some types of errors we may consider are:

(1) word form errors;

(2) syntactic deviations;

(3) spelling mistakes.

Errors of type (1) or (2) are relatively easy to detect by comparing the input to the linguistic information available. An interesting method to deal with such grammatical errors has been suggested by Charniak (1983). In a rule-based parser a rule for which one or more atomic tests (e.g. agreement) fail is not applied. By modifying the tests it is possible to assign a kind of applicability measure to a rule. Instead of returning simply "yes" or "no" each test returns a value that is added to the current value of the applicability measure if the test succeeds and subtracted if the test fails.

Charniak's proposal is also very useful when a grammar is based on a corpus. For instance, it could be that, considering their word class, two words have a relation but that there is a mismatch between their features. An example is *The boy see the man*, in which subject-verb agreement is violated. However, by establishing that *the boy* could be a subject and that *see* takes one and that complete feature unification is not possible the parser classifies the error. The user will then be consulted for clarification, e.g. by being presented two correct alternatives one of which must be chosen:

(a) *The boys see the old man*

(b) *The boy sees the old man*

By using corpus information a likelihood value could be assigned to each alternative, which may be decisive if one alternative turns out to be far more plausible than any of the others, in which case user consultation is not needed.

There are errors that cannot be described on the basis of features or syntactic structures, but may be solved by using knowledge on individual words or their relations. In such cases a corpus-based system is superior. A typical example is a misspelled word, such as *foz*, which might be *fez* or *fox*. By taking the context into account and comparing it with corpus information the selection of one or the other alternative is supported. Compare:

(a) *In Morocco men wear a caftan and a foz.*

(b) *The foz hunts at night.*

The context in (a) points to the interpretation *fez*, whereas the context in (b) points to the interpretation *fox*.

5.1.2. Unusual Input

In this section we will show by means of a simple example how use of a corpus supports the handling of unusual input. We mentioned earlier that in dependency grammar dependencies range over word classes. There are cases, however, in which a word has a syntactic function that is not typical for its word class. Nouns, such as *week*, *month* and *year*, may be used as time adverbials, as in *I saw him last week*. We don't want to call *week* an adverb because it cannot perform the same functions as an adverb. On the other hand, we don't want to extend the functions that are possible for nouns because only a small number of nouns may be used in the same way as *week*.

In a rule-based parser categories are used to formulate some general distributional criteria, as it is not feasible to state for a each word the syntactic functions it may perform. Such information is, however, available in a corpus. We may find:

(1) *He came last week.*

(2) *I have had a very bad week.*

(3) *A week is enough to finish this job.*

From the available parse trees we derive the distribution of *week* in terms of governing or depending relations. Now suppose that we have the input sentence *He arrives next month*, but that we don't have direct evidence that *month* could perform the same function as *week* in (1). The parser will then compare the distribution of *month* and *week*, in order to establish if they are used in the same way, i.e. show **syntactic synonymity**. The more correspondence is found, the higher the probability that *month* may indeed be used as a time adverbial.

The method to establish the possibility for *month* to be used as time adverbial may also be applied in other cases. The syntactic context of a word may suggest a function or even word class for which there is no direct evidence. For example, in *He computers all the time* the noun *computer* is used as verb. From the corpus we may deduce that in English "any noun may be verbed" and that the use of *computer* as a verb is acceptable.

5.2. Restricting the Number of Alternatives

An exhaustive parser often generates alternatives without taking aspects of language use into account. For a system that features user interaction this results in asking the user questions about alternatives that are counter-intuitive. Consider, for example,

Daily inspections should be performed.

Here *daily* modifies *inspections* and although it could modify the verb in an alternative analysis, this interpretation is only evident when *daily* is placed at the end of the sentence:

Inspections should be performed daily.

This is an example in which a corpus could be used to limit the number of possible analyses and, thus, assist the system to behave sensibly in the eyes of the user.

The fact that the corpus sometimes extends and sometimes restricts the number of possible interpretations indicates that there is an important lexical influence in syntax which causes words to behave differently from what we expect, considering their word classes. This suggests that a strict separation between syntax and semantics (or at least language use) is not possible in the case of "realistic" language. The acceptability of certain distributions cannot be explained syntactically; there is no reason why only specific nouns may serve as adverbials. By the same token, there is no reason to exclude some potential analyses other than by observing that a language user would never interpret them that way.

5.3. Ordering Alternatives

An interactive translation system will have to deal with alternative analyses of the SL sentence, even if some of them may be excluded in advance. Particularly in the case of coordination or post-modifier sequences there may by a number of alternatives that have to be taken into account. By using the graph representation we introduced in Section 2 it is possible to represent the alternatives in a compact way. There are various techniques to prevent the combinatorial explosion caused by the generation of the alternatives (see e.g. Tomita 1985), but then we are faced with the problem of evaluating them efficiently. We intend to solve this in the following way.

We start with the incremental generation of all dependency links that are part of one or more of the potential analyses, resulting in a DLR of the input. The DLs that constitute the best analysis according to a given evaluation function are made **active**, all others are made **dormant**. If the multiple analyses are caused by structural ambiguity, such as alternative attachment points, then a simple transformation suffices to generate an alternative analysis. In Figure 3, for example, the activation of DL *man*-ATR2-*with* and the deactivation of DL *sees*-CIRC-*with* or vice versa results in an alternative analysis. So, a transformation is performed by activating/deactivating of a pair of DLs with a common dependent.

The set of DLs with a common dependent forms a **choice point**. Only DLs that are elements of choice points will have to be considered in the search for alternatives. To order the alternatives, that is to find the second best given the current optimum, it may be necessary to perform more than one transformation without knowing what the sequence of transformations is. If there is a large number of choice points, systematic evaluation of all analyses is not feasible and a stochastic optimization technique is necessary. In contrast with the parsing of arbitrary input, such a technique is applicable here since certain requirements are met (Aarts & Korst 1989: 100). The solution space (i.e. a representation of all possible solutions) is given by the DLR and there is a primitive transformation (the activation/deactivation of a pair of DLs) to generate an alternative solution. All the same, in very simple cases it is better to evaluate and compare alternatives directly. In view of this, it is advantageous to have an adaptive optimization technique that is able to select the most efficient strategy.

6. Future Work

The result of our explorative research has been that we see many interesting aspects in corpus-based parsing in connection with probabilistic methods. However, application in a BKB-based DLT system asks for an approach that is different from related proposals by others. Therefore, we have planned a number of small-scale implementations in order to find out to what extent the various ideas and suggestions put forward in this paper are indeed feasible.

Acknowledgements

I would like to thank my colleagues from BSO/Research, in particular Dorine Tamis and Ronald Vendelmans, for fruitful discussions and critical remarks.

References

Aarts, Emile and Jan Korst (1989): *Simulated Annealing and Boltzmann Machines*. John Wiley and Sons, Chichester.

Atwell, Eric, Tim O'Donoghue and Clive Souter (1989): "The COMMUNAL RAP: A Probabilistic Approach to NL Parsing". Leeds University, Leeds.

Charniak, Eugene (1983): "A Parser with Something for Everyone". In: King, Margaret (ed.), *Parsing Natural Language*. Academic Press, London.

Haigh, Robin, Geoffrey Sampson and Eric Atwell (1988): "Project APRIL - a Progress Report". *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*. Buffalo.

Sadler, Victor (1989): *The Bilingual Knowledge Bank*. BSO/Research, Utrecht.

Schubert, Klaus (1987): *Metataxis*. (= Distributed Language Translation 2). Foris, Dordrecht.

Souter, Clive (1989): "The COMMUNAL Project: Extracting a Grammar from the Polytechnic of Wales Corpus". *ICAME Journal*, No. 13, April 1988.

Tomita, Masaru (1985): *Efficient Parsing for Natural Language*. Kluwer, Boston.

Van Zuijlen, Job M. (1988): "A Technique for the Compact Representation of Multiple Analyses in Dependency Grammar". BSO/Research, Utrecht.

Zipf, George (1936): *The Psycho-Biology of Language: An Introduction to Dynamic Philology*. George Routledge, London.

PREDICTIVE NORMAL FORMS FOR FUNCTION COMPOSITION IN CATEGORIAL GRAMMARS

Robert E. Wall, University of Texas at Austin
and
Kent Wittenburg, MCC

Abstract: Extensions to Categorial Grammars proposed to account for nonconstitutent conjunction and long-distance dependencies introduce the problem of equivalent derivations, an issue we have characterized as spurious ambiguity from the parsing perspective. In Wittenburg (1987) a proposal was made for compiling Categorial Grammars into predictive forms in order to solve the spurious ambiguity problem. This paper investigates formal properties of grammars that use predictive versions of function composition. Among our results are (1) that grammars with predictive composition are in general equivalent to the originals if and only if a restriction on predictive rules is applied, (2) that modulo this restriction, the predictive grammars have indeed eliminated the problem of spurious ambiguity, and (3) that the issue of equivalence is decidable, i.e., for any particular grammar, whether one needs to apply the restriction or not to ensure equivalence is a decidable question.

1. **Introduction.** Steedman (1985, 1987), Dowty (1987), Moortgat (1988), Morrill (1988), and others have proposed that Categorial Grammar, a theory of syntax in which grammatical categories are viewed as functions, be generalized in order to analyze "noncanonical" syntactic constructions such as wh-extraction and nonconstituent conjunction. A consequence of these augmentations is an explosion of semantically equivalent derivations admitted by the grammar, a problem we have characterized as spurious ambiguity from the parsing perspective (Wittenburg 1986). In Wittenburg (1987), it was suggested that the offending rules of these grammars could take an alternate predictive form that would eliminate the problem of spurious ambiguity. This approach, consisting of compiling grammars into forms more suitable for parsing, is within the tradition of discovering normal forms for phrase structure grammars, and thus our title. Our approach stands in contrast to those which are attempting to address the spurious ambiguity problem in Categorial Grammars through the parsing algorithm itself rather than through the grammar (see Pareschi and Steedman 1987; Moortgat 1987, 1988; Hepple and Morrill 1989; Koenig 1989; Gardent and Bes 1989). Our approach is more in line with the tack that Bouma (1989) is taking, although his formulation of categorial systems differs radically from our own, more traditional set of assumptions.

In Wittenburg (1987) it was conjectured that predictive forms for Categorial Grammars were equivalent to the source forms and that they did indeed eliminate spurious ambiguity. Here we report on formal results that have ensued from these original conjectures. We have found that, on the whole, the conjectures proved valid although we have discovered that the relationship between predictive normal forms for these grammars and their source forms are more complicated than was implied by the earlier paper. As we will show, an additional condition is necessary to ensure equivalence of these grammars and eliminate spurious ambiguity from the picture.

2. **Source Grammar (G)** In this paper we focus on the role of basic function composition as a way of illustrating the effects of predictive normal forms. For these proofs then, we assume a form of Categorial Grammar that is considerably more restricted than those advocated by van Bentham (1986), Steedman (1987), Moortgat (1988), Morrill (1988), and others. As the work of these authors shows, the simple Categorial Grammars we assume here are not linguistically adequate. We do not consider the effects of type-raising nor of generalized conjunction here, nor do we address the issue of generalized composition. While we intend to address these points in future work, the simplifications we assume here allow us to uncover an intitial set of properties associated with the use of predictive combinators.

We assume for our source grammar G the following combinatory rules together with a lexically assigned system of categories of the usual recursive sort. That is, we assume a set of basic categories, say, {S, NP, N}. If X and Y are categories, so are X/Y and Y\X. Our notation follows Steedman (1987) and Dowty (1985) in that the domain type appears consistently to the right of a slash and a range type to the left. Left directionality is then indicated by a left-leaning slash, and right directionality by a right-leaning slash. Semantically, we assume that lexical categories introduce functional constants in lambda terms where the arity of the functions bears an obvious and direct relation to the syntactic type.[1] Here are example lexical entries.

kicks: S\NP/NP	John: S/(S\NP)	a: NP/N	platypus: N
$\lambda x \lambda y$ ((**kicks** x) y)	λf(f **john**)	λx(**a** x)	**platypus**

We assume the following set of combinatory rules:

Forward function application (fa>)   Backward function application (fa<)

```
X/Y  Y -> X                 Y  X\Y -> X
 f   a   f(a)               a   f     f(a)
```

Forward function composition (fc>)   Backward function composition (fc<)

```
X/Y  Y/Z  ->  X/Z                     Y\Z  X\Y  ->  X\Z
 f    g    λx(f(g(x))) = Bfg           g    f    λx(f(g(x))) = Bfg
```

Given these semantics, G yields equivalence classes of derivations, where equivalence is defined modulo β-conversion of semantic terms.[2] The two sources of spurious ambiguity in G are summarized by the following equivalences generalized over directional variants of the rules:

[1] Although we use the term semantics here to describe the relevant issues of derivational ambiguity, it should be understood that we dealing with a syntactic domain. One might think of our semantics as defining the syntactic structures yielded by derivations using these grammars.

[2] This definition of equivalence does not take quantifier scope differences into account. It is more in harmony with the predictive normalization techniques to assume that scoping structure is not necessarily isomorphic to the derivation tree, a position also advocated by Steedman (1987) and Moortgat (1988).

(apply (compose X Y) Z) = (apply X (apply Y Z))

(compose X (compose Y Z)) = (compose (compose X Y) Z)

An example illustrating the first of these equivalences follows:[1]

```
        S                                  S
      f(g(a))                            f(g(a))
-------------------------fa>      -----------------------fa>
      S/NP                               FVP
     λx(f(g(x)))                         g(a)
-----------------fc>                 --------------fa>
S/FVP    FVP/NP     NP          S/FVP    FVP/NP    NP
 f          g        a            f        g         a
```

Assuming the terminal string "John kicks a platypus", complete derivations would yield the equivalent derivational terms ((**kicks** (**a platypus**))**John**).

The numbers of these equivalent derivations increase "almost exponentially" in string length, with the Catalan series (Wittenburg 1986).

3. **Predictive Normal Form** (G') A predictive normal form version of G replaces each composition rule with two predictive variants.[2]

Forward-predictive forward function composition (fpfc>)

```
X/(Y/Z)  Y/W ->  X/(W/Z)
   f       g       λh(f(Bgh)) = λh(f(λx(g(h(x)))))
```

Backward-predictive forward function composition (bpfc>)

```
W/Z  X\(Y/Z)  ->  X\(Y/W)
 g       f          λh(f(Bhg))= λh(f(λx(h(g(x)))))
```

Backward-predictive backwards function composition (bpfc<)

```
Y\W   X\(Y\Z)  ->  X\(W\Z)
 g       f          λh(f(Bgh)) = λh(f(λx(g(h(x)))))
```

Forward-predictive backwards function composition (fpfc<)

```
X/(Y\Z)   W\Z  ->  X/(Y\W)
   f        g       λh(f(Bhg))= λh(f(λx(h(g(x)))))
```

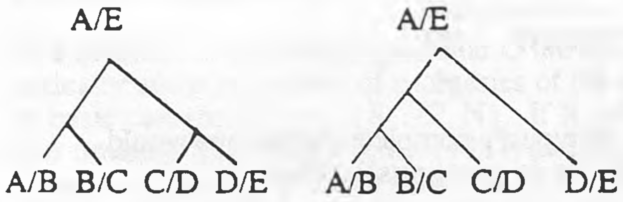
[1]FVP is used as a notational convenience for the category S\NP.

[2]These rules are derivable in the Lambek calculus (Lambek 1958).

We will now consider, first, the question of ambiguity in G'. Second, we will take up the question of whether G and G' are equivalent.

4. **Ambiguity in G'** Is there ambiguity in G'? We will consider first cases that are analogous to the derivations in G known to give rise to spurious ambiguity. Our proof is by induction on the height of a derivation tree.

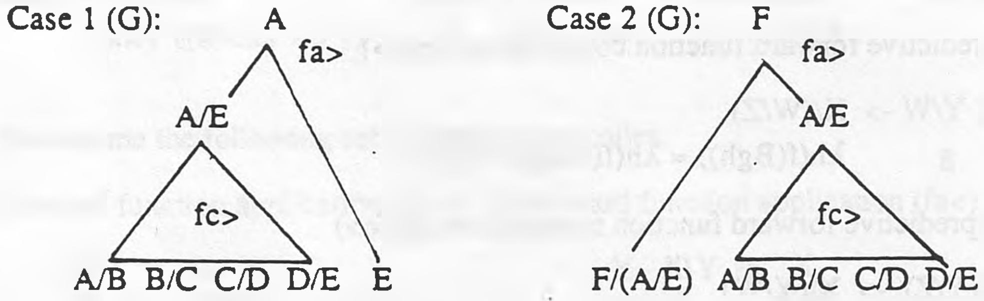
In G, spurious ambiguity arises from the use of composition. Consider any maximal subtree of fc> in a derivation in G, i.e.,



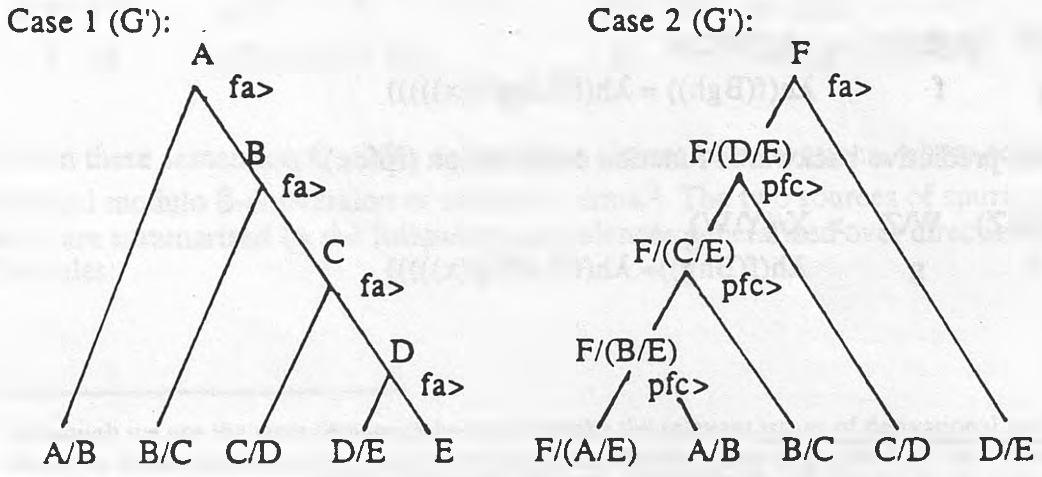
etc.

Since it is part of a derivation of S, it must feed into an instance of fa at the top (either as functor or as argument) -- if it fed into fc, this tree would not be a maximal fc tree.

So subderivations in G with fc> must be of one of the following forms:



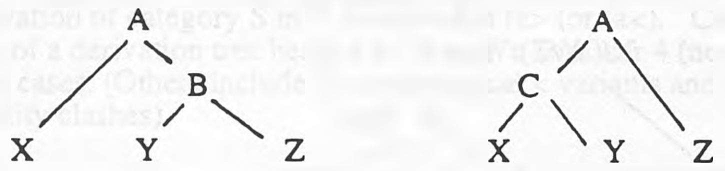
In either case, there is one and only one derivation in G' for the same category sequence.



The cases of fc< are parallel. And since fc> and fc< cannot appear together in a maximal fc tree because of directionality clash, all cases are accounted for.

We have shown here that cases of spurious ambiguity in G do not give rise to analogous spurious ambiguity in G', but of course there may be new sources of ambiguity in G' that we have not yet considered.

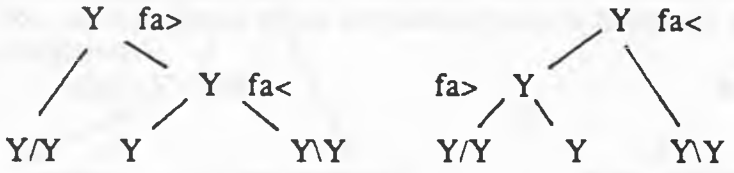
Can there be any cases of derivational ambiguity in G'? That is, can there be derivation trees of the form



for (possibly complex) categories A, B, C, X, Y, Z, where mothers are derived from daughters using just the rules of fa and predictive function composition? An exhaustive list of all the combinatory possibilities reveals just two types:

<u>Type I</u>: X = Y/Y and Z = Y\Y

The central category Y can combine first by fa with Y/Y to its left or with Y\Y to its right, to yield Y in either case. This Y can then combine with the remaining category by fa to give Y again:

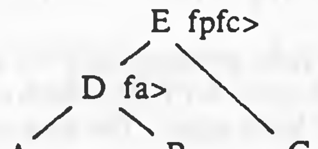


But this is a genuine ambiguity, not a spurious one, for the topmost Y can be assigned different semantic values by the two derivations. If [[Y/Y]] = f, [[Y]] = a, and [[Y\Y]] = g, the left derivation yields f(g(a)) and the right one g(f(a)).

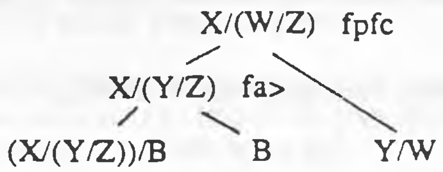
In the more general case, we might have <u>m</u> instances of Y/Y to the left of the Y and <u>n</u> instances of Y\Y to the right. In such a situation the number of syntactically and semantically distinct derivations would be the (m+n)th Catalan number. And since only fa> and fa< are used, the same ambiguity, if it is present, will be found in both G and in G'.

<u>Type II</u>: A predictive combination rule is involved in the derivation. We will illustrate with just one case; the others are similar, differing only the directions of the slashes and the order of constituents.

Consider the derivation tree

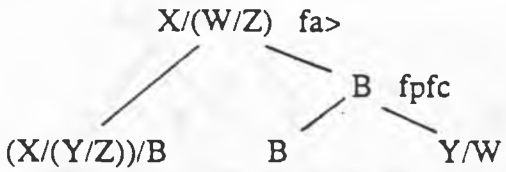


in which each mother node is derived from its daughters by the indicated rule. Since E is derived by fpfc>, D must be of the form X/(Y/Z) and C of the form Y/W; hence E is of the form X/(W/Z). Then because D is derived by fa>, it follows that A must be of the form (X/(Y/Z))/B. That is, the derivation tree is of the form

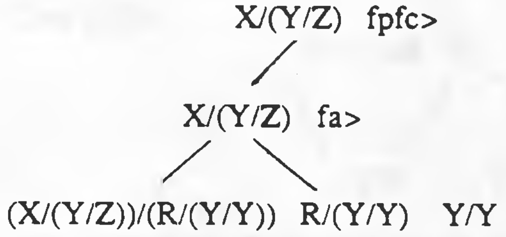


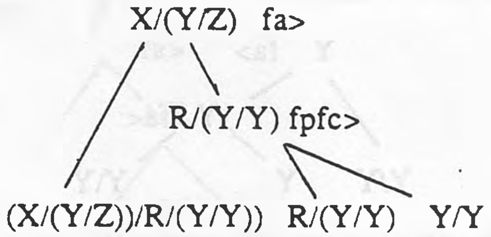
for (possibly complex) categories B, W, X, Y, Z.

Given the rules of fa and predictive composition, there is a distinct derivation tree yielding X/(W/Z) from the category sequence (X/(Y/Z))/B, B, Y/W; namely,



Now because (X/(Y/Z))/B becomes X/(W/Z) by fa>, it follows that X/(Y/Z) = X/(W/Z), and so Y = W. Further, B combines with Y/W (i.e., Y/Y) to give B again, so B is required to be of the form R/(Y/Y), for some R. (Note that R/(Y/Y) could also combine with Y/Y by fa>, but nothing prevents fpfc> from applying here as well.) In summary, G' allows the following sort of derivational ambiguity (and others symmetrical to it):





Is this a spurious or a genuine ambiguity? Letting the three leaf constituents have semantic values f, g, and h, respectively, we obtain $\lambda i[f(g)(Bhi)]$ for the root node of the left tree and $f[\lambda i[g(Bhi)]]$ for the root of the tree on the right. (Bhi denotes the composition of functions h and i.) These expressions are certainly non-equivalent for aribitrary functions f, g, h. [1] At any rate, we might ask if this sort of ambiguity can lead to an explosion of combinatorial possibilities like the one we were trying to rid ourselves of in the first place. The worst case would be when there is a sequence of n categories Y/Y extending rightward, thus:

(X/(Y/Z))/(R/(Y/Y))   R/(Y/Y)   Y/Y   Y/Y . . . Y/Y

Now R/(Y/Y) can combine with Y/Y's by fpfc, yielding R/(Y/Y) each time, then combine with the large category on the left by fa> to give X/(Y/Z), which can then combine with any remaining Y/Y's by fpfc> to give X/(Y/Z) back again. The lone instance of fa> can thus

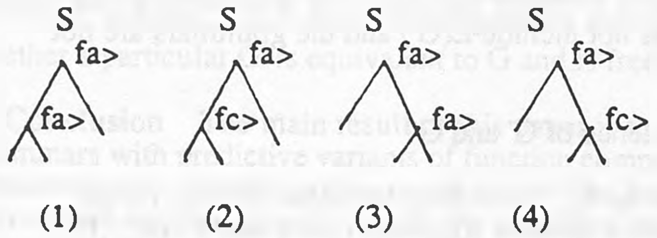
[1]Even so, it appears that if these functions are constrained by the form of the categories to which they are assigned (e.g., h must be a function from [[Y]]-type things to [[Y]]-type things, etc.), then the two expression may be equivalent and the ambiguity is a "spurious" one in the language of G'. At any rate, this point is moot given succeeding comments that these derivations need to be ruled out for G' to be equivalent to G.

occur at any point in the derivation, and if there are n Y/Y's, there will be n+1 distinct derivation trees. Thus, the number of derivations grows only linearly with the number of occurrences of Y/Y, not with a Catalan growth rate.

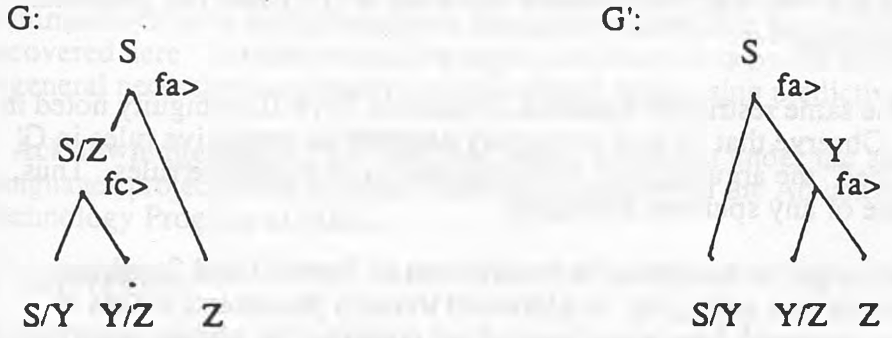
5. Equivalence of G and G' In considering equivalence of these grammars, we first take up the question of whether L(G) is a subset of L(G') followed by the question of whether L(G') is a subset of L(G).

5.1. <u>Predictive composition includes composition</u> Proof sketch: We show by induction on the depth of derivation trees that any derivation in G has a derivation in G'.

Any derivation of category S in G must end in fa> (or fa<). Consider the extension by depth one of a derivation tree headed by fa>. We consider 4 (not always mutually exclusive) cases. (Others include the symmetrical < variants and those that are excluded by directionality clashes).

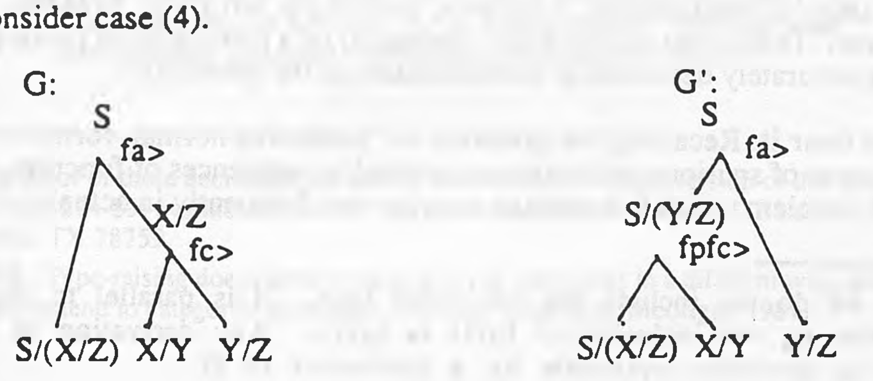


Cases (1) and (3) are common to G and G'. Consider case (2). From the definitions of fa> and fc>, the categories of the derivation must be as shown on the left, where Y and Z are any categories.



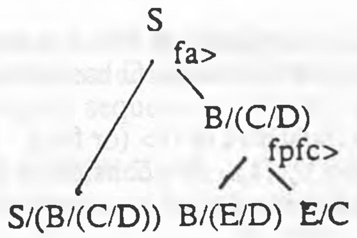
In G' there is a corresponding derivation from the same sequence of categories, as shown on the right. There is also this derivation in G, but G', lacking fc>, has only this one for this category sequence.

Consider case (4).



G' lacks fc>, but fpfc> allows (just) one derivation for this category sequence. The other cases symmetrical to these follow similarly.

5.2. <u>Does L(G) subsume L(G')</u>? Consider the following derivation in G':



There is no corresponding derivation in G. (Neither fa> nor fc> is applicable to the given categories.) Thus, in general, L(G) does not include L(G') and the grammars are not equivalent.

What can be done about the non-equivalence of G' and G?

1. **Restrict rule application in G'**: One may stipulate that the result category of a predictive rule cannot serve as argument in any other rule. (In function application X/Y Y => Z we take Y to be the argument category. In predictive rule X/(Y/Z) Y/W => X/(W/Z) we take the Y/W to be the argument. For backwards rules, the argument category is the leftmost term.) In the derivation just above, the predictive rule fpfc> "feeds" fa> as argument. If derivations in G' are restricted in this way, L(G') is provably included in L(G), and the grammars are weakly equivalent.[1]

Moreover, the same restriction banishes all cases of Type II ambiguity noted in Sec. 4 above. Observe that Type II ambiguity depends on predictive rules in G' being able to "feed" the arguments of further instances of predictive rules. Thus, G' becomes free of any spurious ambiguity.

This approach might be thought to be reminiscent of Pareschi and Steedman (1987), where spurious ambiguity is addressed through procedural means in parsing. Yet our approach here actually need not constrain the parsing algorithm at all. A node formed by a predictive rule can be flagged, say, by a feature, while those formed by fa would not be. All combinatory rules could then have a feature on their "argument" categories that would block when encountering this flag. This rather minimal amount of additional bookkeeping could easily be accommodated in the parsing strategy of one's choice: top-down, bottom-up, left-right, breadth-first, or whatever. Thus, what at first might appear to be a constraint on parsing would be more accurately described as a modification to the grammar.

2. **Grin and bear it**: Recasting the grammar in "predictive normal form" eliminates all cases of spurious ambiguity occasioned by sequences of function composition, a problem which is known to crop up very frequently in actual

[1]For lack of space, we do not include the full proof here. It is parallel to the proof in Sec. 5.1 showing the inclusion of L(G) in L(G'). Any derivation in this newly restricted G' is provably replacable by a derivation in G.

applications and to cause serious delays in parsing times. On the other hand, because of the complexity and the rather specific forms of the categories which give rise to the spurious ambiguities and the "spurious derivations" in the G' examples above, it seems reasonable to suppose that such cases are unlikely to be encountered very often in ordinary applications. In any event, as we noted above, the number of Type II ambiguous derivations in G' grows only linearly and not in Catalan fashion with increasing string length and would not be expected to lead to intolerable parsing times. The slight profligacy of G' over G might, therefore, present no serious practical problem.

For those still inclined to worry, we offer the following reassuring fact: a predictive normal form grammar can misbehave only if categories of sufficient "complexity" can be derived from the given set of categories in the lexicon, e.g., a category of the form S/(X/(W/Z)) in the case of non-equivalence above and of the form (X/(Y/Z))/(R/(Y/Y)) in the instances of Type II ambiguity. But given such a grammar and the lexical categories it is a decidable question whether any categories of the undesired complexity can arise during a derivation.[1] (We wish to thank Jim Barnett for suggestions on how to prove this.) Thus one can tell whether a particular G' is equivalent to G and is free from spurious ambiguity.[2]

6. Conclusion

The main result of this paper is that we have shown that Categorial Grammars with predictive variants of function composition rules can satisfy the requirements for normalization, namely, that the "compiled" grammars preserve equivalence and that they do so with the benefit of eliminating the parsing problem occasioned by spurious ambiguity. We have also enumerated decidability proofs of interest. Our next task is to explore the predictive normal form strategy with more expressive, and more nearly adequate, Categorial systems such as those that incorporate some form of generalized composition and conjunction, type-raising, etc. What we expect to find is that if predictive normalization techniques are applicable at all, the predictive grammars will have a relationship to their source forms that parallels the one we have uncovered here. In other words, we expect the restriction on the use of predictive rules is in general necessary for preserving equivalence when using predictive combinators.

7. Acknowledgements This work has been carried out under the auspices of the Interface Languages project in the Human Interface Laboratory of the Advanced Computer Technology Program at MCC.

8. References

Bouma, G. (1989) Efficient Processing of Flexible Categorial Grammar. In Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics, , 10-12 April 1989, pp. 19-26

Dowty, D. (1987) Type Raising, Functional Composition, and Non-Constituent Conjunction. In Oehrle, R., E. Bach, and D. Wheeler (eds.), Categorial Grammars and Natural Language Structures. Dordrecht: Reidel.

[1] The proof of these decidability results is contained in a longer version of this paper (MCC technical report ACT-HI-274-89) available from MCC, Human Interface Lab, 3500 W. Balcones Research Center Drive, Austin, TX 78759.

[2] N.B. Type-raising does increase complexity of categories in a different way, and thus these observations do not extend to categorial grammars with such rules (e.g., Moortgat 1987).

Gardent, C., and G. Bes. 1989. Efficient Parsing for French. In Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, 26-29 June 1989, Vancouver, pp. 280-287.

Hepple, M., and G. Morrill. (1989) Parsing and Derivational Equivalence. In Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics, 10-12 April 1989, Manchester, England, pp. 9-18.

Koenig, E. 1989. Parsing as Natural Deduction. In Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics, 26-29 June 1989, University of British Columbia, Vancouver, pp. 272-286.

Lambek, J. (1958). The Mathematics of Sentence Structure. American Mathematical Monthly 65:154-170.

Moortgat, M. (1987) Lambek Categorial Grammar and the Autonomy Thesis. Paper presented at the ZWO Symposium 'Morphology and Modularity', Utrecht, 16-18 June 1986. [Available as INL Working Paper 87-03, Instituut voor Nederlandse Lexicologie, Leiden, Netherlands.]

Moortgat, M. (1988) Categorial Investigations: Logical and Linguistic Aspects of the Lambek Calculus. Foris.

Morrill, G. (1988) Extraction and Coordination in Phrase Structure Grammar and Categorial Grammar. Ph.D. dissertation, Centre for Cognitive Science, University of Edinburgh.

Pareschi, R., and M. Steedman (1987) A Lazy Way to Chart-Parse with Categorial Grammars. Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, 6-9 July 1987, Stanford, pp. 81-88.

Steedman, M. (1985) Dependency and Coordination in the Grammar of Dutch and English. Language 61:523-568.

Steedman, M. (1987) Combinatory Grammars and Parasitic Gaps. Natural Language and Linguistic Theory 5:403-440.

Van Bentham, J. (1986) Essays in Logical Semantics. Reidel.

Wittenburg, K. (1986) Natural Language Parsing with Combinatory Categorial Grammars in a Graph-Unification-Based Formalism. Ph.D. dissertation, University of Texas at Austin.

Wittenburg, K. (1987) Predictive Combinators: A Method for Efficient Parsing of Combinatory Categorial Grammars. Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics, 6-9 July 1987, Stanford, pp. 73-80.

Parsing Spoken Language Using Combinatory Grammars*

Mark Steedman
Computer and Information Science, U.Penn.

Combinatory Grammars are a generalisation of Categorial Grammars to include operations on function categories corresponding to the combinators of Combinatory Logic, such as functional composition and type raising. The introduction of such operations is motivated by the need to provide an explanatory account of coordination and unbounded dependency. However, the associativity of functional composition tends to engender an equivalence class of possible derivations for each derivation permitted by more traditional grammars. While all derivations in each class by definition deliver the same function-argument relations in their interpretation, the proliferation of structural analyses presents obvious problems for parsing within this framework and the related approaches based on the Lambek calculus (Moortgat).

This problem has been called the problem of "spurious ambiguity", (although it will become apparent that the term is rather misleading). A number of ways of dealing with it have been proposed, including compiling the grammar into a different form (Wittenburg), "normal form"-based parsing (Hepple and Morrill, Koenig), and a "lazy" chart parsing technique which directly exploits the properties of the combinatory rules themselves to provide a unified treatment for "spurious" ambiguities and "genuine" attachment ambiguities (Pareschi and Steedman).

Recent work suggests that the very free notion of syntactic structure that is engendered by the theory is identical to the notion of structure that is required by recent theories of phrasal intonation and prosody. Intonational Structure is notoriously freer than traditional syntactic structure, and is commonly regarded as conveying distinctions of discourse focus and propositional attitude. It is argued that the focussed entities, propositions, and abstractions that are associated with a given intonational structure can be identified with the interpretations that the grammar provides for the non-standard constituents that it allows under one particular derivation from an equivalence class. The constituent interpretations corresponding to each possible intonational tune belong to the same equivalence class, and therefore reduce to the same canonical function argument relations. However, it is apparent that the ambiguity between derivations in the same equivalence class is not spurious at all, but meaning-bearing.

Of course, not *all* structural ambiguities are resolved by distinctions of intonation. (An example is PP attachment ambiguity). It follows that some of the techniques proposed for written parsing must be implicated as well. However, the theory opens the possibility of unifying phonological and syntactic processing, as well as simplifying the architecture required for integrating higher-level modules in spoken language processing.

*I am grateful to Julia Hirschberg, Aravind Joshi, Mitch Marcus, Janet Pierrehumbert, and Bonnie Lynn Webber for comments and advice. The research was supported by DARPA grant no. N0014-85-K0018, ARO grant no. DAAG29-84-K-0061, and NSF grant no. CER MCS 82-19196.

Structure and Intonation

Phrasal intonation is notorious for structuring the words of spoken utterances into groups which frequently violate orthodox notions of constituency. For example, the normal prosody for the answer (b) to the following question (a) imposes the intonational constituency indicated by the brackets (stress is indicated by capitals):

(1) a. I know that brassicas are a good source of minerals, but what are LEGumes a good source of?
b. (LEGumes are a good source of) VITamins.

Such a grouping cuts right across the traditional syntactic structure of the sentence. The presence of two apparently uncoupled levels of structure in natural language grammar appears to complicate the path from speech to interpretation unreasonably, and to thereby threaten a number of computational applications.

Nevertheless, intonational structure is strongly constrained by meaning. Contours imposing bracketings like the following are not allowed:

(2) # Three doctors (in ten prefer cats)

Halliday [5] seems to have been the first to identify this phenomenon, which Selkirk [16] has called the "Sense Unit Condition", and to observe that this constraint seems to follow from the *function* of phrasal intonation, which is to convey distinctions of focus, information, and propositional attitude towards entities in the discourse. These entities are more diverse than mere nounphrase or propositional referents, but they do not include such non-concepts as "in ten prefer cats."

One discourse category that they *do* include is what E. Prince [15] calls "open propositions". Open propositions are most easily understood as being that which is introduced into the discourse context by a Wh-question. So for example the question in (1), *What are legumes a good source of?* introduces an open proposition which it is most natural to think of as a functional *abstraction*, which would be written as follows in the notation of the λ-calculus:

(3) $\lambda x[good'(source'\ x)\ legumes']$

(Primes indicate interpretations whose detailed semantics is of no direct concern here.) When this function or concept is supplied with an argument *vitamins'*, it *reduces* to give a proposition, with the same function argument relations as the canonical sentence:

(4) $good'(source'\ vitamins')legumes'$

It is the presence of the above open proposition rather than some other that makes the intonation contour in (1) felicitous. (I am not claiming that its presence uniquely *determines* this response, nor that its explicit mention is necessary for interpreting the response.)

All natural languages include syntactic constructions whose semantics is also reminiscent of functional abstraction. The most obvious and tractable class are Wh-constructions themselves, in which exactly the same fragments that can be delineated by a single intonation contour appear as the residue of the subordinate clause. But another and much more problematic class are the fragments that result from coordinate constructions. It is striking that the residues of wh-movement and conjunction reduction are also subject to something like a "sense unit condition". For example, strings like "in ten prefer cats" are not conjoinable:

(5) *Three doctors in ten prefer cats,
    and in twenty eat carrots.

While coordinate constructions have constituted another major source of complexity for natural language understanding by machine, it is tempting to think that this conspiracy between syntax and prosody might point to a unified notion of structure that is somewhat different from traditional surface constituency.

Combinatory Grammars.

Combinatory Categorial Grammar (CCG, [17]) is an extension of Categorial Grammar (CG). Elements like verbs are associated with a syntactic "category" which identifies them as *functions*, and specifies the type and directionality of their arguments and the type of their result:

(6) *eats* `:- (S\NP)/NP: eat'`

The category can be regarded as encoding the semantic type of their translation. Such functions can combine with arguments of the appropriate type and position by functional application:

```
(7) Harry     eats     apples
    ------ ---------- ------
      NP    (S\NP)/NP   NP
            --------------->
                 S\NP
       ----------------<
                 S
```

Because the syntactic functional type is identical to the semantic type, apart from directionality, this derivation also builds a compositional interpretation, *eats'apples'harry'*, and of course such a "pure" categorial grammar is context free. Coordination might be included in CG via the following rule, allowing any constituents of like type, including functions, to form a single constituent of the same type:

(8) $X \quad conj \quad X \quad \Rightarrow \quad X$

```
(9) I   cooked   and    ate    a frog
    -- --------- ---- --------- ------
    NP (S\NP)/NP conj (S\NP)/NP   NP
       -------------------------&
              (S\NP)/NP
```

(The rest of the derivation is omitted, being the same as in (7).) In order to allow coordination of contiguous strings that do not constitute constituents, CCG generalises the grammar to allow certain operations on functions related to Curry's combinators [4]. For example, functions may *compose*, as well as apply, under the following rule

(10) Forward Composition:
$X/Y : F \quad Y/Z : G \quad \Rightarrow \quad X/Z : \lambda x\ F(Gx)$

The most important single property of combinatory rules like this is that they have an invariant semantics. This one composes the interpretations of the functions that it applies to, as is apparent from the right hand side of the rule.[1] Thus sentences like *I cooked, and might eat, the beans* can be accepted, via the following composition of two verbs (indexed as B, following Curry's nomenclature) to yield a composite of the same category as a transitive verb. Crucially, composition also yields the appropriate interpretation, assuming that a semantics is also provided for the coordination rule.

(11)
```
 cooked    and  might     eat
--------- ---- --------- -----
(S\NP)/NP conj (S\NP)/VP VP/NP
               ---------------->B
                  (S\NP)/NP
 ----------------------------&
            (S\NP)/NP
```

Combinatory grammars also include type-raising rules, which turn arguments into functions over functions-over-such-arguments. These rules allow arguments to compose, and thereby take part in coordinations like *I cooked, and you ate, the legumes.* They too have an invariant compositional semantics which ensures that the result has an appropriate interpretation. For example, the following rule allows the conjuncts to form as below (again, the remainder of the derivation is omitted):

(12) Subject Type-raising:
$NP : y \;\; \Rightarrow \;\; S/(S\backslash NP) : \lambda F\, Fy$

(13)
```
   I      cooked    and   you      ate
-------- --------- ---- -------- ---------
   NP    (S\NP)/NP conj    NP    (S\NP)/NP
-------->T                -------->T
S/(S\NP)                  S/(S\NP)
------------------->B     ------------------->B
       S/NP                     S/NP
       ----------------------------&
                   S/NP
```

Intonation in a CCG.

Inspection of the above examples shows that Combinatory grammars embody an unusual view of surface structure, according to which strings like *Betty might eat* are constituents. In fact, according to this view, surface structure is a much more ambiguous affair than is generally realised, for they must also be possible constituents of non-coordinate sentences like *Betty might eat the mushrooms*, as well. (See [11] and [19] for a discussion of the obvious problems that this fact engenders for parsing written text.) An entirely unconstrained combinatory grammar would in fact allow more or less any bracketing on a sentence. However, the actual grammars we write for configurational languages like English are heavily constrained by local conditions. (An example would be a condition on the composition rule that is tacitly assumed here, forbidding the variable Y in the composition rule to be instantiated as NP, thus excluding constituents like $*[\textit{eat the}]_{VP/N}$).

The claim of the present paper is simply that particular surface structures that are induced by the specific combinatory grammar that was introduced to explain coordination in English are identical to the intonational structures that are required to specify the possible intonation contours for those

[1] The rule uses the notation of the λ-calculus in the semantics, for clarity. This should not obscure the fact that it is functional composition itself that is the primitive, not the λ operator.

same sentences of English.[2] More specifically, the claim is that that in spoken utterance, intonation largely determines *which* of the many possible bracketings permitted by the combinatory syntax of English is intended, and that the interpretations of the constituents are related to distinctions of focus among the concepts and open propositions that the speaker has in mind. Thus, whatever problems for parsing written text arise from the profusion of equivalent alternative surface structures engendered by this theory, these "spurious" ambiguities seem to be to a great extent resolved by prosody in spoken language. The theory therefore offers the possibility that phonology and parsing can be merged into a single unitary process.

The proof of this claim lies in showing that the rules of combinatory grammar can be annotated with intonation contour schemata, which limit their application in spoken discourse, and to showing that the major constituents of intonated utterances like (1)b, under the analyses that these rules permit, correspond to the focus structure of the context to which they are appropriate, such as (1)a.

I shall use a notation which is based on the theory of Pierrehumbert [12], as modified in more recent work by Selkirk [16], Beckman and Pierrehumbert [2], [13], and Pierrehumbert and Hirschberg [14]. I have tried as far as possible to take my examples and the associated intonational annotations from those authors.

I follow Pierrehumbert in assuming two abstract pitch levels, and three types of tones, as follows. There are two phrasal tones, written `H` and `L`, denoting high or low "simple" tones — that is, level functions of pitch against time. There are also two boundary tones, written `H%` and `L%`, denoting an intonational phrase-final rise or fall. Of Pierrhumberts six pitch accent tones, I shall only be concerned with two, the `H*` accent and the `L+H*`. The phonetic or acoustic realisation of pitch accents is a complex matter. Roughly speaking, the `L+H*` pitch accent that is extensively discussed below in the context of the `L+H* LH%` melody generally appears as a maximum which is preceded by a distinctive low level, and peaks *later* than the corresponding `H*` pitch accent when the same sequence is spoken with the `H* L` melody that goes with "new" information, and which is the other melody considered below.

In the more recent versions of the theory, Pierrehumbert and her colleagues distinguish *two* levels of prosodic phrase that include a pitch accent tone. They are the intonational phrase proper, and the "intermediate phrase". Both end in a phrasal tone, but only intonational phrases have additional boundary tones `H%` and `L%`. Intermediate phrases are bounded on the right by their phrasal tone alone, and do not appear to be characterised in F_0 by the same kind of final rise or fall that is characteristic of true intonational phrases. The distinction does not play an active role in the present account, but I shall follow the more recent notation of prosodic phrase boundaries in the examples, without further comment on the distinction.

There may also be parts of prosodic phrases where the fundamental frequency is merely interpolated between tones, notably the region between pitch accent and phrasal tone, and the region before a pitch accent. In Pierrehumbert's notation, such substrings bear no indication of abstract tone whatsoever.

A crucial feature of this theory for present purposes is that the position and shape of a given pitch accent in a prosodic phrase, and of its phrase accent and the associated right-hand boundary, are essentially invariant. If the constituent is very short - say, a monosyllabic nounphrase - then the whole intonational contour may be squeezed onto that one syllable. If the constituent is longer, then the pitch accent will appear at its left edge, the phrasal tone and boundary tone if any will appear at its right edge, and the intervening pitch contour will merely be interpolated. In this way, the tune can be spread over longer or shorter strings, in order to mark the corresponding constituents for the particular distinction of focus and propositional attitude that the melody denotes.

Consider for example the prosody of the sentence *Fred ate the beans* in the following pair of discourse

[2] There is a precedent for the claim that prosodic structure can be identified with the structures arising from the inclusion of associative operations in grammar in the work of Moortgat [9] and Oehrle [10], and in [?]

settings, which are adapted from Jackendoff [7, pp. 260]:

```
(14)  Q: Well, what about the BEAns?
         Who ate THEM?
      A: FRED   ate the BEA-ns.
         H*L             L+H*LH%

(15)  Q: Well, what about FRED?
         What did HE eat?
      A: FRED ate the BEAns.
         L+H* LH%     H* LL%
```

In these contexts, the main stressed syllables on both *Fred* and *the beans* receive a pitch accent, but a different one. In (14), the pitch accent contour on *Fred* is H*, while that on *beans* is L+H*. (I base these annotations on Pierrehumbert and Hirschberg's [14, ex. 33] discussion of this example.)

In the second example (15) above, the pitch accents are reversed: this time *Fred* is L+H* and *beans* is H*. The assignment of these tones seem to reflect the fact that (as Pierrehumbert and Hirschberg point out) H* is used to mark information that the speaker believes to be *new to the hearer*. In contrast, L+H* seems to be used to mark information which the current speaker knows to be given to the hearer (because the current hearer asked the original question), but which constitutes a novel topic of conversation for the speaker, standing in a contrastive relation to some *other* given information, constituting the previous topic. (If the information were merely given, it would receive *no* tone in Pierrehumbert's terms — or be left out altogether.) Thus in (15), the L+H* LH% phrase including this accent is spread across the phrase *Fred ate*.[3] Similarly, in (14), the same tune is confined to the object of the open proposition *ate the beans*, because the intonation of the original question indicates that eating beans *as opposed to some other comestible* is the new topic.

Syntax-driven Prosody.

The L+H* LH% intonational melody in example (15) belongs to a phrase *Fred ate* ... which corresponds under the combinatory theory of grammar to a grammatical constituent, complete with a translation equivalent to the open proposition $\lambda x[(ate'\ x)\ fred']$. The combinatory theory thus offers a way to assign contours like L+H* LH% to such novel constituents, entirely under the control of independently motivated rules of grammar. For example, the rule of forward composition should be made subject to a restriction which is in the terms of Pierrehumbert's theory an extremely natural one, amounting to the straightforward injunction "Don't apply this rule across an intonational phrase or intermediate phrase boundary". The modified rule allows the following derivation for *Fred ate* ..., in which for once the semantic interpretation is included:[4]

[3] An alternative prosody, in which the contrastive tune is confined to *Fred*, seems equally coherent, and may be the one intended by Jackendoff. I believe that this alternative is informationally distinct, and arises from an ambiguity as to whether the topic of this discourse is *Fred* or *What Fred ate*. It is accepted by the present rules.

[4] Again primes indicate interpretations whose details are of no concern here. It will be apparent from the derivations that the assumed semantic representation is at a level prior to the explicit representation of matters related to quantifier scope.

```
(16)       Fred                  ate
     ----------------   ---------------
         NP:fred'       (S\NP)/NP:ate'
           L+H*              LH%
     ----------------->T
     S/(S\NP):λ P  P fred'
           L+H*
     ----------------------------------->B
             S/NP: λX  (ate' X) fred'
                 L+H*LH%
```

The options incorporated in the tonal annotations of the rule allow the `L+H* LH%` tune to spread across any sequence that can be composed by repeated applications of the rule. For example, if the reply to the same question *What did Fred eat?* is *FRED must have eaten the BEANS*, then the tune will typically be spread over *Fred must have eaten* ..., as in the following derivation, in which much of the syntactic and semantic detail has been omitted in the interests of brevity:

```
(17)    Fred     must     have    eaten
     -------- --------- ------- -------
        NP    (S\NP)/VP VP/VPen VPen/NP
      L+H*                        LH%
     -------->T
      L+H*
     ------------------->B
         L+H*
         ---------------------->B
                L+H*
                ---------------------->B
                       L+H*LH%
```

On the assumption that forward functional application bears a complementary restriction, and can combine any intonation contours to yield their concatenation, except when the leftmost is a bare phrasal tone or phrasal tone and boundary tone, the derivation of (15) can be completed as follows:

```
(18)  Fred          ate            the      beans
     --------- -------------- ---------- --------
      NP:fred' (S\NP)/NP:ate' NP/N: the' N:beans'
      L+H*          LH%                  H* LL%
     --------->T              ------------------->
     S/(S\NP):                  NP:the' beans'
     λP P fred'                     H* LL%
      L+H*
     ------------------------>B
      S/NP:λX (ate' X) fred'
          L+H* LH%
          ---------------------------------->
             S: ate' (the' beans') fred'
                 L+H* LH%  H* LL%
```

The division into contrastive/given open proposition versus new information is appropriate, and no other derivation is allowed, given this intonation contour. Repeated application of the composition rule, as in (17), would allow the `L+H* LH%` contour to spread further, as in *(FRED must have eaten) the BEANS*.

In contrast, the intonation contour on (14) will not permit the annotated composition rule to apply, because *Fred* end with a L boundary intonation, so the bracketing imposed in (15) (and the formation of the corresponding open proposition) is simply not allowed. However, since forward functional application is unrestricted, the following derivation of (14) *is* allowed. Again, the derivation divides the sentence into new and given information consistent with the context given in the example:

```
(19)  Fred          ate            the       beans
     ---------- -------------- --------- --------
     NP:fred'   (S\NP)/NP:ate' NP/N:the' N:beans'
       H* L                               L+H* LH%
     -------->T                 ------------------>
      S/(S\NP):                   NP:the' beans'
     λP P fred'                     L+H* LH%
       H* L
                -------------------------------->
                        S\NP:eat'(the' beans')
                              L+H* LH%
     ---------------------------------------->
                 S: eat'(the' beans') fred'
                    H* L     L+H* LH%
```

The effect of the rules is to annotate the entire predicate as an L+H* LH%. It is emphasised that this does *not* mean that the *tone* is spread, but that the whole constituent is marked for the corresponding discourse function — roughly, as contrastive. The finer grain information that it is the object that is contrasted, while the verb is given, resides in the tree itself. Similarly, the fact that boundary tones are associated with words at the lowest level of the derivation does not mean that they are *part of* the word, nor that the word is the entity that they are a boundary *of*. It is prosodic phrases that they bound, and these also are defined by the tree. No other analysis is allowed for (19). Other cases considered by Jackendoff are considered in a more extended companion to the present paper [19], and are shown to yield only contextually appropriate interpretations.

Conclusions.

The problem of so-called "spurious" ambiguity, or multiple semantically equivalent derivations, now appears in a quite different light. While the semantic properties of the rules (notably the associativity of functional composition that engenders the problem in the first place) do indeed guarantee that these analyses are semantically equivalent at the level of Argument Structure, they are nonetheless meaning-bearing at the level of Information Structure. To call them "spurious" is rather misleading. What is more, while there are usually a great many different analyses for any given sequence of words, intonation contour often limits or even eliminates the non-determinism arising from this source.

The significance of eliminating non-determinism in this way should not be under-estimated. Similar intonational markers are involved in coordinate sentences, like the following 'right-node-raised" example:

(20) I will, and you won't, eat mushrooms

In such sentences the local ambiguity between composing *won't* and *eat* and applying the latter to its argument first is a *genuine* local ambiguity, equivalent to a local attachment ambiguity in a more traditional grammar, for only *one* of the alternatives will lead to a parse at all. And the correct alternative is the one that is selected by the restriction against forward composition across prosodic phrase boundaries.

However, the extent to which intonation alone renders parsing deterministic should also not be overstated. There still are sources of non-determinism in the grammar, which must be coped with somehow. Most obviously, there are sources common to all natural language grammars, such as the well-known PP-attachment ambiguities in the following example:

(21) Put the block in the box on the table.

While intonation *can* distinguish the two analyses, they do not seem to be *necessarily* so distinguished. There is also a residuum of so-called spurious ambiguity, because function categories bearing *no* tone are free to forward compose *and* to apply.

It is important to observe that this ambiguity is widespread, and that it is a true ambiguity in discourse interpretation. Consider yet another version of the example with which the paper began, uttered with only an H* LL% tune on the last word:

(22)

```
    Legumes are a good source of VItamins.
                                 H*   LL%
```

Such an intonation contour is compatible with *all* the analyses that the unannotated CCG would allow. However, such an utterance is also compatible with a large number of contextual open propositions. For example, it is a reasonable response to the question *What can you tell me about legumes?* But it is similarly reasonable as an answer to *What are legumes?*, or to *What are legumes a good source of?* The ambiguity of intonation with respect to such distinctions is well-known , and it would simply be incorrect not to include it . (See discussion in [1] and [8] for alternative proposals for ways of resolving it that are compatible with the present proposal.)

According to the present theory, the pathway between phonological form and interpretation is much simpler than has been thought up till now. Phonological Form maps directly onto Surface Structure, via rules of combinatory grammar annotated with abstract intonation contours. Surface Structure is identical to intonational structure, and maps directly onto Focus Structure, in which focussed and backgrounded entities and open propositions are represented by functional abstractions and arguments. Such structures reduce to yield canonical Function-Argument Structures. The proposal thus represents a return to the architecture proposed by Chomsky [3] and Jackendoff [7]. The difference is that the concept of surface structure has changed. It now really is *only* surface structure, supplemented by "annotations" which do nothing more than indicate the information structural status and intonational tune of *constituents* at that level.

While many problems remain, both in parsing written text with grammars that include associative operations, and at the signal-processing end, the benefits for automatic spoken language understanding are likely to be significant. Most obviously, where in the past parsing and phonological processing have delivered conflicting structural analyses, and have had to be pursued independently, they now are seen to be in concert. Processors can therefore be devised which use both sources of information at once, thus simplifying both problems. Furthermore, a syntactic analysis that is so closely related to the structure of the signal should be easier to use to "filter" the ambiguities arising from lexical recognition. What is likely to be more important in the long run, however, is that the constituents that arise under this analysis are also semantically interpreted. The paper has argued that these interpretations are directly related to the concepts, referents and themes that have been established in the context of discourse, say as the result of a question. The shortening and simplification of the path from speech to these higher levels of analysis offers the possibility of using those probably more effective resources to filter the proliferation of low level analyses as well.

References

[1] Altmann, Gerry and Mark Steedman: 1988, 'Interaction with Context During Human Sentence Processing' *Cognition*, 30, 191-238

[2] Beckman, Mary and Janet Pierrehumbert: 1986, 'Intonational Structure in Japanese and English', *Phonology Yearbook*, 3, 255-310.

[3] Chomsky, Noam: 1970, 'Deep Structure, Surface Structure, and Semantic Interpretation', in D. Steinberg and L. Jakobovits, *Semantics*, CUP, Cambridge, 1971, 183-216.

[4] Curry, Haskell and Robert Feys: 1958, *Combinatory Logic*, North Holland, Amsterdam.

[5] Halliday, Michael: 1967, *Intonation and Grammar in British English*, Mouton, The Hague.

[6] Hepple, Mark, and Glyn Morrill: 1989, 'Parsing and Derivational Equivalence', Proceedings of the Fourth Conference of the European Chapter of the ACL, Manchester, April 1989, 10-18.

[7] Jackendoff, Ray: 1972, *Semantic Interpretation in Generative Grammar*, MIT Press, Cambridge MA.

[8] Marcus, Mitch, Don Hindle, and Margaret Fleck: 1983, D-theory: Talking about Talking about Trees, Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics, Cambridge Mass, June, 1983, 129-136.

[9] Moortgat, Michael: 1988, *Categorial Investigations*, Foris, Dordrecht.

[10] Oehrle, Richard T.: 1985, paper to the Conference on Categorial Grammar, Tucson, AR, June 1985, in Richard T. Oehrle, E.. Bach and D. Wheeler, (eds), *Categorial Grammars and Natural Language Structures*, Reidel, Dordrecht, (in press).

[11] Pareschi, Remo, and Mark Steedman. 1987. A lazy way to chart parse with categorial grammars, *Proceedings of the 25th Annual Conference of the ACL, Stanford*, July 1987, 81–88.

[12] Pierrehumbert, Janet: 1980, *The Phonology and Phonetics of English Intonation*, Ph.D dissertation, MIT. (Distributed by Indiana University Linguistics Club, Bloomington, IN.)

[13] Pierrehumbert, Janet, and Mary Beckman: 1989, *Japanese Tone Structure*, MIT Press, Cambridge MA.

[14] Pierrehumbert, Janet, and Julia Hirschberg, 1987, 'The Meaning of Intonational Contours in the Interpretation of Discourse', ms. Bell Labs.

[15] Prince, Ellen F. 1986. On the syntactic marking of presupposed open propositions. Papers from the Parasession on Pragmatics and Grammatical Theory at the 22nd Regional Meeting of the Chicago Linguistic Society, 208-222.

[16] Selkirk, Elisabeth: *Phonology and Syntax*, MIT Press, Cambridge MA.

[17] Steedman, Mark: 1987. Combinatory grammars and parasitic gaps. NL&LT, 5, 403-439.

[18] Steedman, Mark: 1989, Structure and Intonation, ms. U. Penn.

[19] Wittenburg, Kent: 1987, 'Predictive Combinators: a Method for Efficient Processing of Combinatory Grammars', *Proceedings of the 25th Annual Conference of the ACL, Stanford*, July 1987, 73–80.

Recognition of Combinatory Categorial Grammars and Linear Indexed Grammars

K. Vijay-Shanker
Department of CIS
University of Delaware
Delaware, DE 19716

David J. Weir
Department of EECS
Northwestern University
Evanston, IL 60208

1 Introduction

In recent papers [14,15,3] we have shown that Combinatory Categorial Grammars (CCG), Head Grammars (HG), Linear Indexed Grammars (LIG), and Tree Adjoining Grammars (TAG) are weakly equivalent; i.e., they generate the same class of string languages. Although it is known that there are polynomial-time recognition algorithms for HG and TAG [7,11], there are no known polynomial-time recognition algorithms that work *directly* with CCG or LIG. In this paper we present polynomial-time recognition algorithms for CCG and LIG that resemble the CKY algorithm for Context-Free Grammars (CFG) [4,16].

The tree sets derived by a CFG can be recognized by *finite state* tree automata [10][1]. This is reflected in CFL bottom-up recognition algorithms such as the CKY algorithm. Intermediate configurations of the recognizer can be encoded by the states of these finite state automata (the nonterminal symbols of the grammar). The similarity of TAG, CCG, and LIG can be seen from the fact that the tree sets derived by these formalisms can be recognized by *pushdown* (rather than finite state) based tree automata. We give recognition algorithms for these formalisms by extending the CKY algorithm so that intermediate configurations are encoded using stacks. In [6] a chart parser for CCG is given where copies of stacks (derived categories) are stored explicitly in each chart entry. In Section 4 we show that storing stacks in this way leads to exponential run-time. In the algorithm we present here the stack is encoded by storing its top element together with information about where the remainder of the stack can be found. Thus, we avoid the need for multiple copies of parts of the same stack through the sharing of common substacks. This reduces the number of possible elements in each entry in the chart and results in a polynomial time algorithm since the time complexity is related to the number of elements in each chart entry.

It is not necessary to derive separate algorithms for CCG, LIG, and TAG. In proving that these formalisms are equivalent, we developed constructions that map grammars between the different formalisms. We can make use of these constructions to adapt an algorithm for one formalism into an algorithm for another. First we present a discussion of the recognition algorithm for LIG in Section 2[2].

[1]A bottom-up finite state tree automaton reads a tree bottom-up. The state that the automaton associates with each node that it visits will depend on the states associated with the children of the node.

[2]We consider LIG that correspond to the Chomsky normal form for CFG although we do not prove that all LIG have an equivalent grammar in this form. A discussion of the recognition algorithm for LIG in this form is sufficient to enable us to adapt it to give a recognition algorithm for CCG, which is the primary purpose of this paper.

We present the LIG recognition algorithm first since it appeares to be the clearest example involving the use of the notion of stacks in derivations. In Section 3 we give an informal description of how to map a CCG to an equivalent LIG. Based on this relationship we adapt the recognition algorithm for LIG to one for CCG.

2 Linear Indexed Grammars

An Indexed Grammar [1] can be viewed as a CFG in which each nonterminal is associated with a stack of symbols. In addition to rewriting nonterminals, productions can have the effect of pushing or popping symbols on top of the stacks that are associated with each nonterminal. A LIG [2] is an Indexed Grammar in which the stack associated with the nonterminal of the LHS of each production can only be associated with one of the occurrences of nonterminals on the RHS of the production. Empty stacks are associated with other occurrences of nonterminals on the RHS of the production. We write $A[\cdot\cdot]$ (or $A[\cdot\cdot\gamma]$) to denote the nonterminal A associated with an arbitrary stack (or an arbitrary stack whose top symbol is γ). A nonterminal A with an empty stack is written $A[\,]$.

Definition 2.1 A LIG, G, is denoted by (V_N, V_T, V_I, S, P) where

V_N is a finite set of nonterminals,
V_T is a finite set of terminals,
V_I is a finite set of indices (stack symbols),
$S \in V_N$ is the start symbol, and
P is a finite set of productions, having one of the following forms.

$$A[\cdot\cdot\gamma] \rightarrow A_1[\,]\ldots A_i[\cdot\cdot]\ldots A_n[\,] \qquad A[\cdot\cdot] \rightarrow A_1[\,]\ldots A_i[\cdot\cdot\gamma]\ldots A_n[\,] \qquad A[\,] \rightarrow a$$

where $A, A_1, \ldots, A_n \in V_N$ and $a \in \{\epsilon\} \cup V_T$.

The relation $\underset{G}{\Longrightarrow}$ is defined as follows where $\alpha \in V_I^*$ and Υ_1, Υ_2 are strings of nonterminals with associated stacks.

- If $A[\cdot\cdot\gamma] \rightarrow A_1[\,]\ldots A_i[\cdot\cdot]\ldots A_n[\,] \in P$ then

$$\Upsilon_1 A[\alpha\gamma]\Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 A_1[\,]\ldots A_i[\alpha]\ldots A_n[\,]\Upsilon_2$$

- If $A[\cdot\cdot] \rightarrow A_1[\,]\ldots A_i[\cdot\cdot\gamma]\ldots A_n[\,] \in P$ then

$$\Upsilon_1 A[\alpha]\Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 A_1[\,]\ldots A_i[\alpha\gamma]\ldots A_n[\,]\Upsilon_2$$

In each of these two cases we say that A_i is the **distinguished** child of A in the derivation.

- If $A[\,] \rightarrow a \in P$ then

$$\Upsilon_1 A[\,]\Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 a \Upsilon_2$$

The language generated by a LIG, G, $L(G) = \{\, w \mid S[\,] \overset{*}{\underset{G}{\Longrightarrow}} w \,\}$.

2.1 Recognition of LIG

In considering the recognition of LIG, we assume that the underlying CFG is in Chomsky Normal Form; i.e., either two nonterminals (with their stacks) or a single terminal can appear on the RHS of a rule. Although we have not confirmed whether this yields a normal form, a recognition algorithm for LIG in this form of LIG is sufficient to enable us to develop a recognition algorithm for CCG. We use an array L consisting of n^2 elements where the string to be recognized is $a_1 \ldots a_n$. In the case of the CKY algorithm for CFG recognition each array element $L_{i,j}$ contains that subset of the nonterminal symbols that can derive the substring $a_i \ldots a_j$. In our algorithm the elements stored in $L_{i,j}$ will encode those nonterminals *and associated stacks* that can derive the string $a_i \ldots a_j$.

In order to obtain a polynomial algorithm we must encode the stacks efficiently. With each nonterminal we store only the top of its associated stack and an indication of the element in L where the next part of the stack can be found. This is achieved by storing sets of tuples of the form $(A, \gamma, A', \gamma', p, q)$ in the array elements. Roughly speaking, a tuple $(A, \gamma, A', \gamma', p, q)$ is stored in $L_{i,j}$ when $A[\alpha\gamma'\gamma] \overset{*}{\Longrightarrow} a_i \ldots a_j$ and $A'[\alpha\gamma'] \overset{*}{\Longrightarrow} a_p \ldots a_q$ where α is a string of stack symbols and A' is the unique distinguished descendent of A in the derivation of $a_i \ldots a_j$.

Note that tuples, as defined above, assume the presence of at least two stack symbols. We must also consider two other cases in which a nonterminal is associated with either a stack of a single element, or with the empty stack. Suppose that A is associated with a stack containing only the single symbol γ. This case will be represented using tuples of the form $(A, \gamma, A', -, p, q)$ ("$-$" indicates that an empty stack is associated with A'). When an empty stack is associated with A we will use the tuple $(A, -, -, -, -, -)$. In discussing the general case for tuples we will use the form $(A, \gamma, A', \gamma', p, q)$ with the understanding that: $A' \in V_N$ or $-$; $\gamma, \gamma' \in V_I$ or $-$; and p, q are integer between 1 and n or $-$. The algorithm can be understood by verifying that at each step the following invariant holds.

Proposition 2.1 $(A, \gamma, A', \gamma', p, q) \in L_{i,j}$ if and only if one of the following holds.

If $\gamma' \neq -$ then $A[\gamma] \overset{*}{\Longrightarrow} a_i \ldots a_{p-1} A'[\,] a_{q+1} \ldots a_j$ and $A'[\alpha\gamma'] \overset{*}{\Longrightarrow} a_p \ldots a_q$ for some $\alpha \in V_I^*$ where A' is a distinguished descendent of A. Note that this implies that for all $\beta \in V_I^*$, $A[\beta\gamma] \overset{*}{\Longrightarrow} a_i \ldots a_{p-1} A'[\beta] a_{q+1} \ldots a_j$. Thus, for $\beta = \alpha\gamma'$, $A[\alpha\gamma'\gamma] \overset{*}{\Longrightarrow} a_i \ldots a_{p-1} A'[\alpha\gamma'] a_{q+1} \ldots a_j$ which implies $A[\alpha\gamma'\gamma] \overset{*}{\Longrightarrow} a_i \ldots a_j$.

If $\gamma' = - \neq A'$ then $A[\gamma] \overset{*}{\Longrightarrow} a_i \ldots a_j$ and $A'[\,] \overset{*}{\Longrightarrow} a_p \ldots a_q$.

If $A' = -$ then $A[\,] \overset{*}{\Longrightarrow} a_i \ldots a_j$.

We now describe how each entry $L_{i,j}$ is filled. As the algorithm proceeds, the gap between i and j increases until it spans the entire input. The input, $a_1 \ldots a_n$, is accepted if $(S, -, -, -, -, -) \in L_{1,n}$. New entries are added to the array elements according to the productions of the grammar as follows.

1. The production $A[\cdot\cdot\gamma] \rightarrow A_1[\,] A_2[\cdot\cdot]$ is used while filling the array element $L_{i,j}$ as follows. For every k where $i \leq k \leq j$, check the previously completed array elements $L_{i,k}$ and $L_{k+1,j}$ for $(A_1, -, -, -, -, -)$ and some $(A_2, \gamma_2, A_3, \gamma_3, p, q)$, respectively. If these entries are found add $(A, \gamma, A_2, \gamma_2, k+1, j)$ to $L_{i,j}$. If $\gamma_2 = \gamma_3 = A_3 = p = q = -$ we place $(A, \gamma, A_2, -, k+1, j)$ in $L_{i,j}$. From these entries in $L_{i,k}$ and $L_{k+1,j}$ we know by Proposition 2.1 that $A_1[\,] \overset{*}{\Longrightarrow} a_i \ldots a_k$

and $A_2[\alpha] \overset{*}{\Longrightarrow} a_{k+1} \ldots a_j$ for some $\alpha \in V_I^*$. Thus, $A[\alpha\gamma] \overset{*}{\Longrightarrow} a_i \ldots a_j$. The production $A[\cdot\cdot\gamma] \rightarrow A_1[\cdot\cdot]A_2[\,]$ is handled similarly.

2. Suppose $A[\cdot\cdot] \rightarrow A_1[\,]A_2[\cdot\cdot\gamma]$ is a production. When filling $L_{i,j}$ we must check whether the tuple $(A_1, -, -, -, -, -)$ is in $L_{i,k}$ and $(A_2, \gamma, A_3, \gamma_3, p, q)$ is in $L_{k+1,j}$ for some k between i and j. If we do find these tuples then we check in $L_{p,q}$ for some $(A_3, \gamma_3, A_4, \gamma_4, r, s)$. In this case we add $(A, \gamma_3, A_4, \gamma_4, r, s)$ to $L_{i,j}$. If $\gamma_3 = -$ then the stack associated with A_3 is empty, $\gamma_4 = A_4 = r = s = -$, and we add the tuple $(A, -, -, -, r, s)$ to $L_{i,j}$. The above steps can be related to Proposition 2.1 as follows.

   (a) If $\gamma_3 \not= -$ then for some $\alpha \in V_I^*$, $A_4[\alpha\gamma_4] \overset{*}{\Longrightarrow} a_r \ldots a_s$ a subderivation of $A_3[\alpha\gamma_4\gamma_3] \overset{*}{\Longrightarrow} a_p \ldots a_q$ a subderivation of $A_2[\alpha\gamma_4\gamma_3\gamma] \overset{*}{\Longrightarrow} a_{k+1} \ldots a_j$. Combining this with $A_1[\,] \overset{*}{\Longrightarrow} a_i \ldots a_k$ we have $A[\alpha\gamma_4\gamma_3] \overset{*}{\Longrightarrow} a_i \ldots a_j$.

   (b) If $\gamma_3 = -$ then $A_3[\,] \overset{*}{\Longrightarrow} a_p \ldots a_q$ is a subderivation of $A_2[\gamma] \overset{*}{\Longrightarrow} a_{k+1} \ldots a_j$. Combining with $A_1[\,] \overset{*}{\Longrightarrow} a_i \ldots a_k$, we get $A[\,] \overset{*}{\Longrightarrow} a_i \ldots a_j$.

   Productions of the form $A[\cdot\cdot] \rightarrow A_1[\cdot\cdot\gamma]A_2[\,]$ are handled similarly.

3. Suppose $A[\,] \rightarrow a$ is a production. This is used by the algorithm in the initialization of the array L. If the terminal symbol a is the same as the i^{th} symbol in the input string, i.e., $a = a_i$, then we include $(A, -, -, -, -, -)$ in the array element $L_{i,i}$.

2.2 Complete Algorithm

```
For $i := 1$ to $n$ do
    $L_{i,i} := \{(A, -, -, -, -, -) \mid A[\,] \rightarrow a_i\}$
For $i := n$ to 1 do
  For $j := i$ to $n$ do
    For $k := i$ to $j - 1$ do
      Step 1a. For each production $A[\cdot\cdot\gamma] \rightarrow A_1[\,]A_2[\cdot\cdot]$
        if $(A_1, -, -, -, -, -) \in L_{i,k}$ and $(A_2, \gamma_2, A_3, \gamma_3, p, q) \in L_{k+1,j}$
        then $L_{i,j} := L_{i,j} \cup \{(A, \gamma, A_2, \gamma_2, k + 1, j)\}$
      Step 1b. For each production $A[\cdot\cdot\gamma] \rightarrow A_1[\cdot\cdot]A_2[\,]$
        if $(A_1, \gamma_1, A_3, \gamma_3, p, q) \in L_{i,k}$ and $(A_2, -, -, -, -, -) \in L_{k+1,j}$
        then $L_{i,j} := L_{i,j} \cup \{(A, \gamma, A_1, \gamma_1, i, k)\}$
      Step 2a. For each production $A[\cdot\cdot] \rightarrow A_1[\,]A_2[\cdot\cdot\gamma]$
        if $(A_2, \gamma, A_3, \gamma_3, p, q) \in L_{k+1,j}$, $(A_3, \gamma_3, A_4, \gamma_4, r, s) \in L_{p,q}$, and $(A_1, -, -, -, -, -) \in L_{i,k}$
        then $L_{i,j} := L_{i,j} \cup \{(A, \gamma_3, A_4, \gamma_4, r, s)\}$
      Step 2b. For each production $A[\cdot\cdot] \rightarrow A_1[\cdot\cdot\gamma]A_2[\,]$
        if $(A_1, \gamma, A_3, \gamma_3, p, q) \in L_{i,k}$, $(A_3, \gamma_3, A_4, \gamma_4, r, s) \in L_{p,q}$, and $(A_2, -, -, -, -, -) \in L_{k+1,j}$
        then $L_{i,j} := L_{i,j} \cup \{(A, \gamma_3, A_4, \gamma_4, r, s)\}$
```

2.3 Complexity of the Algorithm

Any array element, say $L_{i,j}$, is a set of tuples of the form $(A, \gamma, A', \gamma', p, q)$ where p and q are either integers between i and j, or $i = j = -$. The number of possible values for A, A', γ, and γ' are each bounded by a constant. Thus the number of tuples in $L_{i,j}$ is at most $O((j-i)^2)$. For a fixed value of i, j, k, steps 1a and 1b will attempt to place at most $O((j-i)^2)$ tuples in $L_{i,j}$. Before adding any tuple to $L_{i,j}$ we first check whether the tuple is already present in that array element. This can be done in constant time on a RAM by assuming that each array element $L_{i,j}$ is itself an $(i+1) \times (j+1)$ array. A tuple of the form $(A, \gamma, A', \gamma', p, q)$ will be in the $\langle p, q \rangle^{th}$ element of $L_{i,j}$ and a tuple of the form $(A, -, -, -, -, -)$ will be in the $\langle i+1, j+1 \rangle^{th}$ element of $L_{i,j}$. Thus these steps take at most $O((j-i)^2)$ time. Similarly, for a fixed value of i, j, and k, steps 2a and 2b can add at most $O((j-i)^2)$ distinct tuples. However, in these steps $O((j-i)^4)$ not necessarily distinct tuples may be considered. There are $O((j-i)^4)$ such tuples because the integers p, q, r, s can take values in the range between i and j. Thus steps 2a and 2b may each take $O((j-i)^4)$ time for a fixed value of i, j, k. Since we have three initial loops for i, j, and k, the time complexity of the algorithm is $O(n^7)$ where the length of the input is n.

3 Combinatory Categorial Grammars

CCG [9,8] is an extension of Classical Categorial Grammars in which both function composition and function application are allowed. In addition, forward and backward slashes are used to place conditions concerning the relative ordering of adjacent categories that are to be combined.

Definition 3.1 A CCG, G, is denoted by (V_T, V_N, S, f, R) where

V_T is a finite set of terminals (lexical items),
V_N is a finite set of nonterminals (atomic categories),
S is a distinguished member of V_N,
f is a function that maps elements of $V_T \cup \{\epsilon\}$ to finite subsets of $C(V_N)$, the set of categories,[3] where $C(V_N)$ is the smallest set such that $V_N \subseteq C(V_N)$ and $c_1, c_2 \in C(V_N)$ implies $(c_1/c_2), (c_1\backslash c_2) \in C(V_N)$,
R is a finite set of combinatory rules.

There are four types of combinatory rules involving variables $x, y, z, z_1, \ldots$ over $C(V_N)$ and where $|_i \in \{\backslash, /\}$[4].

1. forward application: $(x/y) \quad y \rightarrow x$
2. backward application: $y \quad (x\backslash y) \rightarrow x$
   For these rules we say that (x/y) is the primary category and y the secondary category.
3. generalized forward composition for some fixed $n \geq 1$:

$$(x/y) \quad (\ldots(y|_1 z_1)|_2 \ldots |_n z_n) \rightarrow (\ldots(x|_1 z_1)|_2 \ldots |_n z_n)$$

[3] Note that f can assign categories to the empty string, ϵ, though, to our knowledge, this feature has not been employed in the linguistic applications of CCG.

[4] There is no type-raising rule although its effect can be achieved to a limited extent since f can assign type-raised categories to lexical items.

4. generalized backward composition for some $n \geq 1$:

$$(\ldots(y|_1 z_1)|_2 \ldots |_n z_n) \quad (x\backslash y) \rightarrow (\ldots(x|_1 z_1)|_2 \ldots |_n z_n)$$

For these rules (x/y) is the primary category and $(\ldots(y|_1 z_1)|_2 \ldots |_n z_n)$ the secondary category.

Restrictions can be associated with the use of each combinatory rule in R. These restrictions take the form of constraints on the instantiations of variables in the rules.

1. The leftmost nonterminal (**target category**) of the primary category can be restricted to be in a given subset of V_N.

2. The category to which y is instantiated can be restricted to be in a given finite subset of $C(V_N)$.

Derivations in a CCG, $G = (V_T, V_N, S, f, R)$, involve the use of the combinatory rules in R. Let $\underset{G}{\Longrightarrow}$ be defined as follows, where Υ_1, $\Upsilon_2 \in (C(V_N) \cup V_T)^*$ and $c, c_1, c_2 \in C(V_N)$.

- If R contains a combinatory rule that has $c_1 c_2 \rightarrow c$ as an instance then

$$\Upsilon_1 c \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 c_1 c_2 \Upsilon_2$$

- If $c \in f(a)$ for some $a \in V_T \cup \{\epsilon\}$ and $c \in C(V_N)$ then

$$\Upsilon_1 c \Upsilon_2 \underset{G}{\Longrightarrow} \Upsilon_1 a \Upsilon_2$$

The string languages generated by a CCG, G, $L(G) = \{ w \mid S \overset{*}{\underset{G}{\Longrightarrow}} w \mid w \in V_T^* \}$.

In the present discussion of CCG recognition we make the following assumptions concerning the form of the grammar.

1. In order to simplify our presentation we assume that the categories are parenthesis-free. *The algorithm that we present can be adapted in a straightforward way to handle parenthesized categories and this more general algorithm is given in [12].*

2. We will assume that the function f does not assign categories to the empty string. This is consistent with the linguistic use of CCG although we have not shown that this is a normal form for CCG.

3.1 The LIG/CCG Relationship

In this section, we describe the relationship between LIG and CCG by discussing how we can construct from any CCG a weakly equivalent LIG. The weak equivalence of LIG and CCG was established in [15]. The purpose of this section is to show how a CCG recognition algorithm can be derived from the algorithm given above for LIG.

Given a CCG, $G = (V_T, V_N, S, f, R)$, we construct an equivalent LIG, $G' = (V_T, V_N, V_N \cup \{/, \backslash\}, S, P)$, as follows. Each category in $c \in C(V_N)$ can be represented in G' as a nonterminal and associated stack $A[\alpha]$ where A is the target category of c and $\alpha \in (\{/, \backslash\} V_N)^*$ such that $A\alpha = c$. Note that we are assuming that categories are parenthesis-free.

We begin by considering the function, f, which assigns categories to each element of V_T. Suppose that $c \in f(a)$ where $c \in C(V_N)$ and $a \in V_T$. We should include the production $A[\alpha] \rightarrow a$ where $c = A\alpha$ in P. For each combinatory rule in R we may include a number of productions in P. From the definition of CCG it follows that the length of all secondary categories in the rules R is bounded by some constant. Therefore there are a finite number of possible ground instantiations of the secondary category in each rule. Thus we can remove variables in secondary categories by expanding the number of rules in R. The rules that result will involve a secondary category $c \in C(V_N)$ and a primary category of the form x/A or $x\backslash A$ where $A \in V_N$ is the target category of c. The rule may also place a restriction on the value of the target category of x. In the case of the primary categories of the combinatory rules there is no bound on their length and we cannot remove the variable that will be bound to the unbounded part of the category (the variable x above). Therefore the rules contain a single variable and are linear with respect to this variable; i.e., it appears once on either side of the rule.

It is straightforward to convert combinatory rules in this form into corresponding LIG productions. We illustrate how this can be done with an example. Suppose we have the following combinatory rule.

$$x/A \quad A/B\backslash C\backslash B \rightarrow x/B\backslash C\backslash B$$

where the target category of x must be either C or D. This is converted into the following two productions in P.

$$C[\cdot\cdot/B\backslash C\backslash B] \rightarrow C[\cdot\cdot/A] \quad A[/B\backslash C\backslash B] \qquad\qquad D[\cdot\cdot/B\backslash C\backslash B] \rightarrow D[\cdot\cdot/A] \quad A[/B\backslash C\backslash B]$$

Notice that these LIG productions do not correspond precisely to our earlier definition. We are pushing and popping more that one symbol on the stack and we have not associated empty stacks with all but one of the RHS nonterminals. Although this clearly does not affect weak generative power, as we will see in the next section, it will require a modification to the recognition algorithm given earlier for LIG.

3.2 Recognition of CCG

In order to produce a CCG recognition algorithm we extend the LIG recognition algorithm given in Section 2.2. From the previous section it should be clear that the CCG and LIG algorithms will be very similar. Therefore we do not present a detailed description of the CCG algorithm. We use an array, C, with n^2 elements, $C_{i,j}$ for $1 \leq i \leq j \leq n$. The tuples in the array will have a slightly different form from those of the LIG algorithm. This is because each derivation step may depend on more than one symbol of the category (stack). The number of such symbols is bounded by the grammar and is equal to the number of symbols in the longest secondary category. We define this bound for a CCG, $G = (V_T, V_N, S, f, R)$ as follows. Let $l(c) = k$ if $c \in (\{/, \backslash\}V_N)^k$. Let $s(G)$ be the maximum $l(c)$ of any category $c \in C(V_N)$ such that c can be the secondary category of a combinatory rule in R.

As in the LIG algorithm we do not store the entire category explicitly. However, rather than storing only the top symbol locally, as in the LIG algorithm, we store some bounded number of symbols locally together with a indication of where in C the remainder of the category can be found. This modification is needed since at each step in the recognition algorithm we may have to examine the top $s(G)$ symbols of a category. Without this extension we would be required to trace through $c(G)$ entries in C in order to examine the top $c(G)$ symbols of a category and the algorithm's time complexity would increase.

An entry in C will be a six-tuple of the form $(A, \alpha, \beta, \gamma, p, q)$ where $A \in V_N$, $\alpha, \beta \in (\{/, \backslash\}V_N)^*$ and one of the two cases applies.

$$\text{or} \qquad 2 \leq l(\alpha) \leq s(G) - 1, \quad l(\beta) = s(G) - 1, \quad \gamma \in \{/, \backslash\}V_N, \quad 1 \leq p \leq q \leq n$$

$$0 \leq l(\alpha) < 2s(G) - 2, \quad \beta = \epsilon, \quad \gamma = p = q = -$$

An entry $(A, \alpha, \beta, \gamma, p, q)$ is placed in $C_{i,j}$ when

- If $\beta = \epsilon$ and $\gamma = p = q = -$ then $A\alpha \overset{*}{\underset{G}{\Longrightarrow}} a_i \ldots a_j$.
- If $\beta \neq \epsilon$ then for some $\alpha' \in (\{/, \backslash\}V_N)^*$, $A\alpha'\beta\alpha \overset{*}{\underset{G}{\Longrightarrow}} a_i \ldots a_j$ and $A\alpha'\beta\gamma \overset{*}{\underset{G}{\Longrightarrow}} a_p \ldots a_q$.

The steps of the algorithm that apply for examples of forward application and forward composition are as follows.

- $x/A \quad A \rightarrow x \in R$
  For each k between i and j, we look for $(B, \alpha, \beta, \gamma, p, q) \in C_{i,k}$ and $(A, \epsilon, \epsilon, -, -, -) \in C_{k+1,j}$ where B is a possible target category of x and the string $\beta\alpha$ has $/A$ as a suffix. If we find these tuples then do the following.

  If $l(\alpha) \geq 3$ or $\beta = \epsilon$ then include $(B, \alpha', \beta, \gamma, p, q)$ in $C_{i,j}$ where $\alpha = \alpha'/A$

  If $l(\alpha) = 2$ and $\beta \neq \epsilon$ then look in $C_{p,q}$ for some $(B, \alpha', \beta', \gamma', r, s)$ such that β is a suffix of $\beta'\alpha'$, and include $(B, \alpha'''\alpha'', \beta', \gamma', r, s)$ in $C_{i,j}$ where $\alpha = \alpha''/A$ and $\alpha' = \alpha'''\gamma$.

  If $l(\alpha) = /A$ then we know that $\beta = \epsilon$ and $\gamma = p = q = -$, and we should add $(B, \epsilon, \epsilon, -, -, -)$ in $C_{i,j}$.

- $x/A \quad A\backslash B/C \rightarrow x\backslash B/C \in R$
  For each k between i and j, we look for $(A', \alpha, \beta, \gamma, p, q) \in C_{i,k}$ and $(A, \backslash B/C, \epsilon, -, -, -) \in C_{k+1,j}$ where A' is a possible target category of x and $/A$ is a suffix of $\beta\alpha$. If we find these tuples then do the following.

  If $l(\beta) = s(G) - 1$ or $l(\alpha) = 2s(G) - 3$ then include $(A', \backslash B/C, \beta', /A, i, k)$ in $C_{i,j}$ where β'/A is a suffix of $\beta\alpha$ such that $l(\beta') = s(G) - 1$.

  If $l(\beta) = 0$ and $l(\alpha) < 2s(G) - 3)$ then include $(A', \backslash B/C\alpha', \epsilon, -, -, -)$ in $C_{i,j}$ where $\alpha'/A = \beta\alpha$.

Each of the other forms of combinatory rules can be treated in a similar way yielding an algorithm that closely resembles the LIG algorithm presented in Section 2.2. Note that in a complete algorithm, the forward composition example that we have considered here would have to be made more general since the number of cases that must be considered depends on the length of the secondary category in the rule. The time complexity of the full CCG recognition algorithm is the same as that of the LIG algorithm; i.e., $\mathcal{O}(n^7)$.

4 Importance of Linearity

The recognition algorithms given here have polynomial-time complexity because each array element (e.g., $L_{i,j}$ in LIG recognition) contains a polynomial number of tuples (with respect to the difference between j and i). These tuples encode the top symbol of the stack (or top symbols of the category) together with an indication of where the next part of the stack (category) can be found. If we had stored the entire stack in the array elements[5], then each array entry could include exponentially many elements. The recognition complexity would then be exponential.

It is interesting to consider why it is not necessary to store the entire stack in the array elements. Suppose that $(A, \gamma, A', \gamma', p, q) \in L_{i,j}$. This indicates the existence of a tuple, say $(A', \gamma', A'', \gamma'', r, s)$, in $L_{p,q}$. It is crucial to note that when we are adding the first tuple to $L_{i,j}$ we are not concerned about how the second tuple came to be put in $L_{p,q}$. This is because the productions in LIG (combinatory rules in CCG) are *linear* with respect to their unbounded stacks (categories). Hence the derivations from different nonterminals and their associated stacks (categories) are *independent* of each other. In Indexed Grammars, productions can have the form $A[\cdot\cdot\gamma] \rightarrow A_1[\cdot\cdot]\, A_2[\cdot\cdot]$. In such productions there is no single *distinguished* child that inherits the unbounded stack from the nonterminal in the LHS of the production. In a bottom-up recognition algorithm the identity of the entire stacks associated with A_1 and A_2 has to be verified. This nullifies any advantage from the sharing of stacks since we would have to examine the complete stacks. A similar situation arises in the case of coordination schema used to handle certain forms of coordination in Dutch. A coordination schema has been used by Steedman [9] that has the form $x\ conj\ x \rightarrow x$ where the variable x can be any category. With this schema we have to check the identity of two derived categories. This results in the loss of *independence* among paths in derivation trees. In [13] we have discussed the notion of independent paths in derivation trees with respect to a range of grammatical formalisms. We have shown [12] that when CCG are extended with this coordination schema the recognition problem becomes NP-complete.

5 Conclusion

We have presented a general scheme for polynomial-time recognition of languages generated by a class of grammatical formalisms that are more powerful than CFG. This class of formalisms, which includes LIG, CCG, and TAG, derives more complex trees than CFG due the use of an additional stack-manipulating mechanism. Using constructions given in [15,3], we have described how a recognition algorithm presented for LIG can be adapted to give an algorithm for CCG. These are the first polynomial recognition algorithms that work directly with these formalisms. This approach can also be used to yield TAG recognition algorithm, although the TAG algorithm is not discussed in this paper. A similar approach has been independently taken by Lang [5] who presents a Earley parser for TAG that appears to be very closely related to the algorithms presented here.

[5] In the chart parser for CCG given by Pareschi and Steedman [6] the entire category is stored explicitly in each chart entry.

References

[1] A. V. Aho. Indexed grammars — An extension to context free grammars. *J. ACM*, 15:647–671, 1968.

[2] G. Gazdar. *Applicability of Indexed Grammars to Natural Languages.* Technical Report CSLI-85-34, Center for Study of Language and Information, 1985.

[3] A. K. Joshi, K. Vijay-Shanker, and D. J. Weir. The convergence of mildly context-sensitive grammar formalisms. In T. Wasow and P. Sells, editors, *The Processing of Linguistic Structure*, MIT Press, 1989.

[4] T. Kasami. *An Efficient Recognition and Syntax Algorithm for Context-Free Languages.* Technical Report AF-CRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

[5] B. Lang. *Nested Stacks and Structure Sharing in Earley Parsers.* In preparation.

[6] R. Pareschi and M. J. Steedman. A lazy way to chart-parse with categorial grammars. In 25th *meeting Assoc. Comput. Ling.*, 1987.

[7] C. Pollard. *Generalized Phrase Structure Grammars, Head Grammars and Natural Language.* PhD thesis, Stanford University, 1984.

[8] M. Steedman. Combinators and grammars. In R. Oehrle, E. Bach, and D. Wheeler, editors, *Categorial Grammars and Natural Language Structures*, Foris, Dordrecht, 1986.

[9] M. J. Steedman. Dependency and coordination in the grammar of Dutch and English. *Language*, 61:523–568, 1985.

[10] J. W. Thatcher. Characterizing derivations trees of context free grammars through a generalization of finite automata theory. *J. Comput. Syst. Sci.*, 5:365–396, 1971.

[11] K. Vijay-Shanker and A. K. Joshi. Some computational properties of tree adjoining grammars. In 23rd *meeting Assoc. Comput. Ling.*, pages 82–93, 1985.

[12] K. Vijay-Shanker and D. J. Weir. The computational properties of constrained grammar formalisms. In preparation.

[13] K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. Characterizing structural descriptions produced by various grammatical formalisms. In 25th *meeting Assoc. Comput. Ling.*, 1987.

[14] K. Vijay-Shanker, D. J. Weir, and A. K. Joshi. Tree adjoining and head wrapping. In 11th *International Conference on Comput. Ling.*, 1986.

[15] D. J. Weir and A. K. Joshi. Combinatory categorial grammars: Generative power and relationship to linear context-free rewriting systems. In 26th *meeting Assoc. Comput. Ling.*, 1988.

[16] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Inf. Control*, 10(2):189–208, 1967.

Handling of Ill-designed Grammars in Tomita's Parsing Algorithm

R. Nozohoor-Farshi

School of Computer Science
University of Windsor, Windsor, Canada N9B 3P4

ABSTRACT

In this paper, we show that some non-cyclic context-free grammars with ε-rules cannot be handled by Tomita's algorithm properly. We describe a modified version of the algorithm which remedies the problem.

1. Introduction

Tomita's parsing algorithm [8,9] is an efficient all-paths parsing method which is driven by an LR parse table with multi-valued entries. The parser employs an acyclic parse graph instead of the conventional LR parser stack. The parser starts as an ordinary LR parser, but splits up when multiple actions are encountered. Multiple parses are synchronized on their shift actions and are joined whenever they are found to be in the same state.

The parallel parsing of all possible paths makes this algorithm suitable for parsing nearly all the arbitrary context-free grammars. In fact, one may view this method as a precompiled form of Earley's algorithm [2,3]. Earley [2] proposed a form of precompiled approach to his method in the case of a restricted class of grammars which has undecidable membership. Tomita's algorithm, on the other hand, is intended for use with general grammars. Since the method uses a parse table, it achieves considerable efficiency over the Earley's non-compiled method which has to compute a set of LR items at each stage of parsing. In this respect, Tomita's algorithm can indeed be considered as a breakthrough in efficient parallel parsing in practical systems. However, there seem to be at least two types of context-free grammars that cannot be handled by this method properly. The first type are cyclic grammars. These grammars have infinite ambiguity and therefore have to be excluded from syntactic analyses. The second kind of grammars include certain context-free grammars with ε-productions. Some of these are unambiguous and some have bounded, bounded direct or unbounded degrees of ambiguity.

Grammars of the latter type may seldom be used to describe the syntax of natural language. In fact, we consider them as somewhat ill-designed. But, they may creep in easily when one is designing a natural language grammar with ε-rules. Such rules cause unexpected infinite loops in parsing. In this paper, we modify the parsing algorithm so that it can handle the second type grammars.

The modification introduces cyclic subgraphs in the original graph-structured parse stack. These subgraphs correspond to the parsing of null substrings in the input sentence. Thus, the modification incurs no cost to the grammars or the inputs that do not need this feature. We believe that adding such a feature to Tomita's algorithm is very desirable. Because, it enriches the method to be comparable to Earley's algorithm in its coverage, and yet it is in a precompiled form.

In the following sections, we discuss the two types of the grammars that cause problems in the original algorithm, and we present the modified algorithm.

2. The Two Types of Grammars

Cyclic grammars are those in which a non-terminal, like A, can derive itself (i.e., $A \overset{+}{\Longrightarrow} A$). G_1 and G_2 are examples of cyclic grammars.

G_1:	G_2:
$S \rightarrow A$	$S \rightarrow S\ S$
$A \rightarrow S$	$S \rightarrow x$
$A \rightarrow x$	$S \rightarrow \varepsilon$

In G_1, $A \Longrightarrow S \Longrightarrow A$, and in G_2, $S \Longrightarrow S\ S \Longrightarrow S$. Cyclic grammars produce infinite number of parse trees for a finite length input such as "x" in L(G_1) and L(G_2). They cause problem in every parsing algorithm. Therefore, they have been avoided in describing syntax of languages traditionally.

Both Earley's and Tomita's algorithms will fail to detect the cyclicity of G_1 and G_2. Given an input sentence "x", one can however obtain the minimal parses with respect to either grammar by Earley's algorithm and only with respect to G_1 by Tomita's algorithm. The second algorithm will not terminate when the grammar G_2 is used. Tomita [8] discusses the cyclic grammars and rules out their inclusion in natural language parsing. Such exclusion can be achieved through a simple test before generating a parse table (see [1] for example).

Among the second kind grammars that cannot be handled with the original algorithm are the examples G_3, G_4, G_5 and G_6 below.

G_3:	G_5:
$S \rightarrow A\ S\ b$	$S \rightarrow A\ S\ b$
$S \rightarrow x$	$S \rightarrow x$
$A \rightarrow \varepsilon$	$A \rightarrow t$
	$A \rightarrow \varepsilon$

G_4:	G_6:
$S \rightarrow M$	$S \rightarrow M\ N$
$S \rightarrow N$	$M \rightarrow A\ M\ b$
$M \rightarrow A\ M\ b$	$M \rightarrow x$
$M \rightarrow x$	$N \rightarrow b\ N\ A$
$N \rightarrow A\ N\ b$	$N \rightarrow x$
$N \rightarrow x$	$A \rightarrow \varepsilon$
$A \rightarrow \varepsilon$	

G_3 is unambiguous, G_4 has bounded ambiguity, G_5 has bounded direct ambiguity while G_6 has unbounded ambiguity (see Apendix 1 for the definition of these terms). One may note that in these grammars, unlike cyclic grammars, there are only finite number of parse trees for a given finite length input.

A property common to these grammars is that there exists a non-terminal, say S, such that $S \overset{+}{\Longrightarrow} \alpha\ S\ \beta$ where $\alpha \overset{+}{\Longrightarrow} \varepsilon$ but $\beta \overset{*}{\not\Longrightarrow} \varepsilon$. For example, in G_3 or G_5, S can be rewritten as $S \Longrightarrow A\ S\ b \Longrightarrow S\ b$. Rules like these may be excluded from a grammar by using an appropriate test (see Appendix 2). However, one may keep or include such rules in a grammar for the following reasons.

(1) To capture some rare phenomena, for example, embedded that-sentences [[THAT [[THAT . . . [[THAT S] VP] . . .] VP]] VP] in which a number of terminal 'that's are omitted.

(2) Grammars with ε-productions are more concise and readable than the grammars without ε-rules. In fact, elimination of ε-rules from a grammar may increase the size of the grammar exponentially. Therefore, one may use rules similar to the examples G_3 to G_6 to compact the grammar and the parse table, knowing that their presence should not affect the correct parsing of valid inputs.

(3) More frequently, such rules may appear in a grammar when ε-productions are introduced without an adequate care. It is important to note that replacement of these rules (and their associated symbols) may not always be easy.

Grammars G_3 through G_6 can be parsed by Earley's algorithm with no problem. For example, consider the sentence xbbb ∈ L(G_3). That algorithm will produce the following states.

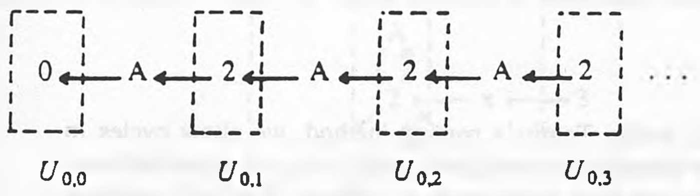
state 0		state 1		state 2		state3	
root → .S#, 0	x →	S → x., 0	b →	S → ASb., 0	b →	S → ASb., 0	b →
S → .ASb, 0		root → S.#, 0		root → S.#, 0		root → S.#, 0	
S → .x, 0		S → AS.b, 0		S → AS.b,0		S → AS.b,0	
A → ε., 0							
S → A.Sb, 0							

state 4		state 5
S → ASb., 0		
root → S.#, 0	# →	root → S#., 0
S → AS.b, 0		

However, the above grammars cause an infinite loop in Tomita's algorithm. Applying the algorithm for ε-grammars (given in [8]) to the input sentence xbbb and the parse table for G_3, the result will be an infinite graph-structured stack as shown below.

State	x	b	#	A	S
0	re3,sh3			2	1
1			acc		
2	re3,sh3			2	4
3		re2	re2		
4		sh5			
5		re1	re1		
	Action table			Goto table	

Grammar G_3:
(1) S → A S b
(2) S → x
(3) A → ε



In Tomita's algorithm the state nodes created in the parse graph are partitioned into U_0, U_1, . . . , U_n where each U_i is the set of state vertices which are created before shifting of word a_{i+1} in the input. Furthermore, in the presence of ε-productions, each U_i is partitioned into $U_{i,0}$, $U_{i,1}$, $U_{i,2}$, ... Each $U_{i,j}$ denotes the set of state vertices created while parsing the j-th null construct after the i-th input symbol a_i is shifted and before the shifting of next actual input symbol a_{i+1} takes place. Tomita assumes that the number of null constituents between every adjacent pair of input symbols is always finite. Though his assumption is correct for non-cyclic grammars, it cannot be incorporated as such in the parser since it will require arbitrary and complex lookaheads in general case. As noted earlier this strategy fails in the example grammars.

It is interesting to note that the same strategy will succeed in the case of LR grammar G_3^r which is the reverse of G_3.

G_3^r:
S → b S A
S → x
A → ε

The difference between G_3 and G_3^r is that in G_3 a null deriving constituent appears on the left part of a recursive phrase, while in G_3^r, it appears on the right side of the recursive construct. Thus, the parser for G_3 does not know how many A's it has to create before consuming the first input word "x". In the case of G_3^r, the left context provides enough information to limit the number of empty constructs to a finite size.

One may observe that though G_3 is an unambiguous grammar, it is not LR(k) for any k. Viewing differently, one may argue that such grammars can be parsed deterministically and more efficiently by non-canonical parsers. Marcus' parser [5] and bottom-up variations of it described in [6,7] can handle this grammar in a much better way, since they create the rightmost A in the parse tree first. The reader may also consult [6,7] to see the advantage of these parsers over Tomita's algorithm when grammars like G_7 are to be parsed.

G_7:
S → a S a
S → B S b
S → C S c
B → a
C → a
S → x

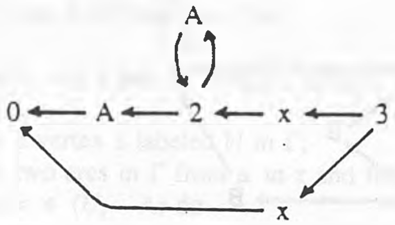
However, we should emphasis that the whole thrust and advantage of Tomita's parser lies in obtaining multiple parses with respect to ambiguous grammars such as those in examples G_4 to G_6.

In the following section, we modify Tomita's algorithm in a way that the second type grammars can be handled within this framework. In doing so, we believe that we are introducing a version of Tomita' algorithm which is a partially-precompiled equivalent of Earley's parser and can be applied to all non-cyclic context-free grammars.

3. Modified Algorithm

To accommodate grammars like G_3 to G_6 within Tomita's parsing method, we allow cycles in the graph-structured parse stack. These cycles are introduced in the parse graph in a very restricted way. Each cyclic subgraph represents a regular expression that corresponds to parsing of a null substring between two adjacent input symbols. Unlike Tomita's algorithm for ε-grammars [8], we do not partition each U_i any further. So, the set of state vertices of each cyclic subgraph entirely lies within a single U_i. Obviously, cycles are created within U_i only if parsing of the input sentence requires them. Since the parse graph is now cyclic, we do reductions along arbitrary paths (i.e., paths that are not simple and may contain repetitive vertices or arcs). Such paths are usually termed *(directed) walks* in graph theory.

Our approach though is intuitive, it has its roots in LR theory. In LR parsing, the finite automaton (from which a parse table is extracted) represents the set of all viable prefixes of the grammar in closed form. The parse stack, on the other hand, represents an actual viable prefix (of a right sentential form) in open form. The actual viable prefix is built from the input symbols which are consumed by the LR parser. It is necessary to hold the actual viable prefix in the stack so that the parser can be provided with the exact left context. However, in the modified all-paths parser we do not need to keep the null-deriving segments of the left context in open form. For example, in parsing sentences like xb. . .b $\in L(G_3)$, ε and A. . .A are the viable prefixes when the parser scans the first input symbol "x". Since each A derives a null string and we do not know exactly how many of them we should assume, we represent the left context in the closed form $\varepsilon + AA^*$. The corresponding parse graph will appear as the figure in below when "x" is just shifted. The parser will pick as many A's as it needs from this regular expression when the remainder of the sentence is seen.



Similarly, consider the example grammar G_5 and the parse table for it as shown below. One will obtain the following snapshot of the parse graph after the parser consumes the prefix txb of the sentence txb. . .b, and all the appropriate reductions are done.

state	t	x	b	#	A	S
0	sh4,re4	sh3,re4			2	1
1				acc		
2	sh4,re4	sh3,re4			2	5
3			re2	re2		
4	re3	re3				
5			sh6			
6			re1	re1		

Action table   Goto table

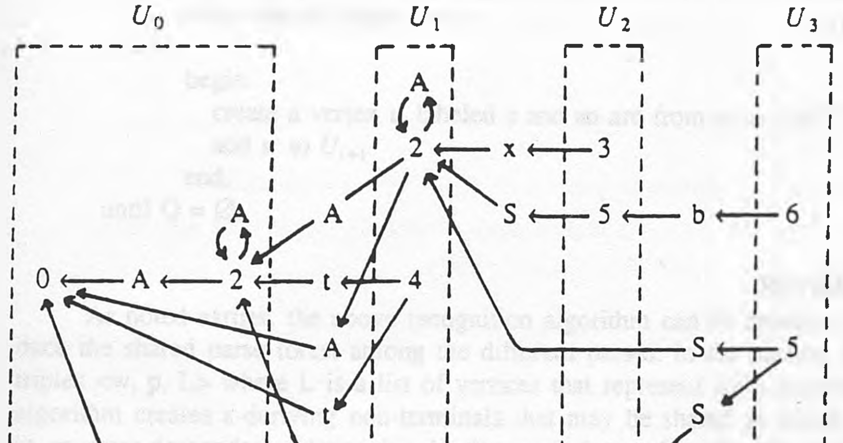
Grammar G_5

(1) $S \rightarrow A\,S\,b$

(2) $S \rightarrow x$

(3) $A \rightarrow t$

(4) $A \rightarrow \varepsilon$



In this example, the left context just before shifting the word "x" can represented as the regular expression (AA* **A** + A) A*. For clarity, the bold faced **A** represents the non-terminal obtained by reducing "t". For the same reason, we are not combining identical symbol vertices which are adjacent to the same state vertex, (a measure of optimization suggested in [8]), in the illustrated examples or in the algorithm that to follow.

As another example, an interested reader using the parse table in Appendix 3 may verify that U_0 for the grammar G_8 will have the following format.

G_8:
S → x
S → B S b
S → A S b
B → A A
A → ε

$$\begin{tikzcd}
 & & & & & & A \arrow[d, bend right] \\
0 & A \arrow[l] & & 4 \arrow[ll] \arrow[dl] & & A \arrow[ll] & 6 \arrow[l] \arrow[u, bend right] \\
 & & A \arrow[dr] & & B \arrow[ul] & & \\
 & B \arrow[uul] & & 3 \arrow[ll] \arrow[rr] \arrow[ur] \arrow[d, bend right] & & B \arrow[uur] & \\
 & & & B \arrow[u, bend right] & & &
\end{tikzcd}$$

In the above examples, we have used an LALR(1) parser generator, similar to YACC [4], to obtain the parse tables with multi-valued entries. Tomita [8,9] also uses LALR(1) tables, however, using non-optimized LR(1) tables will decrease the number of superfluous reductions in general.

We are now in a position to present the modified algorithm. For simplicity, we give an algorithm for a recognizer rather than a parser. The recognizer can be augmented in a way similar to that of [8] to provide a parser that also creates the parse forest.

Recognition Algorithm:

```
PARSE (G, $a_1 \cdots a_n$)
  • $\Gamma := \emptyset$.
  • $a_{n+1}$ := '#'.
  • r := FALSE.
  • Create a vertex $v_0$ labeled $s_0$ in $\Gamma$.
  • $U_0 := \{v_0\}$.
  • For i := 1 to n do PARSEWORD (i).
  • Return r.

PARSEWORD (i)
  • A := $U_i$.
  • R := $\emptyset$; Q := $\emptyset$.
  • Repeat
       if A $\neq \emptyset$ then do ACTOR
       else if R $\neq \emptyset$ then do COMPLETER
    until R = $\emptyset$ and A = $\emptyset$.
  • Do SHIFTER.

ACTOR
  • Remove an element $v$ from A.
  • For all $\alpha \in$ ACTION (STATE ($v$), $a_{i+1}$) do
      begin
       if $\alpha$ = 'accept' then r := TRUE;
```

```
        if $\alpha$ = 'shift s' then add <$v$,s> to Q;
        if $\alpha$ = 'reduce p' then
            for all vertices $w$ such that there exists a directed
            walk of length 2 | RHS (p) | from $v$ to $w$  /* For $\varepsilon$-rules this is a trivial walk, i.e. $w = v$ */
            do add <$w$,p> to R
    end.

COMPLETER
  • Remove an element <$w$,p> from R.
  • N := LHS (p);  s := GOTO (STATE ($w$), N)..
  • If there exists $u \in U_i$ such that STATE($u$) = s then
        begin
            if there does not exist a path of length 2 from $u$ to $w$ then
                begin
                    create a vertex z labeled N in $\Gamma$;
                    create two arcs in $\Gamma$ from $u$ to z and from z to $w$;
                    for all $v \in (U_i$ - A) do
                    /* In the case of non-$\varepsilon$-grammars this loop executes for $v = u$ only */
                      for all q such that 'reduce q' $\in$ ACTION (STATE ($v$), $a_{i+1}$) do
                        for all vertices $t$ such that there exists a directed walk of
                        length 2 | RHS (q) | from $v$ to $t$ that goes through vertex z
                        do add <$t$,q> to R
                end
        end
    else /* i.e., when there does not exist $u \in U_i$ such that STATE ($u$) = s */
        begin
            create in $\Gamma$ two vertices $u$ and z labeled s and N respectively;
            create two arcs in $\Gamma$ from $u$ to z and from z to $w$;
            add $u$ to both A and $U_i$
        end.

SHIFTER
  • $U_{i+1} := \emptyset$.
  • Repeat
        remove an element <$v$,s> from Q;
        create a vertex x labeled $a_{i+1}$ in $\Gamma$;
        create an arc from x to $v$;
        if there exists a vertex $u \in U_{i+1}$ such that STATE ($u$) = s then
            create an arc from $u$ to x
        else
            begin
              create a vertex $u$ labeled s and an arc from $u$ to x in $\Gamma$;
              add $u$ to $U_{i+1}$
            end.
    until Q = $\emptyset$.
```

As noted earlier, the above recognition algorithm can be changed into a parsing algorithm to produce the shared parse forest among the different parses. In the parsing algorithm the elements of R are triples <w, p, L> where L is a list of vertices that represent RHS symbols of p. One must note that our algorithm creates ε-deriving non-terminals that may be shared as a son by other non-terminals that are in ancestor-descendant relationship in the parse forest. To illustrate this point, we show the full parse graphs and corresponding parse trees of example sentences in Appendix 4. As an alternative, in building a parse forest one may replicate a null yielding subtree whenever this subtree participates in a reduction where at least one other sibling has non-empty yield.

As a final remark, we may add that the above algorithm can obtain the minimal parses in the case of cyclic grammars, but does not detect their cyclicity. It is also possible to precompile some subsets of each U_i that are obtained under the transitions with respect to null-deriving non-terminals.

4. Conclusion

We have modified Tomita's parsing algorithm so that it can handle some ill-designed grammars with ε-rules that caused a problem in the original algorithm. We have introduced cycles in the parse graph in a restricted way. This makes the parse graph in the new algorithm a cyclic directed graph in some general cases. However, the new algorithm works exacltly like the original one in case of grammars that have no ε-productions. This algorithm has no extra costs beyond that of the original algorithm.

We believe that the modified algorithm is a precompiled equivalent of Earley's algorithm with respect to its coverage, though we have not provided a formal proof for it. The resulting algorithm suggests that Tomita's graph-structured parsing approach can be used with a broader class of context-free grammars.

Appendix 1: Ambiguous grammars

Definition: A context-free grammar G has bounded ambiguity of degree k if each sentence in L(G) has at most k distinct derivation trees.

Definition: A context-free grammar G has unbounded ambiguity if for each i≥1, there exists a sentence in L(G) which has at least i distinct derivation trees.

Definition: The degree of direct ambiguity of a non-terminal A with respect to a string x is the number of distinct tuples $(p, x_1, x_2, \ldots, x_n)$, where p is a production $A \rightarrow B_1 B_2 \cdots B_n$, and $x_1 x_2 \cdots x_n = x$ is a factorization of x such that $B_i \overset{*}{\Longrightarrow} x_i$ for $1 \leq i \leq n$.

Definition: A context-free grammar has bounded direct ambiguity of degree k if the degree of direct ambiguity of any of its non-terminals with respect to any string is at most k.

For example, the grammar G_5 has direct ambiguity of degree 2, in spite of being unboundedly ambiguous.

Appendix 2: Identifying the ε-grammars that cannot be parsed by the original algorithm.

Let G = (N , T , P , S) be a context-free grammar with ε productions. The following algorithm decides whether G can be parsed by the original algorithm.

(1) Compute the set of non-terminals $E = \{ C \mid C \overset{+}{\Longrightarrow} \varepsilon \}$ that can derive a null string.

(2) Let $\rho \subset N \times N$ be a binary relation such that $(A, B) \in \rho$ if and only if $A \rightarrow C_1 C_2 \cdots C_n B \alpha$ is a production in P and $C_i \in E$ for $1 \leq i \leq n$.

(3) Compute ρ^+ the closure of ρ. If there exists a non-terminal A where $(A, A) \in \rho^+$ then G cannot be parsed by the Tomita's original algorithm for ε-grammars.

Appendix 3: Parse Table for Grammar G_3

state	x	b	#	A	B	S
0	sh2,re5			4	3	1
1			acc			
2		re1	re1			
3	sh2,re5			4	3	5
4	sh2,re5			6	3	7
5		sh8				
6	sh2,re4,re5			6	3	7
7		sh9				
8		re2	re2			
9		re3	re3			

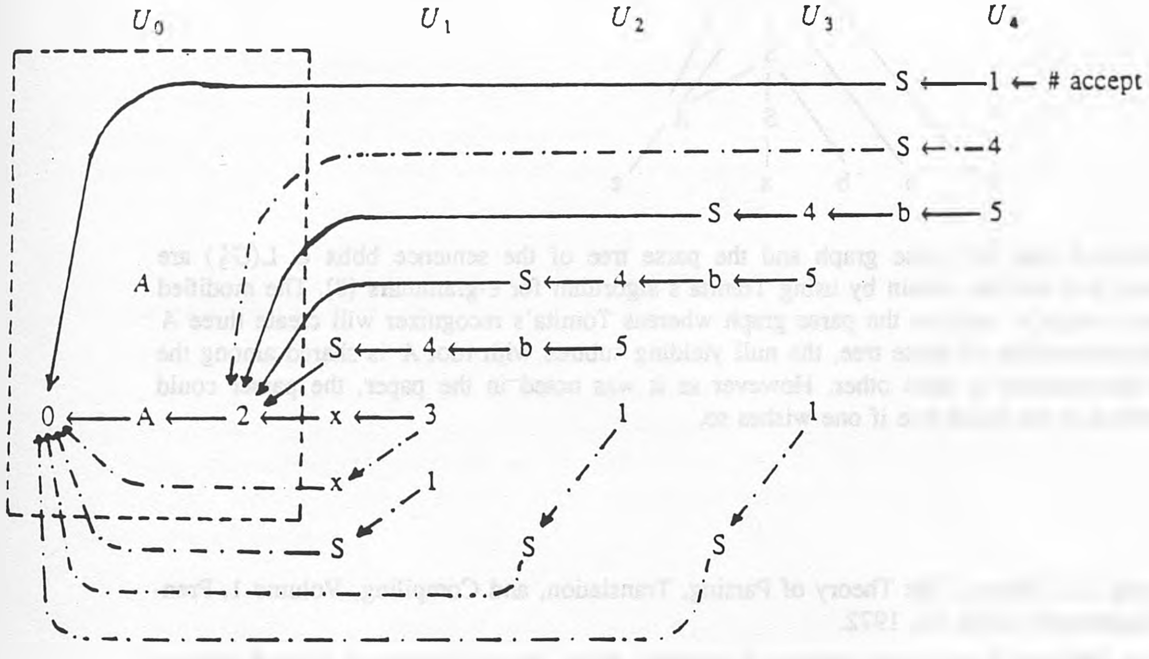
Action table   Goto table

Grammar G_3

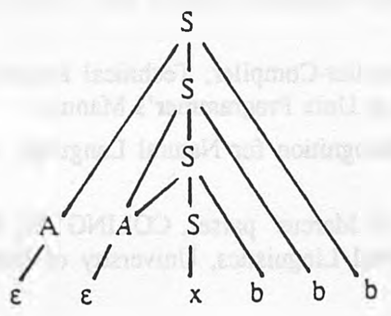
(1) $S \rightarrow x$
(2) $S \rightarrow B\ S\ b$
(3) $S \rightarrow A\ S\ b$
(4) $B \rightarrow A\ A$
(5) $A \rightarrow \varepsilon$

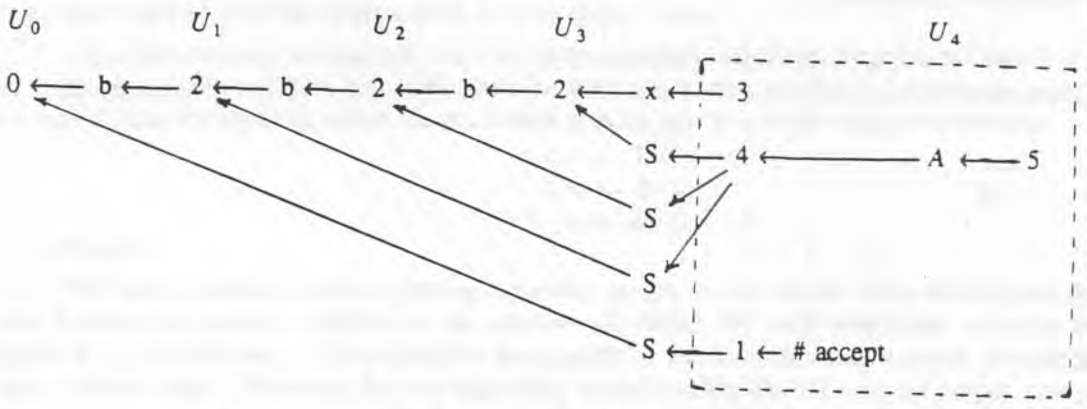
Appendix 4: Parsing of example sentences

The following figures illustrate parsing of the sentences xbbb $\in$ L(G_3) and bbbx $\in$ L(G_3'). The dotted lines indicate the rejected paths. The shared non-terminals are shown in italics.

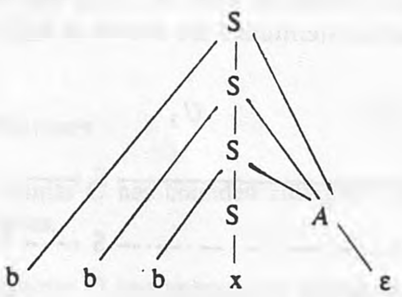


Parse graph and parse tree of the sentence xbbb $\in$ L(G_3)





Parse graph and parse tree of the sentence bbbx $\in$ L(G_3')



One may observe that the parse graph and the parse tree of the sentence bbbx $\in$ L(G_3') are different from those that one can obtain by using Tomita's algorithm for ε-grammars [8]. The modified recognizer creates a single *A* node in the parse graph whereas Tomita's recognizer will create three *A* vertices. In our representation of parse tree, the null yielding subtree with root *A* is shared among the S nodes that are descendants of each other. However as it was noted in the paper, the parser could replicate such subtrees in the parse tree if one wishes so.

References

[1] A.V. Aho and J.D. Ullman, The Theory of Parsing, Translation, and Compiling, Volume 1, Prentice Hall, Englewood Cliffs, NJ, 1972.

[2] J. Earley, An Efficient Context-free Parsing Algorithm, Ph.D. Thesis, Computer Science Department, Carnegie-Mellon University, Pittsburg, PA, 1968.

[3] J. Earley, An efficient context-free parsing algorithm, CACM, vol. 13, no. 2, pp. 94-102, February 1970.

[4] S.C. Johnson, YACC: Yet Another Compiler-Compiler, Technical Report 32, Bell Laboratories, Murray Hill, NJ, 1975. Also reproduced in Unix Programmer's Manual.

[5] M.P. Marcus, A Theory of Syntactic Recognition for Natural Language, MIT Press, Cambridge, MA, 1980.

[6] R. Nozohoor-Farshi, On formalizations of Marcus' parser, COLING' 86, Proceedings of the 11th International Conference on Computational Linguistics, University of Bonn, West Germany, pp. 533-535, August 1986.

[7] R. Nozohoor-Farshi, LRRL(k) Grammars: A Left to Right Parsing Technique with Reduced Lookaheads, Ph.D. Thesis, Department of Computing Science, University of Alberta, Edmonton, Canada, 1986.

[8] M. Tomita, Efficient Parsing for Natural Language, Kluwer Academic Publishers, Boston, MA, 1986.

[9] M. Tomita, An efficient augmented-context-free parsing algorithm, Computational Linguistics, vol. 13, no. 1-2, pp. 31-46, January 1987.

Acknowledgement

The research reported in this paper was supported by the Natural Sciences and Engineering Research Council of Canada grant A9447.

ANALYSIS OF TOMITA'S ALGORITHM FOR GENERAL CONTEXT-FREE PARSING[1]

JAMES R. KIPPS (KIPPS@RAND-UNIX.ARPA)
The RAND Corporation, Santa Monica, CA 90406

Abstract. A variation on Tomita's algorithm is analyzed in regards to its time and space complexity. It is shown to have a general time bound of $O(n^{\tilde{\rho}+1})$, where n is the length of the input string and $\tilde{\rho}$ is the length of the longest production. A modified algorithm is presented in which the time bound is reduced to $O(n^3)$. The space complexity of Tomita's algorithm is shown to be proportional to n^2 in the worst case and is changed by at most a constant factor with the modification. Empirical results are used to illustrate the trade off between time and space on a simple example. A discussion of two subclasses of context-free grammars that can be recognized in $O(n^2)$ and $O(n)$ is also included.

1. INTRODUCTION

Algorithms for general context-free (CF) parsing, e.g., Earley's algorithm (Earley, 1968) and the Cocke-Younger-Kasami algorithm (Younger, 1967), are necessarily less efficient than algorithms for restricted CF parsing, e.g., the LL, operator precedence, and LR algorithms (Aho and Ullman, 1972), because they must simulate a multi-path, nondeterministic pass over their inputs using some form of search, typically, goal-driven. While many of the general algorithms can be shown to theoretically perform as well as the restricted algorithms on a large subclass of CF grammars, due to the inefficiency of goal expansion the general algorithms have not been widely used as practical parsers for programming languages.

A basic characteristic shared by many of the best known general algorithms is that they are top-down parsers. Recently, Tomita (1985) introduced an algorithm for general CF parsing defined as a variation on standard LR parsing, i.e., a table-driven, bottom-up parsing algorithm. The benefit of this approach is that it eliminates the need to expand alternatives of a nonterminal at parse time (what Earley refers to as the predictor operation). For Earley's algorithm, the predictor operation is one of two $O(n^2)$ components. While eliminating this operation would not change the algorithm's time bound of $O(n^3)$, it could be significant to practical parsing. It is of interest to analyze the complexity of Tomita's algorithm and see how it compares.

Upon examination, Tomita's algorithm is found to have a general time complexity of $O(n^{\tilde{\rho}+1})$, where n is as before and $\tilde{\rho}$ is the length of the longest production in the source grammar. Thus, this algorithm achieves $O(n^3)$ for grammars in Chomsky normal form (Chomsky, 1959) but has potential for being worse when productions are of unrestricted lengths. In this paper, I present a modification of Tomita's algorithm that allows it to run in time proportional to n^3 for grammars with productions of arbitrary lengths.

2. TOMITA'S ALGORITHM

The following is an informal description of Tomita's algorithm as a recognizer; familiarity with standard LR parsing is assumed. Tomita views his algorithm as a variation on standard LR parsing. The algorithm takes a shift-reduce approach, using an extended LR parse table to guide its actions. The extended parse table records shift/reduce and reduce/reduce conflicts as multiple action entries, so the parse table can no longer be used for strictly deterministic parsing. The algorithm simulates a nondeterministic parse with pseudo-parallelism. It scans an input string $x_1 \cdots x_n$ from left to right, following all paths in a breath-first manner and merging like subpaths when possible to avoid redundant computations.

[1] This work was supported by the Defense Advanced Research Projects Agency, under contract number MDA-903-85-C-0030.

The algorithm operates by maintaining a number of parsing processes in parallel. Each *process* has a stack, scans the input string from left-to-right, and behaves basically the same as the single parsing process in standard LR parsing. Each stack element is labeled with a parse state and points to its parent, i.e., the previous element on a process's stack. The top-of-stack is the *current state* of a process.

Each process does not actually maintain its own separate stack. Rather, these "multiple" stacks are represented using a single directed acyclic (but reentrant) graph called a *graph-structured stack*. Each stack element corresponds to a vertex of the graph. Each leaf of the graph acts as a distinct top-of-stack to a process. The root of the graph acts as a common bottom-of-stack. The edge between a vertex and its parent is directed toward the parent. Because of the reentrant nature of the graph (as explained below), a vertex may have more than one parent.

The leaves of the graph grow in stages. Each stage U_i corresponds to the ith symbol x_i from the input string. After x_i is scanned, the leaves in stage U_i are in a one-to-one correspondence with the algorithm's *active* processes, where each process references a distinct leaf of the graph and treats that leaf as its current state. Upon scanning x_{i+1}, an active process can either (1) add an additional leaf to U_i, or (2) add a leaf to U_{i+1}. Only processes that have added leaves to U_{i+1} will be active when x_{i+2} is scanned.

In general, a process behaves in the following manner. On x_i, each active process (corresponding to the leaves in U_{i-1}) executes the entries in the action table for x_i given its current state. When a process encounters multiple actions, it *splits* into several processes (one for each action), each sharing a common top-of-stack. When a process encounters an error entry, the process is discarded (i.e., its top-of-stack vertex sprouts no leaves into U_i by way of that process). All processes are synchronized, scanning the same symbol at the same time. After a process shifts on x_i into U_i, it waits until there are no other processes that can act on x_i before scanning x_{i+1}.

The Shift Action. A process (with top-of-stack vertex v) shifts on x_i from its current state s to some successor state s' by

(1) creating a new leaf v' in U_i labeled s';

(2) placing an edge from v' to its top-of-stack v (directed towards v); and

(3) making v' its new top-of-stack vertex (in this way changing its current state).

Any successive process shifting to the same state s' in U_i is *merged* with the existing process to form a single process whose top-of-stack vertex has multiple parents, i.e., by placing an additional edge from the top-of-stack vertex of the existing process in U_i to the top-of-stack vertex of the shifting process. The merge is done because, individually, these processes would behave in exactly the same manner until a reduce action removed the vertices labeled s' from their stacks. Thus, merging avoids redundant computation. Merging also ensures that each leaf in any U_i will be labeled with a distinct parse state, which puts a finite upper-bound on the possible number of active processes and, thus, limits the size of the graph-structured stack.

The Reduce Action. A process executes a reduce action on a production p by following the chain of parent links down from its top-of-stack vertex v to the ancestor vertex from which the process began scanning for p earlier, essentially "popping" intervening vertices off its stack. Since merging means a vertex can have multiple parents, the reduce operation can lead back to multiple ancestors. When this happens, the process is again split into separate processes (one for each ancestor). The ancestors will correspond to the set of vertices at a distance $\bar{p}$ from v, where $\bar{p}$ equals the number of symbols in the right-hand side of the pth production. Once reduced to an ancestor, a process shifts to the state s' indicated in the goto table for D_p (the nonterminal on the left-hand side of the pth production) given the ancestor's state. A process shifts on a nonterminal much as it does a terminal, with the exception that the new leaf is added to U_{i-1} rather than U_i; a process can only enter U_i by shifting on x_i.

The algorithm begins with a single initial process whose top-of-stack vertex is the root of the graph-structured stack. It then follows the general procedure outlined above for each symbol in the input string, continuing until there are either no leaves added to U_i (i.e., no more active processes), which denotes *rejection*, or a process executes the accept action on scanning the $n+1$st input symbol '$\dashv$,' which denotes *acceptance*.

3. ANALYSIS OF TOMITA'S ALGORITHM

In this section, a formal definition of Tomita's algorithm is presented as a recognizer for input string $x_1 \cdots x_n$. This definition is understood to be with respect to an extended LR parse table (with start state S_0) constructed from a source grammar G.

Notation. The productions of G are numbered arbitrarily $1, \cdots, d$, where each production is of the form $D_p \rightarrow C_{p1} \cdots C_{p\bar{p}}$ $(1 \leq p \leq d)$ and where $\bar{p}$ is the number of symbols on the right-hand side of the pth production.

Definition. The entries of the extended LR parse table are accessed with the functions ACTIONS and GOTO.

- ACTIONS(s,x) returns a set of actions from the action table along the row of state s under the column labeled x. This set will contain no more than one of a shift action **sh**s' (shift to state s) *or* an accept action **acc**; it may contain any number of reduce actions **re**p (reduce using production p). An empty action set corresponds to an error.
- GOTO(s,D_p) returns a state s' from the goto table along the row of state s under the column labeled with nonterminal D_p.

Definition. Each *vertex* of the graph-structured stack is a triple $\langle i, s, l \rangle$, where i is an integer corresponding to the ith input symbol scanned (at which point the vertex was created as a leaf), s is a parse state (corresponding to a row of the parse table), and l is a set of parent vertices. The *processes* described in the last section are represented implicitly by the vertices in successive U_i's. The root of the graph-structured stack, and hence the initial process, is the vertex $\langle 0, S_0, \emptyset \rangle$.

The Recognizer. The recognizer is a function of one argument REC($x_1 \cdots x_n$). It calls upon the functions SHIFT(v,s) and REDUCE(v,p). SHIFT(v,s) either adds a new leaf to U_i labeled with parse state s whose parent is vertex v or merges vertex v with the parents of an existing leaf. REDUCE(v,p) executes a reduce action from vertex v using production p. REDUCE calls upon the function ANCESTORS(v,$\bar{p}$), which returns the set of all ancestor vertices a distance of $\bar{p}$ from vertex v. These functions, which vary somewhat from the formal definition given in Tomita (1985),[2] are defined in Figure 3.1.

In REC, [1] adds the end-of-sentence symbol '$\dashv$' to the end of the input string; [2] initializes the root of the graph-structured stack; [3] iterates through the symbols of the input string. On each symbol x_i, [4] processes the vertices (denoting the active processes) of successive U_{i-1}'s, adding each vertex to P to signify that it has been processed. On each vertex v, [5] executes the shift, reduce, and accept actions from the action table according to v's state s. After processing the vertices in U_{i-1}, [6] checks whether a vertex was added to U_i, ensuring that at least one process is still active before scanning x_{i+1}.

In SHIFT, [7] shifts a process into state s by adding a vertex to U_i labeled s. If a vertex labeled s already exists, v is added to its parents, merging processes; otherwise, a new vertex is created with a single parent v.

[2] Tomita's functions REDUCE and REDUCE-E have been collapsed into a single REDUCE function; also added were the ANCESTORS function and the concept of a "clone" vertex. While these changes do not alter Tomita's algorithm significantly, they were helpful in developing ideas about its complexity.

```
        REC($x_1 \cdots x_n$)
[1]       let $x_{n+1}$ := $\dashv$
          let $U_i$ := [ ]   $(0 \leq i \leq n)$
[2]       let $U_0$ := $[\langle 0, S_0, \emptyset \rangle]$
[3]       for $i$ from 1 to $n+1$
            let $P$ := [ ]
[4]         for $\forall v = \langle i-1, s, l \rangle$ s.t. $v \in U_{i-1}$
              let $P$ := $P \circ [v]$
[5]           if $\exists$'sh $s'$' $\in$ ACTIONS($s$,$x_i$), SHIFT($v$,$s'$)
              for $\forall$'re $p$' $\in$ ACTIONS($s$,$x_i$), REDUCE($v$,$p$)
              if 'acc' $\in$ ACTIONS($s$,$x_i$), accept
[6]         if $U_i$ is empty, reject
        SHIFT($v$,$s$)
[7]       if $\exists v' = \langle i, s, l \rangle$ s.t. $v' \in U_i$
            let $l$ := $l \cup \{v\}$
          else
            let $U_i$ := $U_i \circ [\langle i, s, \{v\} \rangle]$
        REDUCE($v$,$p$)
[8]       for $\forall v_1' = \langle j', s', l_1' \rangle$ s.t. $v_1' \in$ ANCESTORS($v$,$\bar{p}$)
            let $s''$ := GOTO($s'$,$D_p$)
[9]         if $\exists v'' = \langle i-1, s'', l'' \rangle$ s.t. $v'' \in U_{i-1}$
[10]          if $v_1' \in l''$
                do nothing (ambiguous)
              else
[11]          if $\exists v_2' = \langle j', s', l_2' \rangle$ s.t. $v_2' \in l''$
                let $v_c''$ := $\langle i-1, s'', \{v_1'\} \rangle$
                for $\forall$'re $p$' $\in$ ACTIONS($s''$,$x_i$), REDUCE($v_c''$,$p$)
              else
[12]            let $l''$ := $l'' \cup \{v_1'\}$
[13]            if $v'' \in P$
                  let $v_c''$ := $\langle i-1, s'', \{v_1'\} \rangle$
                  for $\forall$'re $p$' $\in$ ACTIONS($s''$,$x_i$), REDUCE($v_c''$,$p$)
            else
[14]          let $U_{i-1}$ := $U_{i-1} \circ [\langle i-1, s'', \{v_1'\} \rangle]$
        ANCESTORS($v = \langle j, s, l \rangle$,$k$)
[15]      if $k = 0$
            return($\{v\}$)
          else
            return($\bigcup_{v' \in l}$ ANCESTORS($v'$,$k-1$))
```

Fig. 3.1—Tomita's Algorithm

In REDUCE, [8] iterates through the ancestor vertices a distance of $\bar{p}$ from v, setting s'' to the state indicated in the goto table under D_p given the ancestor's state s'. Each ancestor vertex v_1' is shifted into U_{i-1} on s''. [9] checks whether such a vertex v'' already exists. (If not, [14] adds a vertex labeled s'' to U_{i-1}.) If v'' does already exist, [10] checks that a shift from the current ancestor v_1' has not already been made. (If it has, then some segment of the input string has been recognized as an instance of the same nonterminal D_p in two different ways, and the current derivation can be discarded as ambiguous; otherwise, v_1' is merged with the parents of the existing vertex.) Before merging, [11] checks whether v_1' is a "clone" vertex, created by [13] in an earlier call to REDUCE (as described below). If v_1' is not a clone, [12] adds it to the parents of v'', merging processes. [13] checks if v'' has already been processed. If so, then it missed any reductions through v_1'. To correct this, v'' is "cloned" into v_c'' (i.e., a variant on v'' with a single parent v_1'), and all reduce actions executed on v'' are now executed on v_c''.

Returning to [11], when reducing on a null production, ANCESTORS will return a clone vertex as the ancestor of itself. If a variant v_2' of v_1' already exists in the parents of v'', then v_1' is a clone of v_2'. At this point v'' has already been processed, meaning that there could still be reductions that have not gone through the single parent of v_1'. To correct this, v'' is again cloned, and all reduce actions executed on v'' are executed on the new clone v_c''.

Finally, in ANCESTORS, [15] recursively descends the chain of parents of vertex v, returning the set of vertices a distance of k from v.

The General Case. Tomita's algorithm is an $O(n^{\bar{\rho}+1})$ recognizer in general, where $\bar{\rho}$ is the greatest $\bar{p}$ in G. The reasons for this are as follows:

(a) Since each vertex in U_i must be labeled with a distinct parse state, the number of vertices in any U_i is bounded by the number of parse states;

(b) The number of parents l of a vertex $v = \langle i, s, l\rangle$ in U_i is proportional to i. Because processes could have begun scanning for some production p in each U_j $(j \leq i)$, a process in U_i could reduce using p and split into $\sim i$ processes (one for each ancestor in a distinct U_j). Then each process could shift on D_p to the same state in U_i and, thus, that vertex could have $\sim i$ parents;

(c) For each x_{i+1}, SHIFT will be called a bounded number of times (at most once for each vertex in U_i). SHIFT executes in a bounded number of steps.

(d) For each x_{i+1} and production p, REDUCE(v,p) will be called a bounded number of times in REC, and REDUCE(v_c'',p) (the recursive call to REDUCE) will be called no more than $\sim i$ times. The reason for the former is the same as in (c). The latter is due to the conditions on the recursive call, which maintain that it can be called no more than once for each parent of a vertex in U_i, of which there are at most proportional to i;

(e) REDUCE(v,p), because at most $\sim i$ vertices can be returned by ANCESTORS, executes in $\sim i$ steps plus the steps needed to execute ANCESTORS.

(f) ANCESTORS(v,$\bar{p}$) executes in $\sim i^{\bar{p}}$ steps in the worst case. While at most $\sim i$ processes could have begun scanning for p, the number of paths by which any single process could reach v in U_i is bounded by the number of ways the intervening input symbols can be partitioned among the $\bar{p}$ vocabulary symbols in the right-hand side of production p. For a process that started from U_j $(j \leq i)$, the number of paths to v in U_i in the recognition of p can be proportional to

$$\sum_{m_1=j}^{0}\sum_{m_2=m_1}^{0}\cdots\sum_{m_{\bar{p}-1}=m_{\bar{p}-2}}^{0} 1.$$

Summing from $j = 0, \cdots, i$ gives a closed form proportional to $i^{\bar{p}}$. ANCESTORS(v_c'',$\bar{p}$), where $v_c'' = \langle i, s\{v'\}\rangle$, executes in $\sim i^{\bar{p}-1}$ steps because there is that many ways $\sim i$ ancestor vertices could reach v' and only one way v' could reach v_c'';

(g) The worst case time bound is dominated by the time spent in ANCESTORS, which can be added to the time spent in REDUCE. Since REDUCE(v,p), with a bound $\sim i^{\bar{p}}$, is called only a bounded number of times, and REDUCE(v_c'',p), with a time bound of $\sim i^{\bar{p}-1}$, is called at most $\sim i$ times, the worst case time to process any x_i is $\sim i^{\bar{p}}$, for each $i = 0, \cdots, n+1$ and longest production ρ;

(h) Summing from $i = 0, \cdots, n+1$ gives REC a general time bound proportional to $n^{\bar{p}+1}$.

As a result, this bound indicates that Tomita's algorithm only belongs to complexity class $O(n^3)$ when applied to grammars in Chomsky normal form (CNF)[3] or some other equally truncated notation.

[3] In CNF, productions can have one of two forms, A $\rightarrow$ BC or A $\rightarrow$ a; thus, the length of the longest production is at most 2.

Although any CF grammar can be automatically converted to CNF (Hopcraft and Ullman, 1979), extracting useful information from derivation trees produced by such grammars would be time consuming at best (if possible at all).

4. MODIFYING TOMITA'S ALGORITHM FOR N^3 TIME

In this section, Tomita's algorithm is made an $O(n^3)$ recognizer for CF grammars with productions of arbitrary length. Essentially, the modifications are to the ANCESTORS function. ANCESTORS is the only function that forces us to use $i^{\bar{p}}$ steps. It is interesting to note that ANCESTORS can take this many steps even though it returns at most $\sim i$ ancestor vertices and even though there are at most $\sim i$ intervening vertices and edges between a vertex in U_i and its ancestors. This indicates that ANCESTORS can recurse down the same subpaths more than once. The efficiency of ANCESTORS and Tomita's algorithm can be improved by eliminating this redundancy.

The modification described here turns ANCESTORS into a table look-up function. Assume there is a two-dimensional "ancestors" table. One dimension is indexed on the vertices in the graph-structured stack, and the other is indexed on integers $k = 1, \cdots, \bar{\rho}$, where $\bar{\rho}$ equals the greatest $\bar{p}$. Each entry (v, k) is the set of ancestor vertices a distance of k from vertex v. Then, ANCESTORS(v,k) returns the (at most) $\sim i$ ancestor at (v, k) in ~ 1 steps. Of course, the table must be filled dynamically during the recognition process, so the time expended in this task must also be determined.

In Figure 4.1, ANCESTORS is defined as a table look-up function that dynamically generates table entries the first time they are requested. In this definition, the ancestor table is represented by changing the parent field l of a vertex $v = \langle i, s, l\rangle$ from a set of parent vertices to an ancestor field a. For a vertex $v = \langle i, s, a\rangle$, a consists of a set of tuples $\langle k, l_k\rangle$, such that l_k is the set of ancestor vertices a distance of k from v.

Figure 4.1 illustrates the necessary modifications made to the definitions of Figure 3.1; the function REC is unchanged. In SHIFT, [1] adds a vertex to U_i labeled s. If such a vertex does not already exist, one is created whose ancestor field records that v is the ancestor vertex at a distance of 1; otherwise, v is added to the other distance-1 ancestors.

In REDUCE, [2] iterates through the ancestor vertices a distance of $\bar{p}$ from v, setting s'' to the state indicated in the goto table under D_p given the ancestor's state s'. Each ancestor vertex v_1' is shifted into U_{i-1} on s''. [3] checks whether such a vertex v'' already exists. (If not, [10] will add a vertex labeled s'' to U_{i-1}.) If v'' does already exist, [4] checks that a shift from the current ancestor v_1' has not already been made. If it has, then v_1' can be discarded as ambiguous; if not, then v_1' can be merged with the other ancestors a distance of 1 from v''. Before merging, [5] checks whether v_1' is a clone vertex as described in Section 3. If v_1' is a clone (the result of being reduced on a null production), v'' is again cloned, and all reduce actions executed on v'' are executed on the new clone v_c''. After the application of REDUCE, [6] updates the ancestor table stored in v'' to record entries made in the ancestor field a_c'' of the clone when $k \geq 2$. Otherwise, if v_1' is not a clone, [7] adds it to the distance-1 ancestors of v'', merging processes. [8] checks if v'' has already been processed. If so, then it missed any reductions through v_1', so v'' is cloned into v_c'' and all reduce actions executed on v'' are now executed on v_c''. After reducing v_c'', [9] updates the ancestor table stored in v'' to record entries made in the ancestor field a_c'' of the clone when $k \geq 2$.

In ANCESTORS, [11] searches a (the portion of the ancestor table stored with v) for ancestor vertices at a distance of k from v. If an entry exists, those vertices are returned; if not, [12] calls ANCESTORS recursively to generated those vertices and, before returning the generated vertices, records them in the ancestor field of v.

The question now becomes how much time is spent filling the ancestor table. For ANCESTORS(v,$\bar{p}$), time is bounded in the worst case by $\sim i^2$ steps, while for ANCESTORS(v_c'',$\bar{p}$), it is bounded by $\sim i$ steps. In general, ANCESTORS(v,k), where $v = \langle i, s, a\rangle$, will take $\sim i$ steps to execute the first time it is called (one for each recursive call to ANCESTORS(v',$k-1$), where

```
      SHIFT($v$,$s$)
[1]     if $\exists v' = \langle i, s, a\rangle$ s.t. $v' \in U_i \wedge \langle 1, l\rangle \in a$,
          let $l := l \cup \{v\}$
        else
          let $U_i := U_i \circ [\langle i, s, [\langle 1, \{v\}\rangle]\rangle]$

      REDUCE($v$,$p$)
[2]     for $\forall v_1' = \langle j', s', a_1'\rangle$ s.t. $v_1' \in$ ANCESTORS($v$,$\bar{p}$)
          let $s'' :=$ GOTO($s'$,$D_p$)
[3]       if $\exists v'' = \langle i-1, s'', a''\rangle$ s.t. $v'' \in U_{i-1} \wedge \langle 1, l''\rangle \in a''$
[4]         if $v_1' \in l''$
              do nothing (ambiguous)
            else
[5]         if $\exists v_2' = \langle j', s', a_2'\rangle$ s.t. $v_2' \in l''$
              let $v_c'' := \langle i-1, s'', a_c''\rangle$ s.t. $a_c'' = [\langle 1, \{v_1'\}\rangle]$
              for $\forall$'re $p' \in$ ACTIONS($s''$,$x_i$), REDUCE($v_c''$,$p$)
[6]           let $l_{k_1} := l_{k_1} \cup l_{k_2}$ s.t. $\langle k, l_{k_1}\rangle \in a'' \wedge \langle k, l_{k_2}\rangle \in a_c''$ $(k \geq 2)$
            else
[7]           let $l'' := l'' \cup \{v_1'\}$
[8]           if $v'' \in P$
                let $v_c'' := \langle i-1, s'', a_c''\rangle$ s.t. $a_c'' = [\langle 1, \{v_1'\}\rangle]$
                for $\forall$'re $p' \in$ ACTIONS($s''$,$x_i$), REDUCE($v_c''$,$p$)
[9]             let $l_{k_1} := l_{k_1} \cup l_{k_2}$ s.t. $\langle k, l_{k_1}\rangle \in a'' \wedge \langle k, l_{k_2}\rangle \in a_c''$ $(k \geq 2)$
          else
[10]        let $U_{i-1} := U_{i-1} \circ [\langle i-1, s'', \{v_1'\}\rangle]$

      ANCESTORS($v = \langle j, s, a\rangle$,$k$)
[11]    if $k = 0$,
          return($\{v\}$)
        else
        if $\exists \langle k, l_k\rangle \in a$,
          return($l_k$)
        else
[12]      let $l_k := \bigcup_{v' \in l_1 | \langle 1, l_1\rangle \in a}$ ANCESTORS($v'$,$k-1$)
          let $a := a \cup \{\langle k, l_k\rangle\}$
          return($l_k$)
```

Fig. 4.1—Modified Algorithm

$v' \in l_1$ and $\langle 1, l_1\rangle \in a$) and ~ 1 steps thereafter. When ANCESTORS(v,$\bar{p}$) is executed, there are $\sim i$ such "virgin" vertices between v and its ancestors, and so this call can execute $\sim i^2$ steps in the worst case. ANCESTORS(v_c'',$\bar{p}$) is called only after the call to ANCESTORS(v,$\bar{p}$) has been made, where v_c'' is a clone of v. This means that $\sim i$ of the vertices between v' and the ancestor vertices have been processed, so the call to ANCESTORS(v',$\bar{p} - 1$) could take at most proportional to i steps for each of a bounded number of intervening vertices.

Given this, the upper bound on the number of steps that can be executed by the total calls on REDUCE for a given x_i is proportional to i^2. Summing from $i = 0, \cdots, n+1$ gives $\sim n^3$ steps as the worst case upper bound on the execution time of the modified algorithm.

5. SPACE BOUNDS

The space complexity of Tomita's algorithm as it appears in Section 3 is proportional to n^2 in the worst case. This is because the space requirements of the algorithm are bounded by the requirements of the graph-structured stack. There are a bounded number of vertices in each U_i of the graph-structured stack, and each vertex can have at most $\sim i$ parents. Summing again from $i = 0, \cdots, n+1$ gives $\sim n^2$ as the worst case space requirement for the graph-structured stack.

With the modification of Section 4, the space requirements of the graph-structured stack are increased by at most a constant factor of n^2. This is because the modification replaces the $\sim i$ parents of a vertex in U_i with at most $\sim \bar{\rho} i$ entries in the ancestors field. So, for a vertex $v = \langle i, s, a \rangle$ s.t. $v \in U_i$, the ancestors field a will be a subset of $\{(c, l_c) | 1 \leq c \leq \bar{\rho}\}$ where $|l_c| \simeq i$. Summing from $i = 0, \cdots, n+1$ gives $\sim \bar{\rho} n^2$ or $\sim n^2$ still as a worst case upper bound on space.

6. EMPIRICAL RESULTS

The variation on Tomita's algorithm presented in Section 3 and the modified algorithm presented in Section 4 have both been implemented in C. The graphs in figures 6.1 and 6.2 show empirical results comparing the time and space requirements of both implementations. Each time/space graph set corresponds to the grammars, $G1$, $G2$, and $G3$, which are dominated by productions of length 2, 3 and 4.

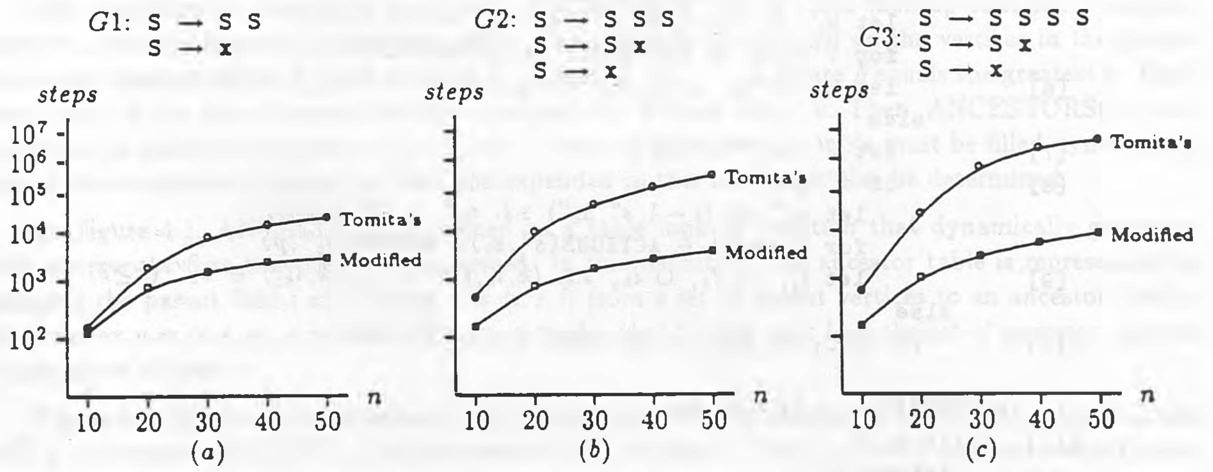


Fig. 6.1—Comparison of Time Complexity

The time graphs in Figure 6.1 measure the number of calls to SHIFT, REDUCE, and ANCESTORS. The input sentences are strings of x's of length 10 to 50. Our analysis of time complexity predicts that the modified algorithm will take roughly the same number of steps for each grammar, while the steps taken by Tomita's algorithm will increase as a function of the length of the dominant production. The empirical data gathered from our two implementations agrees with this prediction. When $n = 50$, the modified algorithm took ~ 7000 steps for grammar $G1$ in Figure 6.1 (a), ~ 6000 for $G2$ in Figure 6.1 (b), and ~ 10000 for $G3$ in Figure 6.1 (c); Tomita's algorithm took $\sim 44{,}000$ steps for grammar $G1$, $\sim 660{,}000$ for $G2$, and $\sim 7{,}300{,}000$ for $G3$.

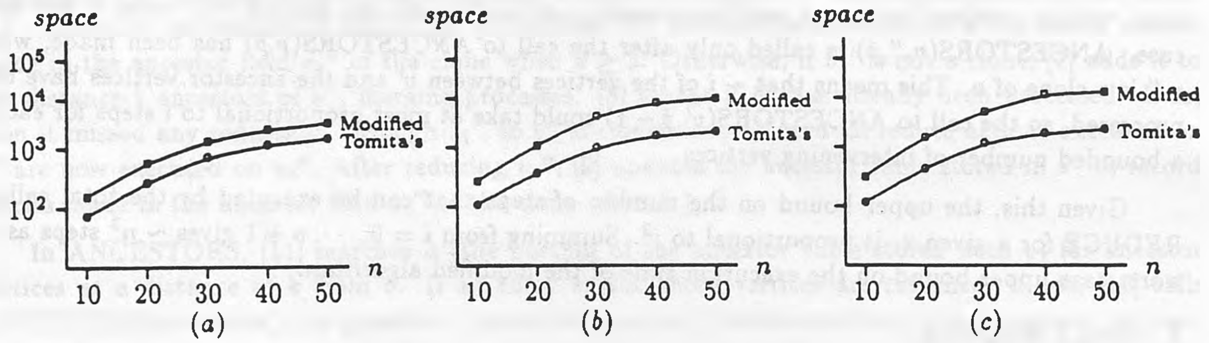


Fig. 6.2—Comparison of Space Complexity

The space graphs in Figure 6.2 measure the number of edges required by the graph-structured stack (in Tomita's algorithm) and the length of entries in the ancestors table (in the modified algorithm). The number of vertices required is the same for both algorithms and is not counted; space that can be reclaimed before scanning successive x_i's is also not counted. Our analysis of space complexity predicts that Tomita's algorithm will require $\sim n^2$ space and that the modified algorithm will require at most a factor of n^2 additional space. The empirical evidence also agrees with this prediction. The space requirements of the modified algorithm differs from Tomita's algorithm by a factor of ~ 2.1 for grammar $G1$ in Figure 6.2 (a), ~ 3.9 for $G2$ in Figure 6.2 (b), and ~ 4.7 for $G3$ in Figure 6.2 (c).

7. LESS THAN N^3 TIME

Several of the better known general CF algorithms have been shown to run in less than $O(n^3)$ time for certain subclasses of grammars. Therefore, it is of interest to ask if Tomita's algorithm, as well as the modified version presented here, can also recognize some subclasses of CF grammars in less than $O(n^3)$ time. In this section, I informally describe two such subclasses that can be recognized in $O(n^2)$ and $O(n)$ time, respectively. The arguments for their existence parallel those given by Earley (1968), where they are formally specified.

Time $O(n^2)$ Grammars. ANCESTORS is the only function that forces us to use $\sim i^{\tilde{\rho}}$ steps in Tomita's algorithm and $\sim i^2$ steps in the modified algorithm. We determined that this could happen when a ancestor vertex v' from U_j ($j \leq i$) reached the reducing vertex v in U_i by more than a single path, i.e., the symbols $x_j \cdots x_i$ were derived from a nonterminal D_p in more than one way, indicating that grammar G is ambiguous. If G were unambiguous, then there would be at most one path from a given v' to v. This means that the bounded calls to ANCESTORS(v,$\bar{p}$) can take at most $\sim i$ steps and that ANCESTORS(v_c'',$\bar{p}$) can take at most a bounded number of steps. The first observation is due to the fact that there are $\sim i$ ancestor vertices that can be reached in only one way. Similarly, the second observation is due to the fact that if ANCESTORS(v_c'',$\bar{p}$) took $\sim i$ steps, returning $\sim i$ ancestors, and was called $\sim i$ times, then some ancestor vertices must have shifted into U_i in more than one way, which would be a contradiction, meaning grammar G must be ambiguous. So, if the grammar is unambiguous, then the total time spent in REDUCE for any x_i is $\sim i$ and the worst case time bound for the Tomita's algorithm is $O(n^2)$. A similar result is true for the modified algorithm.

Time $O(n)$ Grammars. In his thesis, Earley (1968) points out that "*... for some grammars the number of states in a state set can grow indefinitely with the length of the string being recognized. For some others there is a fixed bound on the size of any state set. We call the latter grammars bounded state grammars.*" While Earley's "states" have a different meaning than states in Tomita's algorithm, a similar phenomena occurs, i.e., for the bounded state grammars there is a fixed bound on the number of parents any vertex can have. In Tomita's algorithm, bounded state grammars can be recognized in time $O(n)$ for the following reason. No vertex can have more than a bounded number of ancestors (if otherwise, then $\sim i$ vertices could be added to the parents of some vertex in U_i, proving by contradiction that the grammar is not bounded state). This means that the ANCESTORS function can execute in a bounded number of steps. Likewise, REDUCE can only be called a bounded number of times. Summing over the x_i gives us an upper bound $\sim n$. Again, a similar result is true for the modified algorithm. Interestingly enough, Earley states that almost all LR(k) grammars are bounded state, as well, which suggests that Tomita's algorithm, given k-symbol look ahead, should perform with little loss of efficiency as compared to a standard LR(k) algorithm when the grammar is "close" to LR(k). Earley also points out that not all bounded state grammars are unambiguous; thus, there are non-LR(k) grammars for which Tomita's algorithm can perform with LR(k) efficiency.

8. CONCLUSION

The results in this paper support in part Tomita's claim (1985) of efficiency for his algorithm. With the modification introduced here, Tomita's algorithm is shown to be in the same complexity class as existing general CF algorithms. These results also give support to his claim that his algorithm should run with near LR(k) efficiency for near LR(k) grammars.

It should be noted that while the modification to Tomita's algorithm has theoretic interest it would detract from a practical parser. Realistic grammars are constrained by the fact that they must be human-readable. Since human-readable grammars should never realize the worst-case $O(n^{\tilde{\rho}+1})$ time bound of Tomita's algorithm, the benefits of the ancestors table in the modified algorithm would not balance out its overhead cost. In this regard, the modified algorithm should not be viewed as an "improvement" over Tomita's algorithm but as a means of illustrating its place among other general CF algorithms.

The variation on Tomita's algorithm described in this paper, as well as the modified algorithm, have been implemented in both LISP and C at The RAND Corporation. The LISP implementation (Kipps, 1988) is distributed with ROSIE (Kipps et al., 1987), a language for applications in artificial intelligence with a highly ambiguous English-like syntax. The C implementation is part of the RAND Translator-Generator project, which is developing a "next generation" YACC[4] for non-LR(k) languages.

REFERENCES

Aho, A.V., J.D. Ullman, *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, Englewood Cliffs, NJ, 1972.

Chomsky, N., "On Certain Formal Properties of Grammars," in *Information and Control*, vol. 2, no. 2, pp. 137-167, 1959.

Earley, J., *An Efficient Context-Free Parsing Algorithm*, Ph.D. Thesis, Computer Science Dept., Carnegie-Mellon University, Pittsburg, PA, 1968.

Hopcraft, J.E., J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

Kipps, J.R., B. Florman, H.A. Sowizral, *The New ROSIE Reference Manual and User's Guide*, R-3448-DARPA, The RAND Corporation, 1987.

Kipps, J.R., "A Table-Driven Approach to Fast Context-Free Parsing," N-2841-DARPA, The RAND Corporation, 1988.

Knuth, D.E., "On the Translation of Languages from Left to Right," *Information and Control*, vol. 8, pp. 607-639, 1965.

Johnson, S.C., "YACC—Yet Another Compiler Compiler," CSTR 32, Bell Laboratories, Murray Hill, NJ, 1975.

Tomita, M., *An Efficient Context-Free Parsing Algorithm for Natural Languages and Its Applications*, Ph.D. Thesis, Computer Science Dept., Carnegie-Mellon University, Pittsburg, PA, 1985.

Younger, D.H., "Recognition and Parsing of Context-Free Languages in Time n^3," in *Information and Control*, vol. 10, no. 2, pp. 189-208, 1967.

[4] YACC (Johnson, 1975) is a parser-generator for LALR(1) languages.

The Computational Complexity of Tomita's Algorithm

Mark Johnson

April 26, 1989

1 Introduction

The Tomita parsing algorithm adapts Knuth's (1967) well-known parsing algorithm for LR(k) grammars to non-LR grammars, including ambiguous grammars. Knuth's algorithm is provably efficient: it requires at most $O(n|G|)$ units of time, where $|G|$ is the size of (i.e. the number of symbols in) G and n is the length of the string to be parsed. This is often significantly better than the $O(n^3|G|^2)$ worst case time required by standard parsing algorithms such as the Earley algorithm. Since the Tomita algorithm is closely related to Knuth's algorithm, one might expect that it too is provably more efficient than the Earley algorithm, especially as actual computational implementations of Tomita's algorithm outperform implementations of the Earley algorithm (Tomita 1986, 1987).

This paper shows that this is not the case. Two main results are presented in this paper. First, for any m there is a grammar L_m such that Tomita's algorithm performs $\Omega(n^m)$ operations to parse a string of length n. Second, there is a sequence of grammars G_m such that Tomita's algorithm performs $\Omega(nc^{|G_m|})$ operations to parse a string of length n. Thus it is not the case that the Tomita algorithm is always more efficient than Earley's algorithm; rather there are grammars for which it is exponentially slower. This result is forshadowed in Tomita (1986, p. 72), where the author remarks that Tomita's algorithm can require time proportional to more than the cube of the input length. The result showing that the Tomita parser can require time proportional to an exponential function of the grammar size is new, as far as I can tell.

2 The Tomita Parsing Algorithm

This section briefly describes the relevant aspects of the Tomita parsing algorithm: for further details see Tomita (1986). Familiarity with Knuth's LR parsing algorithm is presumed: see the original article by Knuth (1967), Aho and Ullman (1972), or Aho, Sethi and Ullman (1986) for details.

The Tomita algorithm and Knuth's LR parsing algorithm on which it is based are both shift-reduce parsing algorithms, and both use the same LR automaton to determine the parsing actions to be performed. The LR automaton is not always deterministic: for example, if the grammar is ambiguous then at some point in the analysis of an ambiguous string two different parsing actions must be possible that lead to the two distinct analyses of that string. Knuth's algorithm is only defined for grammars for which the parsing automaton is deterministic: these are called the LR(k) grammars, where k is the length of the lookahead strings. Tomita's algorithm extends Knuth's to deal with non-deterministic LR automata.

Tomita's algorithm in effect simulates non-determinism by computing *all* of the LR stacks that result from each of the actions of a non-deterministic LR automaton state. Tomita's algorithm mitigates the cost of this non-determinism by representing the set of all the LR stacks possible at a given point of the parse as a multiply-rooted directed acyclic graph called a *graph-structured stack*, which is very similiar to a parsing chart (Tomita 1988). Each node of this graph represents an LR state of one or more of the LR stacks, with the root nodes representing the top states of LR parse stacks. The graph contains exactly one leaf node (i.e. a node with no successors). This leaf node represents the start state of the LR automata (since this is the bottom element of all LR parse stacks), and each maximal path through the graph (i.e. from a root to the leaf) represents an LR parse stack.

As each item in the input string is read all of the parsing actions called for by the top state of each LR stack are performed, resulting in a new set of LR stacks. Because of the way in which the set of LR stacks are represented, Tomita's algorithm avoids the need to copy the each LR stack in its entirity at non-deterministic LR automaton states; rather the top elements of the two (or more) new stacks are represented nodes whose successors are the nodes that represent the LR stack elements they have in common. Similiarly, if the same LR state appears as the top element of two or more new stacks then these elements are represented by a single node whose immediate successors are the set of nodes that represent the other elements of these LR stacks. This "merging" of identical top elements of distinct LR stacks allows Tomita's algorithm to avoid duplicating the same computation in different contexts.

Finally, Tomita employs a *packed forest* representation of the parse trees in order to avoid enumerating these trees, the number of which can grow exponentially as a function of input length. In this representation there is at most one node of a given category at any string location (i.e. a pair of beginning and ending string positions), so the number of nodes in such a packed forest is at most proportional to the square of the input length. Each node is associated with a set of sequences of daughter nodes where each sequence represents one possible expansion of the node; thus the trees represented can easily be "read off" the packed forest representation.

3 Complexity as a Function of Input Length

The rest of this paper shows the complexity results claimed above. This section describes a sequence of grammars L_m such that on sufficiently long inputs the Tomita algorithm performs more than $\Omega(n^m)$ operations to parse an input of length n. This result follows from properties of the packed forest representation alone, so it applies to *any* algorithm that constructs packed forest representations of parse trees.

Consider the sequence of grammars L_m for $m > 0$ defined in (1), where S^{m+2} abbreviates a sequence of S's of length $m + 2$.

$$\begin{array}{l} S \rightarrow a \\ S \rightarrow SS \\ S \rightarrow S^{m+2} \end{array} \qquad (1)$$

All of these grammars generate the same language, namely the set of strings a^+. Consider the input string a^{n+2} for $n > m$. By virtue of the first two rules in (1) any non-empty string location can be analyzed as an S. Thus the number of different sequences of daughter nodes of the matrix or top-most S node licensed by the third rule in (1) is $\binom{n}{m+1}$ the number of ways of choosing different right string positions of the top-most S node's first $m + 1$ daughters. Since $\binom{n}{m+1}$ is a polynomial in n of order $m + 1$, it is bounded below by cn^m for some $c > 0$ and sufficiently large n, i.e. $\binom{n}{m+1} = \Omega(n^m)$. Since any algorithm which uses the packed forest representation, such as Tomita's algorithm, requires the construction of these sequences of daughter nodes, any such algorithm must perform $\Omega(n^m)$ operations.

Finally, it should be noted that this result assumes that the sequences of daughter nodes are completely enumerated. It might be possible these sequences could themselves be "packed" in such a fashion that avoids their enumeration, possibly allowing the packed forest representations to be constructed in polynomial time.

4 Complexity as a Function of Grammar Size

This section shows that there are some grammars such that the total number of operations performed by the Tomita algorithm is an exponential function of the size of the grammar.

The amount of work involved in processing a single input item is proportional to the number of distinct top states of the set of LR stacks corresponding to the different non-deterministic analyses of the portion of the input string shown so far. By exhibiting a sequence of grammars in which the number of such states is an exponential function of the size of the grammar we show that the total number of operations performed by the Tomita algorithm can be at least exponentially related to the size of the grammar.

Consider the sequence of grammars G_m for $m > 0$ defined in (2).

$$\begin{array}{ll} S \rightarrow A_i & 0 \leq i \leq m \\ A_i \rightarrow B_j A_i & 0 \leq i,j \leq m, i \neq j \\ A_i \rightarrow B_j & 0 \leq i,j \leq m, i \neq j \\ B_j \rightarrow a & 0 \leq j \leq m \end{array} \tag{2}$$

All of the grammars G_m generate the same language, namely the set of strings a^+. Since these grammars are ambiguous they are not LR(k) for any k.

Consider the behaviour of a non-deterministic LR parser for the grammar G_m on an input string a^n where $n > m$. The items of the start state are shown in (3).

$$\left[\begin{array}{l} S \rightarrow \cdot A_i \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{array}\right] \quad 0 \leq i,j \leq m, i \neq j \tag{3}$$

The parser shifts over the first input symbol a to the state shown in (4).

$$[B_j \rightarrow a \cdot] \quad 0 \leq j \leq m \tag{4}$$

This is a non-deterministic state, since all of the m reductions $B_j \rightarrow a$ are possible parsing actions from this state. Suppose that the reduction to B_{k_1} is chosen. The state that results from the reduction to B_{k_1} is shown in (5). There are m such states.

$$\left[\begin{array}{l} A_i \rightarrow B_{k_1} \cdot A_i \\ A_i \rightarrow B_{k_1} \cdot \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{array}\right] \quad 0 \leq i,j \leq m, i \neq j, k_1 \tag{5}$$

After shifting over the next input symbol the parser again reaches the same ambiguous state as before, namely the state shown in (4). Suppose the reduction to B_{k_2} is chosen. If $B_{k_1} = B_{k_2}$ then the resulting state is the one shown in (5). On the other hand, if $B_{k_1} \neq B_{k_2}$ then the resulting state is as shown in (6). There are $m(m-1)/2$ distinct states of the form shown in (6), so after reducing B_{k_2} there will be $m(m+1)/2$ distinct LR states in all.

$$\left[\begin{array}{l} A_i \rightarrow B_{k_1} \cdot A_i \\ A_i \rightarrow B_{k_1} \cdot \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{array}\right] \quad 0 \leq i,j \leq m, i \neq j, k_1, k_2 \tag{6}$$

It is not hard to see that after $n \geq m$ input symbols have been read and reduced to $B_{k_1} \ldots B_{k_n}$ respectively the resulting state will be as shown in (7).

$$\begin{bmatrix} A_i \rightarrow B_{k_n} \cdot A_i \\ A_i \rightarrow B_{k_n} \cdot \\ A_i \rightarrow \cdot B_j A_i \\ A_i \rightarrow \cdot B_j \\ B_j \rightarrow \cdot a \end{bmatrix} \qquad 0 \leq i, j \leq m, i \neq j, k_1 \ldots k_n \tag{7}$$

Since there are $2^m - 1$ distinct such states, the Tomita parser must perform at least $2^m - 1$ computations per input item after the first m items have been read. Since $|G_m| = 5m^2 - m = O(m^2)$, the ratio of the average number of computations per input item for a sufficiently long string to grammar size is $\Omega(2^m/m^2) = \Omega(c^m)$ for some $c > 1$. Thus the total number of operations performed by the parser is $\Omega(c^{|G_m|})$, exponential function of grammar size.

5 Conclusion

The results just demonstrated do not show that Tomita's algorithm is always slower than polynomially bounded algorithms such as Earley's, in fact in practice it is significantly faster than Earley's algorithm (Tomita 1986). On the other hand, the results presented here show that this superior performance is not just a property of the algorithm alone, but also depend on properties of the grammars (and possibly the inputs) used. It would be interesting to identify the properties that are required for efficient functioning of Tomita's algorithm.

Second, it might possible to modify Tomita's algorithm so that it provably requires at most polynomial time. For example, requiring all grammars used by the algorithm to be in Chomsky Normal Form would prohibit the grammars used to show that Tomita's algorithm does not always run in polynomial time. Whether this restriction would ensure polynomial time behaviour with respect to input length is an open question (note that the grammars used to show the exponential complexity with respect to grammar size are already in Chomsky Normal Form).

Finally, the non-polynomial behaviour of Tomita's algorithm with respect to input length followed from the properties of the packed forest representation of parse trees, so it follows that any algorithm which uses packed forest representations will also exhibit non-polynomial behaviour.

6 Bibliography

Aho and Ullman (1972) *The Theory of Parsing, Translation and Compiling*, vol. 1, Prentice Hall, New Jersey.

Aho, Sethi and Ullman (1986) *Compilers: Principles, Techniques and Tools*, Addison-Wesley, Reading, Mass.

Tomita (1986) *Efficient Parsing for Natural Language*, Kluwer, Boston, Mass.

Tomita (1987) "An Efficient Augmented-Context-Free Parsing Algorithm", *Computational Linguistics*, vol. 13, 31-46.

Tomita (1988) "Graph-Structured Stack and Natural Language Parsing", in *The Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, SUNY Buffalo, New York.

Probabilistic Parsing for Spoken Language Applications

Stephanie Seneff
Spoken Language Systems Group
Laboratory for Computer Science
MIT Cambridge, MA 02139

Abstract

A new natural language system, TINA, has been developed for applications involving spoken language tasks, which integrates key ideas from context free grammars, Augmented Transition Networks (ATN's) [6], and Lexical Functional Grammars (LFG's) [1]. The parser uses a best-first search strategy, with probability assignments on all arcs obtained automatically from a set of example sentences. An initial context-free grammar, derived from the example sentences, is first converted to a probabilistic network structure. Control includes both top-down and bottom-up cycles, and key parameters are passed among nodes to deal with long-distance movement, agreement, and semantic constraints. The probabilities provide a natural mechanism for exploring more common grammatical constructions first. One novel feature of TINA is that it provides an automatic sentence generation capability, which has been very effective for identifying overgeneration problems. A fully integrated spoken language system using this parser is under development.

1 Introduction

Most parsers have been designed with the assumption that the input word stream is deterministic: i.e., at any given point in the parse tree it is known with certainty what the next word is. As a consequence, these parsers generally cannot be used effectively, if at all, to provide linguistically directed constraint in the speech recognition component of a speech understanding system. In a fully integrated speech understanding system, the recognition component should only be allowed to propose partial word sequences that the natural language component can interpret; any word sequences that are syntactically or semantically anomalous should probably be pruned prior to the acoustic match, rather than examined for approval in a verification mode. To operate in such a fully integrated mode, a parser has to have the capability of considering a multitude of hypotheses simultaneously. The control strategy should have a sense of which of these hypotheses, considering both linguistic and acoustic evidence, is most likely to be correct at any given instant in time, and to pursue that hypothesis only incrementally before reexamining the evidence. The linguistic evidence should include probability assignments on proposed hypotheses; otherwise the perplexity of the task becomes too high for practical recognition applications.

This paper describes a natural language system, TINA, which addresses many of these issues. The grammar is constructed by converting a set of context-free rewrite rules to a form that merges common elements on the right-hand side (RHS) of all rules sharing the same left-hand side (LHS). Elements on the LHS become parent nodes in a family tree. Through example sentences, they acquire knowledge of who their children are and how they can interconnect. Such a transformation permits considerable structure sharing among the rules, as is done in typical shift-reduce parsers [5]. Probabilities are established on arcs connecting pairs of right siblings rather than on rule productions. This has several advantages, which will be discussed later. Context-dependent constraints to deal with agreement and gaps are realized through simple logical functions applied to flags or features passed among immediate relatives.

2 General Description

TINA is basically a context-free grammar, implemented by expansion at run-time into a network structure, and augmented with flags/parameters that activate filtering operations. The grammar is built from a set of training sentences, using a bootstrapping procedure. Initially, each sentence is translated by hand into a list of the rules invoked to parse it. After the grammar has built up a substantial knowledge of the language, many new sentences can be parsed automatically, or with minimal intervention to add a few new rules incrementally. The arc probabilities can be incrementally updated after the successful parse of each new sentence.

The process of converting the rules to a network form is straightforward. All rules with the same LHS are combined to form a structure describing possible interconnections among children of a parent node associated with the left-hand category. A probability matrix connecting each possible child with each other child is constructed by counting the number of times a particular sequence of two siblings occurred in the RHS's of the common rule set, and normalizing by counting all pairs from the particular left-sibling to *any* right sibling. Two distinguished nodes, a START node and an END node, are included among the children of every grammar node. A subset of the grammar nodes are terminal nodes whose children are a list of vocabulary words.

This process can be illustrated with the use of a simple example. Consider the following three rules:

NP ⇒ ARTICLE NOUN
NP ⇒ ARTICLE ADJECTIVE NOUN
NP ⇒ ARTICLE ADJECTIVE ADJECTIVE NOUN

These would be converted to a network as shown in Figure 1, which would be associated with a grammar node named NP. Since ADJECTIVE is followed twice by NOUN and once by ADJECTIVE, the network shows a probability of 1/3 for the self loop and 2/3 for the advance to NOUN. Notice that the system has now generalized to include any number of adjectives in a row.

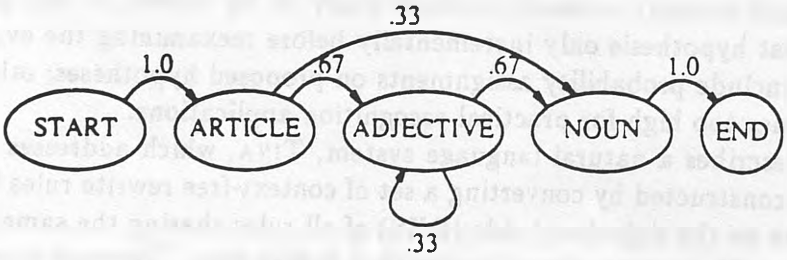


Figure 1: Probablistic Network Resulting from three Context-Free Rules given in Text.

A functional block diagram of the control strategy is given in Figure 2. At any given time, a set of active *parse nodes* are arranged on a priority queue. Each parse node contains a pointer to a corresponding grammar node, and has access to all the information needed to pursue its partial theory. The top node is popped from the queue, and it then creates a number of new nodes (either children or right siblings depending on its state), and inserts them into the queue according to their probabilities. If the node is an END node, it collects up all subparses from its sequence of left siblings, back to the START node, and passes the information up to the parent node, giving that node a completed subparse. The process can terminate on the first successful completion of a sentence, or the Nth successful completion if more than one hypothesis is desired.

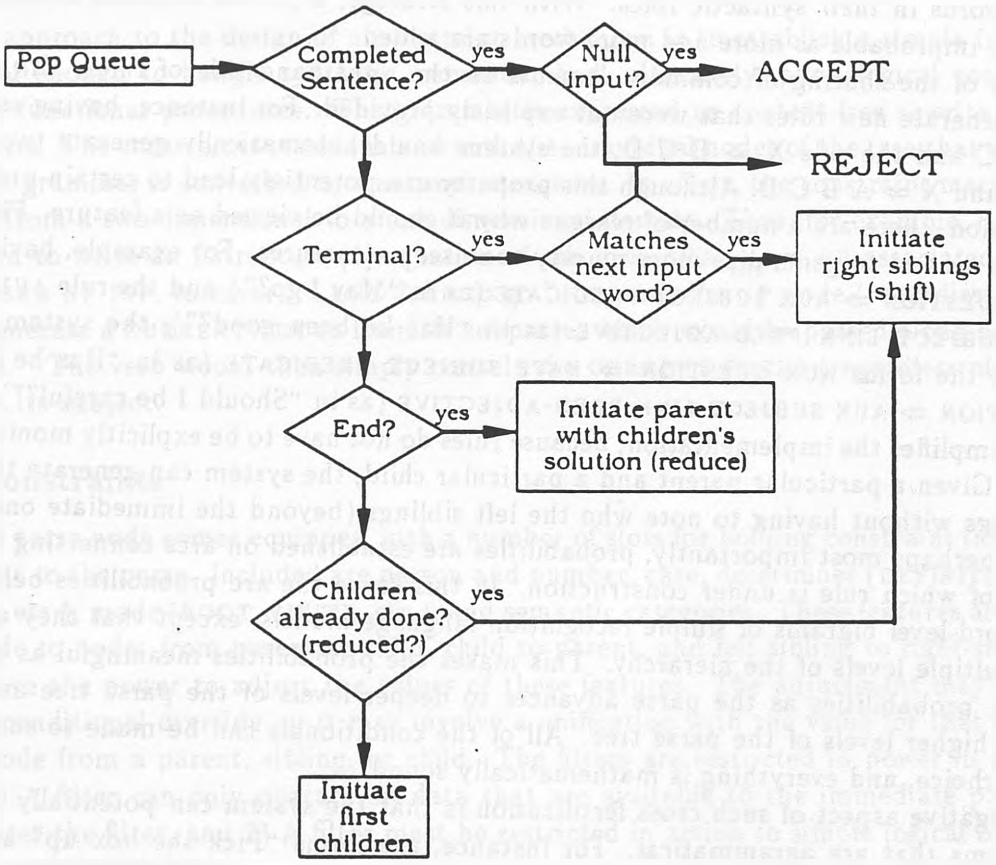


Figure 2: Functional Block Diagram of Control Strategy.

A parse in TINA begins with a single parse node linked to the grammar node SENTENCE, which is entered on the queue with probability 1.0. This node creates new parse nodes with categories like STATEMENT, QUESTION, and REQUEST, and places them on the queue, prioritized. If STATEMENT is the most likely child, it gets popped from the queue, and returns nodes indicating SUBJECT, IT, etc., to the queue. When SUBJECT reaches the top of the queue, it activates units such as NOUN-GROUP (for noun phrases and associated post-modifiers), GERUND, and NOUN-CLAUSE. Each node, after instantiating first-children, becomes inactive, pending the return of a successful subparse from a sequence of children. Eventually, the cascade of first-children reaches the terminal-node ARTICLE, which proposes the words "the," "a," and "an," testing these hypotheses against the input stream. If a match with "the" is found, then the ARTICLE node fills its subparse slot with the entry (ARTICLE "the"), and activates all of its possible right-siblings.

Whenever a terminal node has successfully matched an input word, the path probability is reset to 1.0. Thus the probabilities that are used to prioritize the queue represent not the *total* path probability but rather the probability *given* the partial word sequence. Each path climbs up from a terminal node and back down to a next terminal node, with each new node adjusting the path probability by multiplying by a new conditional probability. The resulting conditional path probability for a next word represents the probability of that word in its syntactic role given all preceding words in *their* syntactic roles. With this strategy, a partial sentence does not become increasingly improbable as more and more words are added. [1].

Because of the sharing of common elements on the right hand side of rules, TINA can automatically generate new rules that were not explicitly provided. For instance, having seen the rule X ⇒ A B C and the rule X ⇒ B C D, the system would automatically generate two new rules, X ⇒ B C, and X ⇒ A B C D. Although this property can potentialy lead to certain problems with overgeneration, there are a number of reasons why it should be viewed as a feature. First of all, it permits the system to generalize more quickly to unseen structures. For example, having seen the rule AUX-QUESTION ⇒ AUX SUBJECT PREDICATE (as in "May I go?") and the rule AUX-QUESTION ⇒ HAVE SUBJECT LINK PRED-ADJECTIVE (as in "Has he been good?"), the system would also understand the forms AUX-QUESTION ⇒ HAVE SUBJECT PREDICATE (as in "Has he left?") and AUX-QUESTION ⇒ AUX SUBJECT LINK PRED-ADJECTIVE (as in "Should I be careful?").[2] Secondly it greatly simplifies the implementation, because rules do not have to be explicitly monitored during the parse. Given a particular parent and a particular child, the system can generate the allowable right siblings without having to note who the left siblings (beyond the immediate one) were. Finally, and perhaps most importantly, probabilities are established on arcs connecting sibling pairs regardless of which rule is under construction. In this sense the arc probabilities behave like the familiar word-level bigrams of simple recognition language models, except that they apply to siblings at multiple levels of the hierarchy. This makes the probabilities meaningful as a product of conditional probabilities as the parse advances to deeper levels of the parse tree and also as it returns to higher levels of the parse tree. All of the conditionals can be made to sum to one for any given choice, and everything is mathematically sound.

One negative aspect of such cross fertilization is that the system can potentially generalize to include forms that are agrammatical. For instance, the forms "Pick the box up" and "Pick up the box," if defined by the same LHS name, would allow the system to include rules producing forms such as "Pick up the box up" and "Pick up the box up the box!" This problem can be overcome either by giving the two structures different LHS names or by grouping "up the box" and "the box up" into distinct parent nodes, adding another layer to the hierarchy on the RHS. A third alternative is to include a PARTICLE slot among the features which, once filled, cannot be refilled. In fact, there were only a few situations where such problems arose, and they were always correctable.

3 Constraints and Gaps

This section describes how TINA handles several issues that are often considered to be part of the task of a parser. These include agreement constraints, semantic restrictions, subject-tagging for verbs, and long distance movement (often referred to as *gaps*, or the *trace*, as in "(which article)$_i$

[1]Some modification of this scheme will be necessary when the input stream is not deterministic. See [4] for a discussion of these very important issues regarding scoring in a best-first search.

[2]The auxiliary verb sets the mode of the main verb to be root or past participle as appropriate.

do you think I should read (t_i)?"). TINA is particulary effective in handling gaps. Complex cases of nested or chained gaps are handled correctly, and appropriately ill-formed gaps are rejected. The mechanism resembles the *Hold* register idea of ATN's [6] and the treatment of bounded domination metavariables in LFG's ([1], p. 235 ff), but I believe it is more straightforward than both of these.

3.1 Design Philosophy

Our approach to the design of a constraint mechanism is to establish a simple framework that is general enough to handle syntactic, semantic, and, ultimately, phonological constraints using identical functional procedures. The grammar is expressed as context-free rewrite rules without constraints. The constraints reside instead with the individual nodes of the tree that are established when the grammar is converted to a network structure. In effect, the constraint mechanism is thus reduced from a two-dimensional to a one-dimensional domain. Thus, for example, it would not be permitted to write an f-structure [1] equation of the form $\text{SUBJ}_{\text{Inf}} \Rightarrow \text{NP}$ associated with the rule VP $\Rightarrow$ VERB NP INF, to cover, "I told John to go." Instead, the NP node (regardless of its parent) would generate a CURRENT-FOCUS from its subparse, which would be passed along passively to the verb "go." The verb would then simply consult the CURRENT-FOCUS (regardless of its source) to establish its subject.

3.2 Constraints

Each parse node comes equipped with a number of slots for holding constraint information that is relevant to the parse. Included are person and number, case, determiner (DEFINITE, INDEFINITE, PROPER, etc.), mode (ROOT, FINITE, etc.), and semantic categories. These features are passed along from node to node: from parent to child, child to parent, and left-sibling to right-sibling. Certain nodes have the power to adjust the values of these features. The adjustment may take the form of an unconditional override, or it may involve a unification with the value for that feature passed to the node from a parent, sibling, or child. The filters are restricted in power in two important ways: 1) A filter can only operate on data that are available to the immediate parse node that instantiates the filter, and 2) A filter must be restricted in action to simple logical operations such as AND, SET, RESET, etc.

Some specific examples of constraint implementations will help explain how this works. Certain nodes specify person/number/determiner restrictions which then propagate up to higher levels and back down to later terminal nodes. Thus, for example, A NOUN-PL node sets the number to PLURAL, but only if the left sibling passes to it a description for number that includes PLURAL as a possibility (otherwise it dies, as in "each boats"). This value then propagates up to the SUBJECT node, across to the PREDICATE node, and down to the verb, which then must agree with PLURAL, unless its MODE is marked as non-finite. Any non-auxilliary verb node blocks the transfer of any predecessor person/number information to its right siblings, reflecting the fact that verbs agree in person/number with their subject but not their object.

A more complex example is a compound noun phrase, as in "Both John and Mary have decided to go." Here, each individual noun is singular, but the subject requires the plural form of "have." TINA deals with this by making use of a node category AND-NOUN-PHRASE, which sets the number constraint to PLURAL *for its parents*, and blocks the transfer of number information *to its children*. Some nodes also have special powers to set the mode of the verb either for their children or for their right-siblings. Thus, for example, "have" as an auxilliary verb sets mode to PAST-PARTICIPLE for its *right-siblings.* The category GERUND sets the mode to PRESENT-PARTICIPLE for its *children.* Whenever a PREDICATE node is invoked, the verb's mode has always been set by a predecessor.

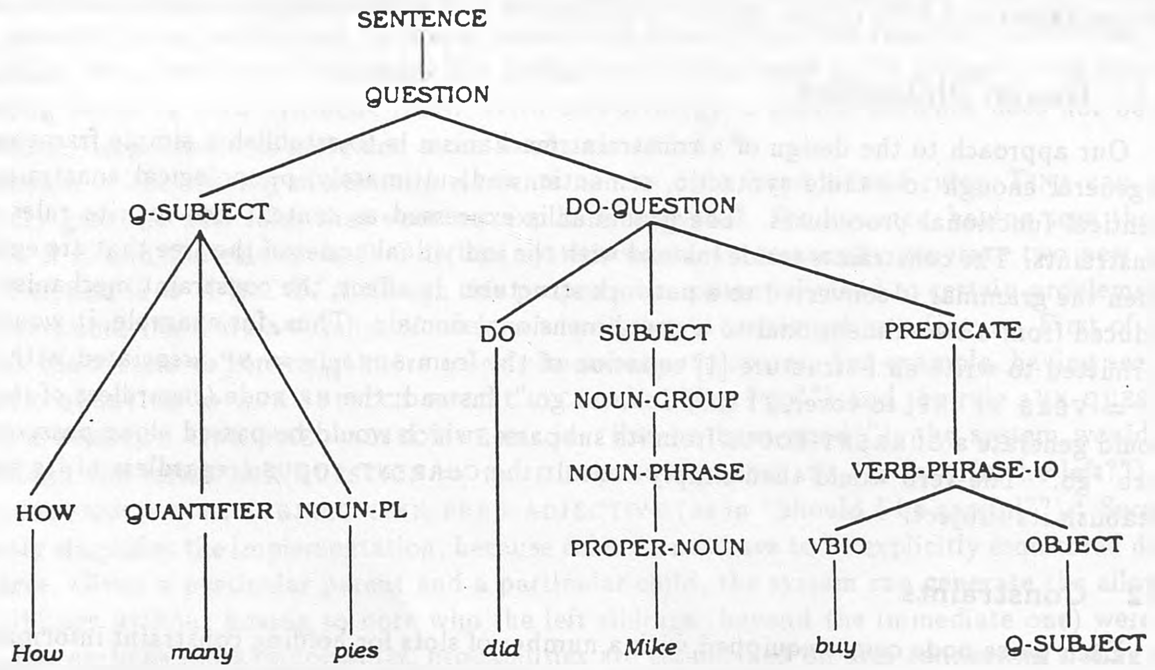


Figure 3: Example of a Parse Tree Illustrating a Gap.

3.3 Gaps

The mechanism to deal with gaps resembles in certain respects the *Hold* register idea of ATN's, but with an important difference, reflecting the design philosophy that no node can have access to information outside of its immediate domain. The process of getting into the *Hold* register (or the FLOAT-OBJECT slot, using my terminology) requires two steps, executed independently by two different nodes. The first node, the *generator*, fills the CURRENT-FOCUS slot with the subparse returned to it by its children. The second node, the *activator*, moves the CURRENT-FOCUS into the FLOAT-OBJECT position, for its children. It also requires that the FLOAT-OBJECT be absorbed somewhere among its descendants by a designated *absorber* node. The CURRENT-FOCUS only gets passed along to siblings and their descendants, and hence is unavailable to activators at higher levels of the parse tree. Finally, certain (*blocker*) nodes block the transfer of the FLOAT-OBJECT to their children.

A simple example will help explain how this works. For the sentence "(How many pies)$_i$ did Mike buy (t$_i$)?" as illustrated by the parse tree in Figure 3, the Q-SUBJECT "how many pies" is a generator, so it fills the CURRENT-FOCUS with its subparse. The DO-QUESTION is an activator; it moves the CURRENT-FOCUS into the FLOAT-OBJECT position. Finally, the object of "buy," an absorber, takes the Q-SUBJECT, as its subparse. The DO-QUESTION refuses to accept any solutions from its children if the FLOAT-OBJECT has not been absorbed. Thus, the sentence "How many pies did Mike buy the pies?" would be rejected. Furthermore, the same DO-QUESTION node deals with the yes/no question "Did Mike buy the pies?," except in this case there is no CURRENT-FOCUS and hence no gap.

More complicated sentences involving nested or chained traces, are handled staightforwardly by this scheme. For instance, the phrase, "(the violin)$_i$ that (these Sonatas)$_j$ are easy to play (t_j) on (t_i)" can be parsed correctly by TINA, identifying "Sonatas" as the object of "play" and "violin" as the object of "on." This works because the VERB-PHRASE-P-O, an activator, writes over the FLOAT-OBJECT "violin" with the new entry "Sonatas," but only for its children. The original FLOAT-OBJECT is still available to fill the OBJECT slot in the following prepositional phrase.

The example used to illustrate the power of ATN's [6], "John was believed to have been shot," also parses correctly, because the OBJECT node following the verb "believed" acts as both an absorber and a (re)generator. Cases of crossed traces are automatically blocked because the second CURRENT-FOCUS gets moved into the FLOAT-OBJECT position at the time of the second activator, overriding the preexisting FLOAT-OBJECT set up by the earlier activator. The wrong FLOAT-OBJECT is available at the position of the first trace, and the parse dies:

*(Which books)$_i$ did you ask John (where)$_j$ Bill bought (t_i) (t_j)?

The CURRENT-FOCUS slot is not restricted to nodes that represent nouns. Some of the generators are adverbial or adjectival parts-of-speech (POS). An absorber checks for agreement in POS before it can accept the FLOAT-OBJECT as its subparse. As an example, the question, "(How oily)$_i$ do you like your salad dressing (t_i)?" contains a Q-SUBJECT "how oily" that is an adjective. The absorber PRED-ADJECTIVE accepts the available float-object as its subparse, but only after confirming that POS is ADJECTIVE.

The CURRENT-FOCUS has a number of other uses besides its role in movement. It always contains the subject whenever a verb is proposed, including verbs that are predicative objects of another verb, as in "I want to go to China." In the case of passive voice, it contains 'NIL at the time of instantiation of the verb. It has also been found to be very effective for passing semantic information to be constrained by a future node, and it plays an integral role in pronoun-reference. These issues are addressed more fully in [4]

3.4 Semantic Filtering

In the most recent version of the parser, we implemented a number of semantic constraints using procedures that were very similar to those used for syntactic constraints. We found it effective to filter on the ACTIVE-NOUN's semantic category, as well as to constrain absorbers in the gap mechanism to require a match on semantics before they could accept a FLOAT-OBJECT. Semantic categories were implemented in a hierarchy such that, for example, RESTAURANT automatically inherits the more general properties BUILDING and PLACE. We also introduced semantically-loaded categories at the low levels of the parse tree. It seems that, as in syntax, there is a trade-off between the number of unique node-types and the number of constraint filtering operations. At low levels of the parse tree it seems more efficient to label the categories, whereas information that must pass through higher levels of the hierarchy is better done through constraint filters.

4 Practical Issues

Two unique practical aspects of TINA's design are its generation-mode capability and its ability to build a grammar automatically from a set of parsable sentences. We have found generation mode to be an essential tool for identifying overgeneration problems in the grammar. The ability to automatically provide a subset grammar for a set of sentences makes it easy to design a very specific, well constrained grammar, leading to improved performance in restricted-domain spoken language tasks.

Generation mode uses the same low-level routines as those used by the parser, but chooses only a single path based on the outcome of a random-number generator. Since all of the arcs have assigned probabilities, the parse tree is traversed by generating a random number at each node and deciding which arc to take based on the outcome, using the arc probabilities to weight the alternatives. Occasionally, the generator chooses a path which leads to a dead end, due to unanticipated constraints. In this case, it can back up and try again. Table 1 contains five examples of consecutively generated sentences. Since these were not selectively drawn from a larger set, they accurately reflect the current performance level. Because a number of semantic filtering operations have been applied within this task, most of the generated sentences are semantically as well as syntactically sound.

It is a two-step procedure to acquire a grammar from a specific set of sentences. The rule set is first built up gradually, by parsing the sentences one-by-one, adding rules and/or constraints as needed. Once a full set of sentences has been parsed in this fashion, the parse trees from the sentences are automatically converted to the set of rules used to parse each sentence. The training of both the rule set and the probability assignments is established directly from the provided set of parsed sentences; i.e. the parsed sentences *are* the grammar.

Another useful feature of TINA is that, as in LFG's, all unifications are nondestructive, and as a consequence explicit back-tracking is never necessary. Every hypothesis on the queue is independent of every other one, in the sense that activities performed by pursuing one lead do not disturb the other active nodes. This feature makes TINA an excellent candidate for parallel implementation. The control strategy would simply ship off the most probable node to an available processor.

Table 1: Sample sentences generated consecutively by the most recent version of TINA.

Do you know the most direct route to Broadway Avenue from here?
Can I get Chinese cuisine at Legal's?
I would like to walk to the subway stop from any hospital.
Locate a T-stop in Inman Square.
What kind of restaurant is located around Mount Auburn in Kendall Square of East Cambridge?

5 Discussion

This paper describes a new grammar formalism that addresses issues of concern in building a fully integrated speech understanding system. The grammar includes arc probabilities reflecting the frequency of occurrence of the syntactic structures within the domain. These probabilities are used to control the order in which hypotheses are considered, and are trained automatically from a set of parsed sentences, which makes it straightforward to tailor the grammar to a particular need. Ultimately, one could imagine the existence of a very large grammar that could parse almost anything, which would be subsetted for a particular task by simply providing it with a set of example sentences within that task.

I believe that, at the time a set of word candidates is proposed to the acoustic matcher of a recognizer, all of the constraint available from the restrictive influence of syntax, semantics, and phonology should have already been applied. The parse tree of TINA can be used to express various constraints ranging from acoustic-phonetic to semantic and pragmatic. Each parse node would contain slots for all kinds of constraint information – syntactic filters such as person, number and mode, semantic filters such as the permissible semantic categories for the subject/object of the hypothesized verb, and acoustic-phonetic filters (for instance, restricting the word to begin with a vowel if the preceding word ended in a flap, as in "Wha*t* is"). As the parse tree advances, it accumulates additional constraint filters that further restrict the number of possible next-word candidates. Thus the task of the predictive component is formulated as follows: given a sequence of words that has been interpreted to the fullest capability of the syntactic/semantic/phonological components, what are the likely words to follow, and what are their associated a priori probabilities?

While TINA's terminal nodes are lexical words, I believe that the nodes should continue down below the word level. Prefixes and suffixes alter the meaning/part-of-speech in predictable ways, and therefore should be represented as separate subword grammar units that can take certain specified actions. Below this level would be syllabic units, whose children are subsyllabic units such as onset and rhyme, finally terminating in phoneme-like units. Acoustic evidence would enter at several stages. Important spectral matches would take place at the terminal nodes, but duration and intonation patterns would contribute to scores at many higher levels of the hierarchy.

Three different task-specific versions of TINA have been implemented. The first one was designed to handle the 450 "phonetically rich" sentences of the TIMIT database [2]. The system was then ported to the DARPA Resource Management domain. A number of evaluation measures have been applied for these tasks, as described in [3]. Little else will be said here, except to note that perplexity was reduced nine-fold for the Resource Management task when arc probabilities established from the training data were incorporated, instead of using the equal-probability scheme. The latest version has been tailored to the new VOYAGER task, under development at MIT. This task involves navigational assistance within a geographical region. Our goal is to utilize constraints offered by both syntax and semantics so as to reduce perplexity as much as possible without sacrificing coverage. The parser is implemented on the Symbolics Lisp machine and runs quite efficiently. A sentence, entered in text form, is typically processed in a fraction of a second.

An effort to integrate the VOYAGER version of TINA with the SUMMIT speech recognition system [7] is currently underway. Two important issues are 1) how to combine the scores for the recognition component and the predictive component of the grammar, and 2) how to take advantage of appropriate pruning strategies to prevent an explosive search problem. The fully integrated spoken language system will use TINA both to constrain the recognition space and to provide an input to the back-end. Our current approach is to link together all words and all start-times that are equivalent within the parse, letting them proceed at a pace in accordance with the best-scoring word/time for the set. Viterbi pruning can take place within the recognizer, by having each terminal node initialize the recognizer with all the active phonetic nodes provided by its set of active hypotheses.

6 Acknowledgements

In developing this parser, I have benefitted from interactions with Lynette Hirschman, Mark Johnston, and Victor Zue. This research was supported by DARPA under Contract N00014-89-j-1332, monitored through the Office of Naval Research.

References

[1] Bresnan, J., ed., *The Mental Representation of Grammatical Relations*, MIT Press, 1982.

[2] Lamel, L., R.H. Kassel, and S. Seneff, "Speech Database Development: Design and Analysis of the Acoustic-Phonetic Corpus," DARPA Speech Recognition Workshop Proceedings, Palo Alto, CA, Feb 19-20, 1986.

[3] Seneff, S. "TINA: A Probablistic Syntactic Parser for Speech Understanding Systems," *Darpa Speech and Natural Language Workshop Proceedings*, Feb.1989.

[4] Seneff, S. "TINA: A Probablistic Syntactic Parser for Speech Understanding Systems," Laboratory for Computer Science Technical Report, forthcoming.

[5] Tomita, M., *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, Boston, MA, 1986.

[6] Woods, W.A., "Transition Network Grammars for Natural Language Analysis," Commun. of the ACM 13, 591-606, 1970.

[7] Zue, V., J. Glass, M. Phillips, and S. Seneff, "The MIT Summit Speech Recognition System: A Progress Report," *DARPA Speech and Natural Language Workshop Proceedings*, Feb.1989.

Connectionist Models of Language

James L. McClelland

Traditional models of language processing process language by rule. This approach faces two problems. First, there are difficulties in using the rules during processing, since often one rule must be pitted against another. In this case traditional approaches face the difficult problem of deciding which rule should win in such cases. Second, there are difficulties in acquiring rules, since it is often hard to know when a rule should be proposed, or when a sentence should be handled as one of many special cases.

In the connectionist approach my colleagues and I have been taking, language processing is viewed as a constraint satisfaction process. Each constituent of a sentence is viewed as imposing constraints on the representation of the state or event described by the sentence. During processing, as each constituent is encountered, it constrains the evolving representation of the sentence.

The knowledge that governs this constraint satisfaction is stored in the strengths of the connections among the units in a connectionist network. These connection strengths encode the knowledge that is traditionally encoded in the form of rules, but have the advantage that they are naturally capable of capturing constraints that differ in magnitude or degree. The acquisition of these connection strengths occurs through a connection adjustment process based on the back-propagation learning algorithm. The algorithm performs gradient descent in a measure of the extent to which the answers that the network gives to questions about the event described by a sentence actually match the probability that those answers are correct given the sentence. This algorithm is able to learn to assign the correct interpretations even when there are conflicting cues to the correct interpretation of a sentence.

To date this approach has been applied successfully to the processing of one-clause sentences. We have shown that it can learn to assign meanings to sentences containing vague and ambiguous words; that it fills in implicit arguments, and that it can use both word meaning and word order information correctly in making assignments of constituents to roles.

Current extensions focus on improving the rate of learning and on extending the approach to sentences of arbitrary complexity. In this regard we have recently established that a simpler variant of the model used for the comprehension of one-clause sentences is capable of learning, from a finite set of examples, to process all of the infinite corpus of sentences generated by a Finite State Automaton.

References

The following two Technical Reports give details of the research described above:

St. John, M. & McClelland, J. L. *Learning and applying contextual constraints in sentence comprehension.* AIP Technical Report, Departments of Psychology and Computer Science, Carnegie Mellon University.

Servan-Schreiber, D., Cleeremans, A., and McClelland, J. L. *Encoding sequential structure in simple recurrent networks.* Technical Report CMU-CS-88-183, Department of Computer Science, Carnegie Mellon University.

A Connectionist Parser Aimed at Spoken Language

Ajay Jain Alex Waibel

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We describe a connectionist model which learns to parse single sentences from sequential word input. A parse in the connectionist network contains information about role assignment, prepositional attachment, relative clause structure, and subordinate clause structure. The trained network displays several interesting types of behavior. These include predictive ability, tolerance to certain corruptions of input word sequences, and some generalization capability. We report on experiments in which a small number of sentence types have been successfully learned by a network. Work is in progress on a larger database. Application of this type of connectionist model to the area of spoken language processing is discussed.

This research was funded by grants from ATR Interpreting Telephony Research Laboratories and the National Science Foundation under grant number EET-8716324. The views and conclusions contained in this document are the authors' and should not be interpreted as representing the official policies, either expressed or implied, of ATR Interpreting Telephony Research Laboratories, the National Science Foundation, or the U.S. Government.

Introduction

Traditional methods employed in parsing natural language have focused on developing powerful formalisms to represent syntactic and semantic structure along with rules for transforming language into these formalisms. The builders of such systems must accurately anticipate and model all of the language constructs that their systems will encounter. Spoken language, with its weak grammatical structure, complicates matters. We believe that connectionist networks which *learn* to transform input word sequences into meaningful target representations offer advantages in this area.

Much work has been done applying connectionist computational models to various aspects of language understanding. Some researchers have used connectionist networks to implement formal grammar systems for use in syntactic parsing [1, 5, 10, 6]. These networks do not learn their grammars. Other work has focused on semantics [8, 11, 3, 2] but either ignored parsing, or the networks did not *learn* to parse. The networks presented in this paper learn their own "grammar rules" for transforming an input sequence of words into a target representation, and learn to use semantic information to do role assignment.

The remainder of this paper is organized as follows. First, there is a description of our network formalism. Next, we describe in detail a modest experiment in which a network was taught to parse a small class of sentences. We show how the network behaves with some novel sentences and with sentences that have been corrupted as in spoken language. Then, we show how we have generalized our architecture to model a much larger class of sentences and discuss the work as it currently stands. Lastly, we offer some concluding remarks about this work and suggest future directions.

Network Formalism

The most common type of deterministic connectionist network is a back propagation network [9]. Processing units are connected to each other, and each connection has an associated weight. Connections are unidirectional. Units have an activity value and an output value which is usually a sigmoidal function of the activity. For a connection from unit A to unit B, we define the stimulation along the connection to be the output value of unit A multiplied by the weight associated with the connection. A unit's activity is simply the sum of the stimulation along each of its input connections. A network learns input / output mappings by iteratively updating its weight values using a gradient descent technique.

Spoken language is an inherently sequential domain, and standard back propagation is not well suited to such a task. Recently, some recurrent extensions to back propagation where sequences of connections can form cycles have been proposed that can handle sequential input [4, 7]. Our networks extend these notions by explicitly accounting for time in our processing units. Units have activities which decay during each discrete time step by a constant factor. Thus, the activation of a unit can be built up over time from repetitive weak stimulation. Activity values are also damped to prevent unstable behavior. By gently "integrating" activities, the network has time to adapt to new information smoothly.

The activity of a unit is passed through a sigmoid squashing function to produce an output value as in standard back propagation. In addition, a value called the *velocity* is calculated. It is the rate of change of the output of a unit. Each connection in the network has two weights associated with it -- one for the output value and one for the velocity value. The velocity values are important to represent dynamic behavior which depends on changes in activation more than on absolute activation.

In order to facilitate symbolic processing, we use special units, called gating units, which gate the connections between groups of units. Fig. 1 diagrams the behavior of gating units. Slot C represents a particular word. It can be

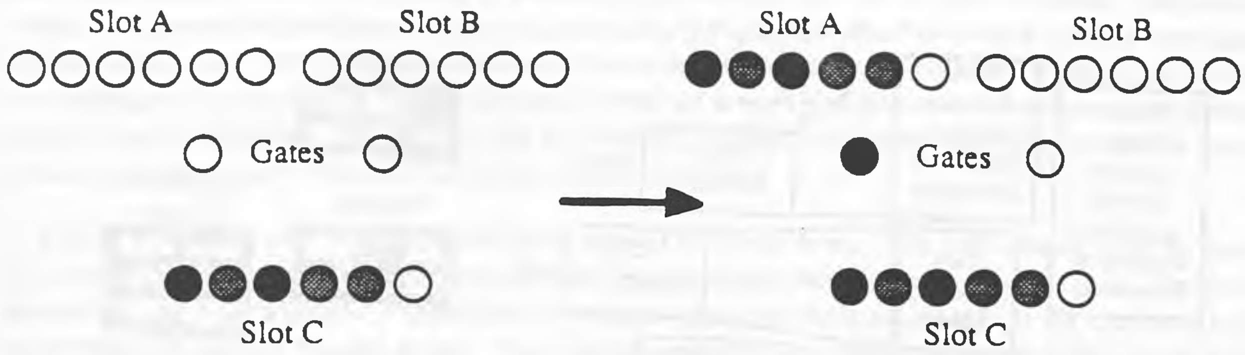


Figure 1: Gating Units

assigned to either slot A or slot B. The connections from the units of Slot C to both Slots A and B are gated by the two units below the slots (the connections are not shown here). In this case, the gating unit for slot A becomes active (see the right hand side of the diagram), and the pattern of activation across slot C becomes active across slot A. This type of assignment behavior can, in principle, be learned by a network without using gating units but is computationally wasteful.

Parsing Sentences

Our domain for this experiment consists of active and passive sentences consisting of up to 3 noun phrases and 2 verb phrases each. There are three roles for nouns to fill for each verb -- agent, patient, and recipient. The network also models subordinate and relative clause structure as well as prepositional attachment. The lexicon consists of 40 words which are divided into 7 classes -- nouns, verbs, adjectives, adverbs, auxiliaries, prepositions, and determiners. Each word is defined at most once within a class, but some words belong to two classes.

Words are represented as patterns of activation across a set of feature units. There are seven sets of feature units, one for each class of words. The pattern for a word consists of two parts: a feature part and an identification part. The feature part contains a small set of binary features encoding semantic information about a word. The identification part serves to disambiguate words which have identical feature parts (like a serial number). This allows one to add words to the lexicon which have the same features as existing words without any re-training of the network (the modifiable connections of the network do not connect to any identification units). Our 40 word lexicon is in a virtual sense much larger than 40 words. Each word is associated with one unit in the network which has hard-wired connections to excite the appropriate pattern across the feature units. A sentence is presented to the network by stimulating the word units corresponding to the words in the sentence each for a short time in sequence.

The target representation for sentences in the network has two levels: the Phrase level and the Structure level. Refer to Fig. 2 for a picture of the network structure. The Phrase level consists of groups of units called blocks, each of which contain a noun or a verb and its modifiers. A noun block has slots for a noun, two adjectives, a preposition, and a determiner. A verb block has slots for a verb, an auxiliary, and an adverb. There are 3 noun blocks and 2 verb blocks. Each block captures a phrase. The blocks are filled in order, with the first noun phrase occupying the first noun block, the second NP occupying the second noun block, and so on. The exact ordering relationship between the verb phrases and the noun phrases is lost in this representation, but due to the simplicity of the sentences this is not a problem.

The units in the Structure level describe the relationships between the phrases in the Phrase level the clauses they make up. There are six relationships possible:

- Agent: Noun block (NB) is agent of Verb block (VB). Group of 3 by 2 units.

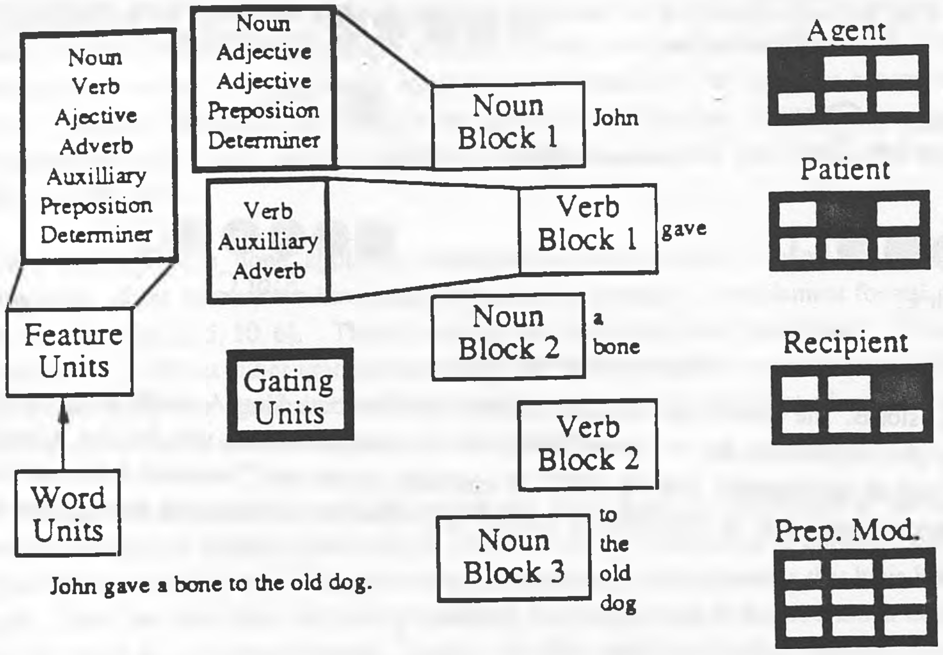


Figure 2: Network Structure

- Patient: NB is patient of VB. Group of 3x2.
- Recipient: NB is recipient of VB. Group of 3x2.
- Prepositional Modification: NB modifies other NB. Group of 3x3.
- Relative Clause: VB modifies NB. Group of 2x3.
- Subordinate Clause: VB subordinate to other VB. Group of 2x2.

The sentence, "John gave a bone to the old dog." is shown in Fig. 2.

In Fig. 2, the units shown in thick lined boxes have modifiable input connections -- they learn their behavior. The gating units at the Phrase level share a group of hidden units. These hidden units have connections from the feature units, the noun and verb blocks, and the gating units themselves. The Phrase level forms a recurrent subnetwork. The representation units of the Structure level also share a set of hidden units. These hidden units "see" all that the other set of hidden units see plus the structure representation units. The Structure level also forms a recurrent subnetwork. None of the hidden units have connections to the identification bit portions of the slots in the network.

The network whose performance we will characterize below was trained in two phases. First, the gating units in the Phrase level which are responsible for the behavior of the slots of the noun and verb blocks were trained. Their behavior is quite complex. They must learn to turn on when a word appears across the feature units for their slot (and their slot is supposed to be filled), stay on until the word disappears (even after the word has been assigned to the slot), turn off sharply, and stay off even when another word appears across their feature units. They must also learn to overwrite or empty out incorrectly assigned slots. Words get assigned incorrectly when they have representations in more than one class and there is insufficient information to disambiguate the usage. The word "was" has representations both as a verb and as an auxiliary verb. The network must assign it to both the auxiliary and the verb slots of the current verb block, and disambiguate the assignment when the next word comes in by either overwriting the verb slot with the real verb or emptying out the auxiliary slot.

The next phase involves adding the Structure level and training the structure representation units. The targets for the structure units are set at the beginning of a sentence and remain the same for the whole sentence. This forces the units to try to make decisions about sentence structure as early as possible; otherwise, they accumulate error signals. On the surface, it may seem that these units should have more or less monotonic behavior. However, the sentences in our domain do not necessarily contain sufficient information at word presentation time to make accurate decisions about the word's function. This coupled with the network's attempt to make decisions early causes the structure units to have surprisingly complicated activation patterns over time.

A set of 9 sentences was used to train the gating units of the Phrase level. They were selected to be the smallest set of sentences which would cover a reasonably rich set of sentences for training the Structure units. The network generalized very well to include "compositions" of sentence types from the initial set of 9. It was tolerant of varying word speed and silences between words. This is an important property, useful for integration of speech systems with natural language processing.

From this network, the Structure units were added. Eighteen sentences which were correctly processed at the Phrase level were chosen to train the Structure level. A variety of sentences was included. There were more active constructions than passive, more single clause than two clause sentences. Many different role structures were present in the training set. The network learned the set successfully.

Network Performance

The trained network displays several interesting properties on both the sentences in the training set and other new input sentences. A novel sentence is one which is not isomorphic to a training sentence modulo the identification bits of the words in the sentences. Thus, "Peter gave Fido the bone" is not different from "John gave Fido the bone." However, "Peter gave Fido the snake" is different since "snake" is animate, but "bone" is not.

The sentence "A snake ate the girl." is an example of the simplest type from the training set. The behavior of the key structure units corresponding to the roles of verb block 1 are shown in Fig. 3. Each box contains the indicated

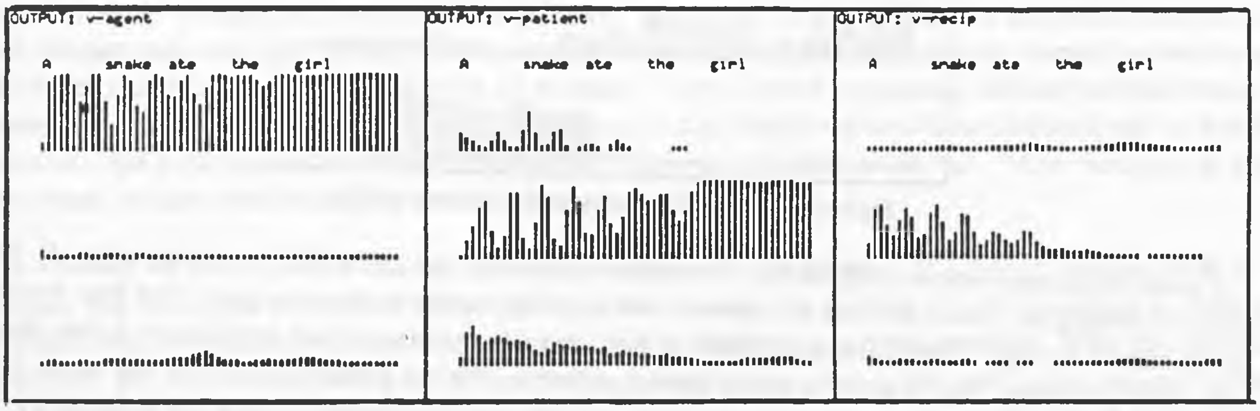


Figure 3: A snake ate the girl.

relationship units. The horizontal axis corresponds to time. Each word is presented for ten time steps. The first row of each box corresponds to the first noun phrase, the second to the second noun phrase and so on. The initial representation shows low activities for all of the relationship units. During presentation of the first word, the agent unit representing the first noun becomes quite active. It has not yet quite decided on its final value however, as can be seen by the oscillations. The other units are all either weakly active or oscillating. When the verb "ate" is presented, the agent unit corresponding to noun 1 fires strongly since it is now clear that the sentence is not a passive construction. Similarly, the patient unit for noun 2 becomes more active since "ate" is transitive. The last part of the sentence further verifies the correct representation. If "near the house" is appended to the sentence (forming a sentence not in the training set), it gets attached to "the girl".

In spoken language, determiners and other short function words tend to be poorly articulated. This is indeed a persistent problem for speech recognition systems, as it leads to word deletions. Despite such deletions, our network makes appropriate role assignments with such sentences as "Snake ate girl." The role assignment is agent / patient as in the uncorrupted sentence. Non-speech interjections are also possible as in, "A snake (ahh) ate the girl." A speech recognition system could easily interpret the non-speech "ahh" as "a". Our network puts the non-speech "a" in the determiner slot of the second noun block, and then overwrites it with "the". The result is a good parse of the ill-formed sentence. Similarly, simple stuttering does not adversely affect network performance in many cases. It is important to note that this behavior was not taught in any way to the network.

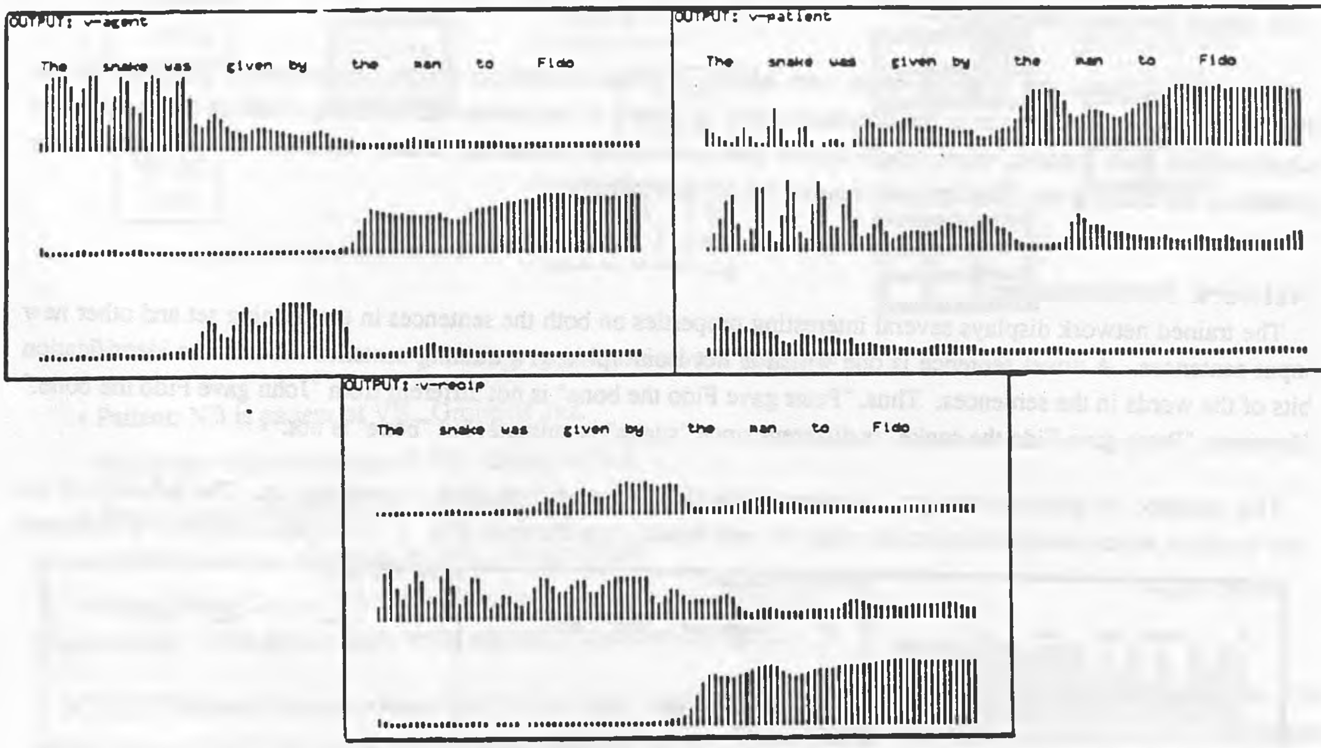


Figure 4: The snake was given by the man to Fido.

A more complicated sentence is given by, "The snake was given by the man to Fido." as shown in Fig. 4. It was not in the training set. There was only one sentence with a similar structure in the training set: "The bone was given by the man to the dog." They differ significantly in that "snake" is animate and less significantly in their detailed noun phrase structure. Fig. 4 shows a similar display as before. For the duration of the first two words of this sentence, the units behave as they did in the previous one. However, the passive construction indicated by "was given" causes the agent unit for the first noun to decay and the agent unit for the third noun to grow. This is because several other passive sentences in the training set were structured where the third noun was the agent. The word "by" causes the agent units to move toward their final positions and indicate "by the man" is the agent block. The recipient and patient units make their final decisions with a little residual oscillation at this time as well. At the arrival of "to Fido" finally, the correct parse is locked up.

In the previous example, the network seized the preposition "by" to make its role assignments. The network is also able to use semantic cues from words in the absence of meaningful function words. Fig. 5 show the network's behavior on the sentence, "A snake was given an apple by John." Here, the network must rely on the semantic features of "snake" and "apple" to make the proper role assignment. Since "snake" is animate, and apple is not, their

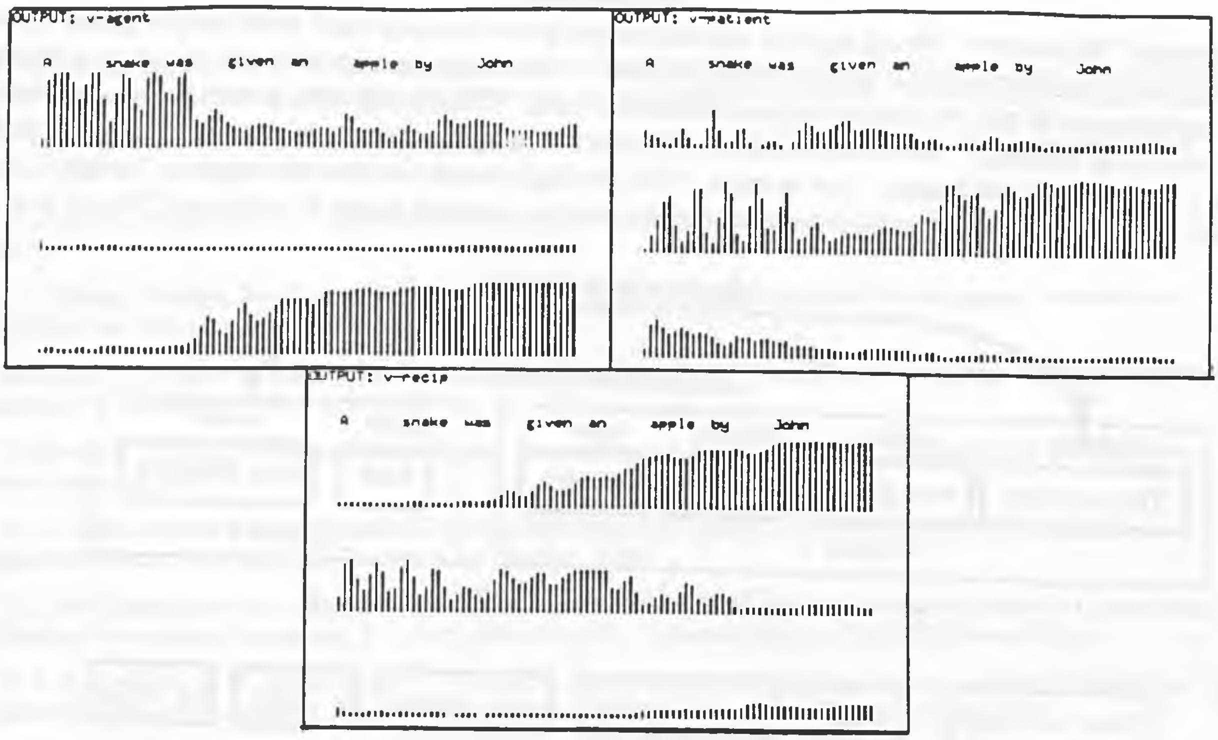


Figure 5: A snake was given an apple by John.

roles are assigned as recipient and patient, respectively. This occurs when "an apple" is processed. The opposite role assignment is made in, "A bone was given the dog by John." The heuristic learned by the network is that inanimate objects are preferred as patients over animate objects.

Single clause sentences dominated the training set, but a few two clause sentences were presented to explore the network's ability to learn the interactions among clauses. Since the network architecture allowed for only three noun phrases with two verb phrases, these sentences were quite simple. The network learned to recognize subordinate clauses as in, "John slept after he ate an apple." It also learned to recognize sentence terminal relative clauses as in, "John kissed the girl who slept." Generalization capability in the two clause sentences was not tested extensively due to the paucity of sentences constructible within the constraints of the task. Minor variations in the noun phrase structure from the training sentences were properly treated.

In summary, we have observed four key features in the network's performance. It is able to combine syntactic, semantic, and word order information effectively to perform its task. The network tries to be predictive, making decisions about the structure of the sentence as soon as sufficient information becomes available. When the network is uncertain, the units oscillate among sets of possible future states in a way that is detectable by the network via the velocity weights. The network responds reasonably to sentences which have been modified from those in its training set.

Extending the Architecture

The architecture described above is still limited in its present form. To extend and scale it to more complex sentences and to allow for a more flexible representation, we have designed a more general architecture. The new architecture is modular, hierarchical, and recurrent. It has four levels: Phrase, Clause Structure, Clause Roles, and Interclause. The Phrase level is analogous to that of the network described earlier, but differs in three important ways. The words in the lexicon all share the same feature units instead of being separated into classes. The phrases are not separated into verb and noun blocks; the input sentence is parsed into blocks of contiguous words which form phrases. The sentence "The old dog who was sleeping was given a bone by John" would be split up into "(The old dog) (who) (was sleeping) (was given) (a bone) (by John)". The Clause Structure level uses the evolving Phrase level representation to split the sentence into its constituent clauses: "(The old dog) (was given) (a bone) (by John)" and "(who) (was sleeping)". The Clause Roles level does the role assignment and noun phrase attachment for each of the clauses as they are mapped. For example, "(The old dog)" would be called the recipient, "(a bone)" the patient etc. The final level, Interclause, encodes the fact that the embedded clause is relative to "(The old dog)".

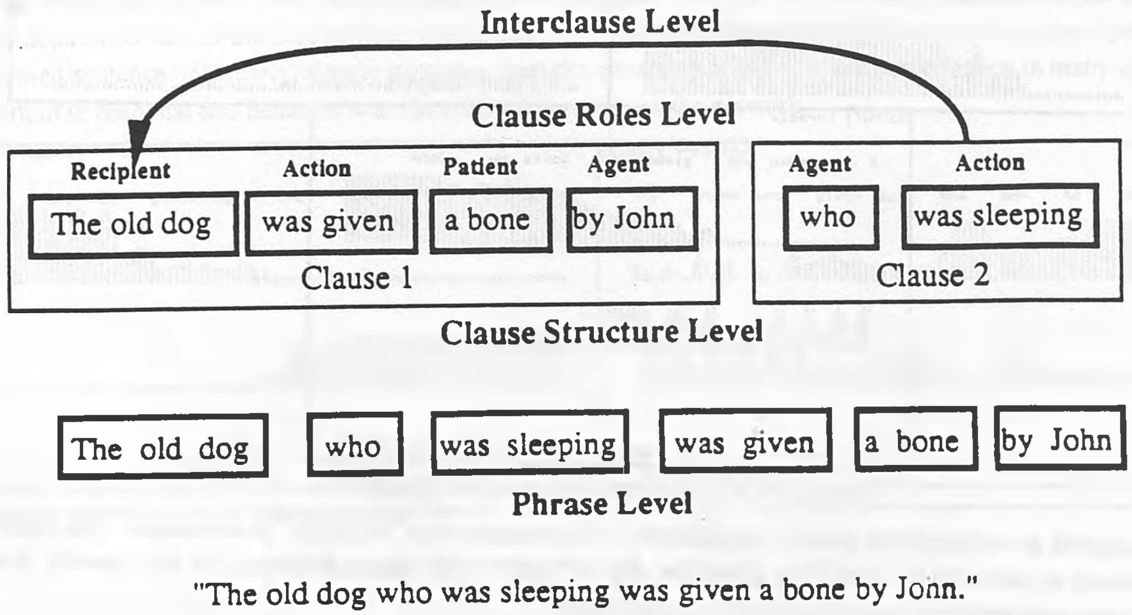


Figure 6: New Representation

Fig. 6 shows the representation of this sentence.

At the Phrase level and the Clause Roles level, the network consists of horizontally replicated modules which are trained on all of the phrases and clauses from a set of sentences. This artificially creates the effect of a very large training set on a very large network without the cost associated with building such networks. The Clause Structure and Interclause levels cannot be treated in this manner since they deal with whole sentence structure.

We are currently exploring such a network on a set of over 200 sentences. These include sentences with passive constructions, center embedded clauses, and some lexical ambiguity. Preliminary results on the individual modules comprising the network have been encouraging, and we hope to begin testing on the fully integrated network shortly.

Conclusion

We have presented a connectionist architecture which learns to incrementally parse sentences. Our networks exhibit behavior that could potentially be extremely useful for the integration of speech and language processing. Tolerance to corruptions of input including ungrammaticality, word deletions and insertions, and varying word speed are all desirable for speech applications. Connectionist networks appear to be less rigid than more formal systems thereby allowing them to handle a wider variety of sentences given only a limited initial set of examples. Their ability to learn complex dynamical behaviors from diverse knowledge sources makes them well suited for speech processing applications.

References

1. E. Charniak and E. Santos. A Connectionist Context-Free Parser Which is not Context-Free But Then It is not Really Connectionist Either. Proceedings of the Ninth Annual Conference of the Cognitive Science Society, 1987.

2. G. Cottrell. Connectionist Parsing. Proceedings of the Seventh Annual Conference of the Cognitive Science Society, 1985.

3. G. Cottrell. *A Connectionist Approach to Word Sense Disambiguation*. Ph.D. Th., University of Rochester, May 1985.

4. J. L. Elman. Finding Structure in Time. Tech. Rept. 8801, Center for Research in Language, University of California, San Diego, 1988.

5. M. Fanty. Context Free Parsing in Connectionist Networks. Tech. Rept. TR174, Computer Science Department, University of Rochester, November, 1985.

6. T. Howells. VITAL: A Connectionist Parser. Proceedings of the Tenth Annual Conference of the Cognitive Science Society, 1988.

7. M. I. Jordan. Serial Order: A Parallel Distributed Processing Approach. Tech. Rept. 8604, Institute for Cognitive Science, University of California, San Diego, 1986.

8. J. L. McClelland and A. H. Kawamoto. Mechanisms of Sentence Processing: Assigning Roles to Constituents. In *Parallel Distributed Processing*, J. L. McClelland and D. E. Rumelhart, Ed., The MIT Press, 1986.

9. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Internal Representations by Error Propagation. In *Parallel Distributed Processing*, J. L. McClelland and D. E. Rumelhart, Ed., The MIT Press, 1986.

10. B. Selman and G. Hirst. A Rule-Based Connectionist Parsing System. Proceedings of the Seventh Annual Conference of the Cognitive Science Society, 1985.

11. D. Waltz and J. Pollack. "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation". *Cognitive Science 9* (1985).

Massively Parallel Parsing in ΦDMDIALOG: Integrated Architecture for Parsing Speech Inputs

Hiroaki Kitano, Teruko Mitamura and Masaru Tomita
Center for Machine Translation
Carnegie Mellon University
Pittsburgh, PA 15213 U.S.A.

Abstract

This paper describes the parsing scheme in the ΦDMDIALOG speech-to-speech dialog translation system, with special emphasis on the integration of speech and natural language processing. We propose an integrated architecture for parsing speech inputs based on a parallel marker-passing scheme and attaining dynamic participation of knowledge from the phonological-level to the discourse-level. At the phonological level, we employ a stochastic model using a transition matrix and a confusion matrix and markers which carry a probability measure. At a higher level, syntactic/semantic and discourse processing, we integrate a case-based and constraint-based scheme in a consistent manner so that a priori probability and constraints, which reflect linguistic and discourse factors, are provided to the phonological level of processing. A probability/cost-based scheme in our model enables ambiguity resolution at various levels using one uniform principle.

1. Introduction

This paper discusses a method of integrating speech recognition and natural language processing. In order to develop speech-based natural language systems such as a speech-to-speech translation system and a speech input natural language interface, an integration of speech recognition and natural language processing is essential, because it improves the recognition rate of the speech inputs. Improvement of the recognition rate can be attained by an integration of natural language processing with speech recognition, providing a more appropriate assignment of *a priori probability* to each hypothesis and imposes more constraints to reduce search space. Thus, the quality of the *language model* is an important factor. Since our goal is to create accurate translation from speech input, a sophisticated parsing and discourse understanding scheme are necessary. We propose an architecture for parsing speech inputs that integrates speech (phonological-level processing) and natural language processing with full syntactic/semantic analysis and discourse understanding.

In our system, we assume that an acoustic processing device provides a symbol sequence for a given speech input. In this paper, we assume that a phoneme-level sequence is provided to the system[1]. The phoneme sequence given from the phoneme recognition device contains substitution, insertion and deletion of phonemes, as compared to a correct transcription which contains only expected phonemes. We call such a phoneme sequence a *noisy phoneme sequence*. The task of phonological-level processing is to activate a hypothesis as to the correct phoneme sequence from this noisy phoneme sequence. Inevitably, multiple hypotheses can be generated due to the stochastic nature of phoneme recognition errors. Thus, we want each hypothesis to be assigned a measure of its being correct. In the stochastic models of speech recognition, a probability of each hypothesis is determined by $P(y|h) \times P(h)$. $P(y|h)$ is the probability of a series of input sequence being observed when a hypothesis h is articulated. $P(h)$ is an a priori probability of the hypothesis derived from the language model. Apparently, when phonological-level processing is the same, the system with a sophisticated language model attains a higher recognition rate, because a priori probability differenciates between hypotheses of high acoustic similarity which would otherwise lead to confusion. At the same time, we want to eliminate less-plausible hypotheses as early as possible so that the search space is kept within a certain size. We use syntactic/semantic and discourse knowledge to impose constraints which reduce search space, in addition to the probability-based pruning within the phonological level.

[1]We use Matsushita Institute's Japanese speech recognition system[Morii et. al., 1985] for a current implementation.

2. ΦDMDIALOG Project

2.1. Overview

ΦDMDIALOG is a speech-to-speech dialog translation system based on a massively parallel computational model [Kitano, 1989b] [Kitano et. al., 1989b] [2]. It accepts speaker-independent continuous speech inputs. Some of the significant features of ΦDMDIALOG include:
I. Use of a hybrid parallel paradigm as a basic computational scheme, which is an integrated model of a direct memory access (DMA) type of a massively parallel marker passing scheme and a connectionist network;
II. Dymanic utilization of knowledge from morphophonetics to discourse by distributively encoding this knowledge in a memory network on which actual computations are performed;
III. Integration of case-based and constraint-based processing to capture linguistically complex phenomena without losing cognitive realities;
IV. A cost-based ambiguity resolution scheme which applies to all levels of ambiguity (from phoneme recognition to discourse context selection)[Kitano et. al., 1989a];
V. Almost concurrent parsing and generation, so that a part of a sentence can be translated before the whole sentence is parsed[Kitano, 1989a].
The philosophy behind our model is to view parsing as a process on a dynamic system where the law of energy conservation, entropy production and other laws of physics can be effective analogies. We also demand that our model be consistent with psycholinguistic studies.

2.2. A Baseline Algorithm

We employ the hybrid parallel paradigm in order to model two distinct aspects of the parsing: information building and hypothesis selection. In the hybrid parallel paradigm, a parallel marker-passing scheme and a connectionist network are integrated and computations are performed directly in a memory network. Knowledge from the morphophonetic level to the discourse level is represented as a memory network which is consists of nodes and links. Several types of nodes are in the memory network.
Concept Sequence Class (CSC) captures configurational patterns of linguistic phenomena such as phoneme sequences, concept sequences and plan sequences. CSCs have an internal structure. The internal structure is composed of a label, IS-A links, a sequence, presuppositions, effects, and constraint equations. This structure is same for all CSCs except CSCs in the phonological layer.
Concept Class (CC) represents concepts such as phonemes, concepts, and plans.
Concept Instance (CI) are instances of CCs. They are used to represent discourse entities[Webber, 1983] and instance of utterances.
Nodes are connected by labelled links. Abstraction links (IS-A) and compositional links (PART-OF) are typical types of links. The memory network is organized in a hierarchical manner. There are hierarchies of nodes representing concepts from specific instances (using CIs) to general concepts (using CCs) and hierarchies of structured nodes representing relations of concepts which are indexed into relevant concepts and specific instances (using CIs and their links). When CSCs represent specific cases, they are already co-indexed to the specific instances in the memory network. Abstract CSCs hold various constraints described as constraint equations, presuppositions and effects. These abstract CSCs are instantiated during parsing and newly created specific CSCs are indexed into the memory network as cases of utterance. Parsing with abstract CSCs is computationally more expensive than parsing with cases, but it maintains productivity of the knowledge.

Three types of markers (A-, P-, and C-Markers) are used for parsing. Two other types of markers, G- and V-Markers are used for generation; thus they are not described in this paper.
Activation Markers (A-Markers) contain information including discourse entities, features and cost. They propagate upward through abstraction links.
Prediction Markers (P-Markers) predict possible next activations. They contain binding lists (a list of role-instance pairs binded so far), a measure of cost, and linguistic and pragmatic constraints.
Contextual Markers (C-Markers) are used as an alternative to a connectionst network and indicate contextual priming. C-Markers are not used when the connectionist network is fully deployed.

[2] Φ indicates that our system is a speech input system. This notation is a tradition of the Center for Machine Translation. DM implies that the system was initially designed as a direct memory access (DMA) based system. However, our system evolved differently from the DMAP[Riesbeck and Martin, 1985] and now DM implies both DMA and *dynamics modeling* which reflects our philosophy of viewing a cognitive process as a dynamic process governed by the laws of physics. DIALOG means that our system translates dialogs.

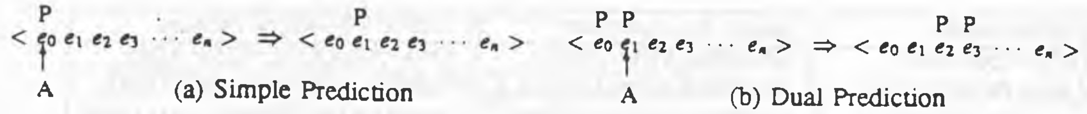


Figure 1: Movement of P-Markers

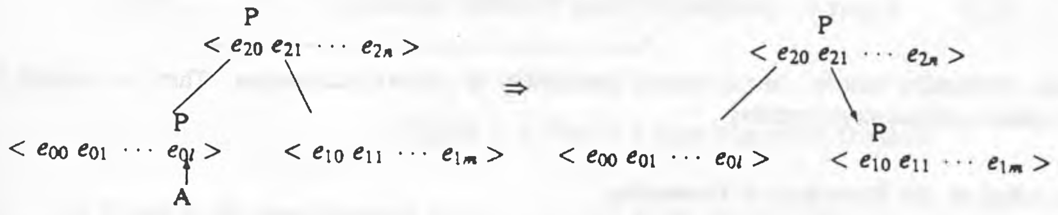


Figure 2: Movement of P-Markers in Layered Sequences

A basic cycle of our algorithm is as follows:

1. **Activation:**
   For each input symbol, a corresponding node is activated and an A-Marker is created. A unit of input may be either a phoneme or a word, depending on the input device. The A-Marker is passed up through IS-A links. The A-Marker contains information relevant to the processing of that layer.
2. **A-P-Collision:**
   When an A-Marker and a P-Marker collide at a certain element of a CSC, the P-Marker is moved to the next possible concept element of the CSC. At this stage, constraints are checked.
3. **Prediction:**
   As a result of moving P-Markers to the next possible element of the CSC, predictions are made describing possible next inputs.
4. **Recognition (Network Modification and Information Propagation):**
   When the CSC is accepted, (1) the memory network may be modified as a side-effect, and (2) an A-Marker containing aggregated information is passed up through IS-A links.

The movements of P-Markers on a CSC are illustrated in figure 1. In (a), a P-Marker (initially located on e_0) is hit by an A-Marker and moved to the next element. In (b), two P-Markers are used and moved to e_2 and e_3. In the dual prediction, two P-Markers are placed on elements of the CSC (on e_0 and e_1). This dual prediction is used for phonological processing.

Figure-2 shows movement of a P-Marker on the layers of CSCs. When the P-Marker at the last element of the CSC gets an A-Marker, the CSC is accepted and an A-Marker is passed up to the element in the higher layer CSC. Then, a P-Marker on the element of the CSC gets the A-Marker, and the P-Marker is moved to the next element. At this time, a P-Marker which contains information relevant to the lower CSC is passed down and placed on the first element of the lower CSC. This is a process of accepting one CSC and predicting the possible next word and syntactic structure.

3. Phonological Parsing

This section describes phonological-level activities. We assume a noisy phoneme sequence, as shown in Figure 3, to be the input of the phonological-level processing. In order to capture the stochastic nature of speech inputs, we adopt a probabilistic model similar to that used in other speech recognition research. First, we describe a simple

kaigi ni sanka shitai nodesu	*youshi ha arimasuka*	*oname wo onegai shimasu*
DAI*I*IPAUTAQPAINO*EKU	BJOHIRAARI*ATAWA	O*A*AEJOORE*EISI*AS@
BAII*IPAA=KAS@PAINODUSU	JOSJUWAARINAOQZAA	WO*A*AEJOORE*EEHJANA
BAII*I*IPAU=KAIQPAI*O*ESU	IOUSIWAARIMAUQKA	WONA*AEJOBO*E*EIHJAH@
KAIIMIPAA=KAS@PEEI*ODESU	JOOSIHAKARI*AUQKA	O*A*AEJO*O*E*EEISINAKU
KAI*I*IPAA=ZAS@PAIWO*USJU	IOOSJUWAWARI*AACA	O*A*AEJOO*E*EEIHJAZU

Figure 3: Examples of Noisy Phoneme Sequences

model using a static probability matrix. In this model, probability is context-independent. Then, we extend the model to capture context-dependent probability.

3.1. The Organization of the Phonological Processing

The algorithm described as a baseline algorithm is deployed on phonetic-level knowledge. In the memory network, there are CSCs representing the phoneme sequence for each lexical entry. The dual prediction method is used in order to handle deletion of a phoneme.

We use a probabilistic model to capture the stochastic nature of speech processing. Probability measures involved are: a priori probability given by the language model, a confusion probability given by a confusion matrix, and a transition probability given by a transition martix.

A priori probability is derived from the language model and is a measure of which phoneme sequence is likely to be recognized. A method of deriving a priori probability is described in the section on syntax/semantic parsing and discourse processing.

A confusion matrix defines the output probability of a phoneme when an input symbol is given. Given an input sign i_i, the confusion matrix a_{ij} determines the probability that the sign i_i will be recognized as a phoneme p_j. It is a measure of the distance between symbols and phonemes as well as a measure of the cost of hypotheses that interpret the symbol i_i as the phoneme p_j. In the context-dependent model, the confusion matrix will defined as a_{ijk} which gives a probability of a symbol i_i to be interpreted as a phoneme p_j at a transition t_k. We call such matrix a *dynamic confusion matrix.*

A transition matrix defines the transition probability which is a probability of a symbol i_{i+1} to follow a symbol i_i. For an input sequence $i_0\ i_1\ \cdots\ i_n$, the a priori probability of transition between i_0 and i_1 is given by b_{i_0,i_1}. Since we have a finite set of input symbols, each transition can be indexed as t_k. The transition probability and the confusion probability are intended to capture the context-dependency of phoneme substitutions – a phenomena whereby a certain phoneme can be actually articulated as other phonemes in certain environments.

3.2. Context-Independent Model

First, we explain our algorithm using a simple model whose confusion matrix is context-independent. Later, we describe the context-dependent model which uses a dynamic confusion matrix. Initially, P-Markers contain a priori probability (π_l) given by the language model. In ΦDMDIALOG, the language model reflects full natural language knowledge from syntax/semantics to discourse. The P-Markers are placed on each first and second element of CSCs representing expected phoneme sequences. For an input symbol i_i, A-Markers are passed up to all phoneme nodes that have a probability(b_{ij}) greater than the threshold (Th). When a P-Marker, which is at i-th element, and an A-Marker collide, the P-Marker is moved to the i+1-th and i+2-th elements of the sequence (This is a dual prediction). When the next input symbol i_{i+1} generates an A-Marker that hits the P-Marker on the i+1-th element, the P-Marker is moved using the dual prediction method. The probability density measure computed on the P-Marker is as follows:

$$ppm(i) = ppm(i-1) \times a_{i_{i-2},i_{i-1}} \times b_{p_{i-2},i_{i-2}} \quad (1)$$

$$ppm(0) = \pi_l \quad (2)$$

where $ppm(i)$ is a probability measure of a P-Marker at the i-th element of the CSC which is a probability of the input sequence being recognized as a phoneme sequence traced by the P-Marker.

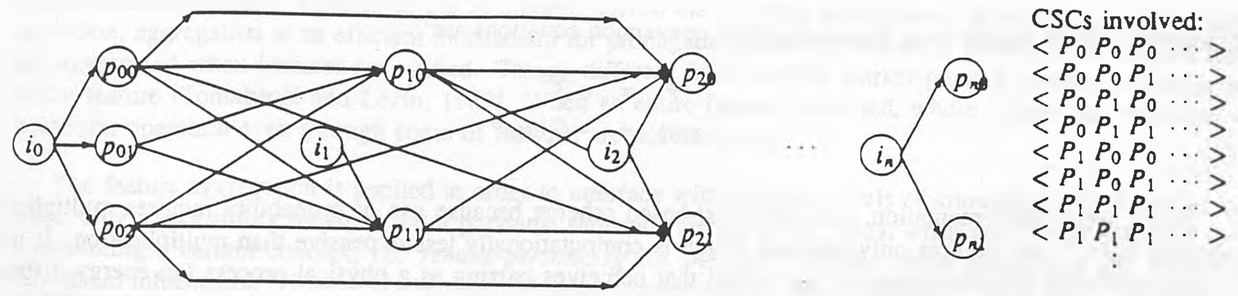

CSCs involved:
$< P_0\, P_0\, P_0 \cdots >$
$< P_0\, P_0\, P_1 \cdots >$
$< P_0\, P_1\, P_0 \cdots >$
$< P_0\, P_1\, P_1 \cdots >$
$< P_1\, P_0\, P_0 \cdots >$
$< P_1\, P_0\, P_1 \cdots >$
$< P_1\, P_1\, P_1 \cdots >$
$< P_1\, P_1\, P_1 \cdots >$
i_0
p_{00}
p_{01}
p_{02}
i_1
p_{10}
p_{11}
i_2
p_{20}
p_{21}
i_n
p_{n0}
p_{n1}

Figure 4: A Part of a State-Transition Diagram

In Figure-4, an input sequence is $i_0\ i_1\ \cdots\ i_n$. p_{ij} in the diagram denotes a phoneme P_j at i-th element of the CSC. p_{ij} is a state rather than an actual phoneme, and P_j in the CSC refers to the actual phoneme. P-Markers at p_{00}, p_{01}, p_{02}, P-Markers on the 0-th element of the CSCs referring P_0, P_1, and P_2, respectively, are hit by A-Markers. Eventually, P-Markers are moved to the next element of CSCs. For instance, p_{00} will move to $p_{10}, p_{11}, p_{20}, p_{21}$ depending on which CSC the P-Marker is placed on. Probabilities are computed with each movement. A P-Marker at p_{11} has the probability π_0. When the P-Marker received an A-Marker from i_1, the probability is re-computed and it will be $\pi_0 \times\ b_{i_0,p_{00}}\ \times\ a_{p_{i_0},p_{i_1}}$. Transitions such as $p_{00} \rightarrow p_{21}$ and $p_{00} \rightarrow p_{20}$ insert an extra phoneme which does not exist in the input sequence. Probability for such transitions are computed in such a way as: $\pi_0 \times\ b_{i_0,p_{00}}\ \times\ a_{i_0,\phi} \times\ b_{i_2,p_{20}}\ \times\ a_{\phi,i_2}$. A P-Marker at p_{10} does not get an A-Marker from i_1 due to the threshold. In such cases, a probability measure of the P-Marker is re-computed as $\pi_0 \times\ b_{i_0,p_{00}}\ \times\ a_{i_0,noise}$. This represents a decrease of probability due to an extra input symbol.

P-Markers at the last element (p_n) and one before the last (p_{n-1}) are involved in the word boundary problem. When a P-Marker at p_n is hit by an A-Marker, the phoneme sequence is accepted and an A-Marker that contains the probability and the phoneme sequence is passed up to the syntactic/semantic-level of the network. Then, the next possible words are predicted using syntactic/semantic knowledge, and P-Markers are placed on the first and the second element of the phoneme sequence of the predicted words. When a P-Marker at p_{n-1} is hit by an A-Marker, the P-Marker is moved to p_n and, independently, the phoneme sequence is accepted, due to the dual prediction, and the first and the second elements of the predicted phoneme sequences get P-Markers.

3.3. The Context-Dependent Model

The context-dependent model can be implemented by using the dynamic confusion matrix. The algorithm described above can be applied with some modificaitions. First, A-Markers are passed up to phonemes whose maximun output probability is above the threshold. Second, output probability used for probability calculation is defined by the dynamic confusion matrix.

$$ppm(i) \;=\; ppm(i-1) \times\ a_{i_{i-2},i_{i-1}}\ \times\ b_{p_{i-2},i_{i-2},k} \tag{3}$$

where k denotes a transition from i_{i-2} to i_{i-1}. It is interesting that our context-dependent model is quite similar to the *Hidden Markov Model (HMM)* when the transition of the state of P-Markers are synchronously determined by, for example, certain time intervals. We can implement a forward-passing algorithm and the Viterbi algorithm [Viterbi, 1967] using our model. This implies that when we decide to employ the HMM as our speech recognition model, instead of a current speech input device, it can be implemented within the framework of our model.

3.4. Probability Cost Equality

Since we have been using the cost-based ambiguity resolution scheme [Kitano et. al., 1989a], the equivalency of the probabilistic approach and the cost-based approach need to be discussed. Our motivation in introducing the cost-based scheme was to perceive parsing as a dynamic process. Thus the hypothesis with the least cost, hence minimum workload, is selected as the best hypothesis. When a stochasity is introduced, the process that requires more workload is less likely to be chosen. Thus, qualitatively, higher probability means less cost and lower probability means higher cost. Probability/cost conversion equations are[3]:

$$P = e^{\frac{-cost}{C}} \quad (4)$$
$$cost = -C \log P \quad (5)$$

In the actual implementation, we use a cost-based scheme because use of probability requires multiplication, whereas use of cost requires only addition which is computationally less expensive than multiplication. It is also a straightforward implementation of our model that perceives parsing as a physical process (an energy dispersion process). Thus, in the cost-based model, we introduce an *accumlated acoustic cost (AAC)* as a measure of cost which is computed by:

$$aac(i) = aac(i-1) + cc_{i_{i-1},p_{i-1}} + tc_{i_{i-2},i_{i-1}} - pe \quad (6)$$

where $aac(i)$, $cc_{i_{i-1},p_{i-1}}$, $tc_{i_{i-2},i_{i-1}}$, and pe are an AAC measure of the P-Marker at i-th element, confusion cost between i_{i-1} and p_{i-1}, transition cost between i_{i-2} and i_{i-1}, and phonetic energy, respectively. Phonetic energy reflects an influx of energy from external acoustic energy.

4. Syntactic/Semantic Parsing

Unlike most other language models employed in speech recognition research, our language model is a complete implementation of a natural language parsing system. Thus, complete semantic interpretations, constraint checks, ambiguity resolution and discourse interpretations are performed. The process of prediction is a part of parsing in our model, thereby attaining an integrated architecture of speech input parsing. In syntactic/semantic processing, the central focus is on how to build the informational content of the utterance and how to reflect syntactic/semantic constraints at phonological-level activities. Throughout the syntactic/semantic-level and discourse-level, we use a method to fuse constraint-based and case-based approaches. In our model, the difference between a constraint-based process and a case-based process is a level of abstraction; the case-based process is specific and the constraint-based process is more abstract. The constraint-based approach is represented by various unification-based grammar formalisms [Pollard and Sag, 1987] [Kaplan and Bresnan, 1982]. We use semantic grammar which combines syntactic and semantic constraints[4]. In our model, propagation of features and unification are conducted as a *feature aggregation* by A-Markers and *constraints satisfaction* performed by operations involving P-Markers. The case-based approach is a basic feature of our model. Specific cases of utterances are indexed in the memory network and reactivated when similar utterances are given to the system. One of the motivations for the case-based parsing is that it encompasses *phrasal lexicons* [Becker, 1975][5]. The scheme described in this section is applied to discourse-level processing and attains an integration of the syntactic/semantic-level and the discourse-level.

4.1. Feature Aggregation

Feature aggregation is an operation which combines features in the process of passing up A-Markers so that minimal features are carried up. Due to the hierarchical organization of the memory network, features which need to be carried by A-Markers are different depending on which level of abstraction is used for parsing. When knowledge of cases is used for parsing, features are not necessary because this knowledge is already indexed to specific discourse entities. Features need to be carried when more abstract knowledge is used for parsing. For example, the parsing of a sentence *She runs* can be handled at different levels of abstraction using the same mechanism. The word *she* refers to a certain discourse entity so that very specific case-based parsing can directly access a memory which recalls previous memory in the network. Since previous cases are indexed into specific discourse entities, the activation can directly identify which memory to recall. When this word *she* is processed in a more abstract level such as PERSON, we need to check features such as number and gender. Thus, these features need to be contained in the A-Marker. Further abstraction requires more features to be contained in the A-Marker. Therefore, the case-based process and the constraint-based process is treated in one mechanism. Aggregation is a cheap operation since it simply adds

[3]The equations are based on the Maxwell-Boltzmann distribution $P = e^{\frac{-E}{kT}}$.

[4]This does not preclude use of unification grammar formalism in our system. In fact, we are now developing a cross-compiler that compiles grammar rules written in LFG into our network. Designing of a cross-compiler from HPSG to our network is also underway.

[5]Discussions on benefits of phrasal lexicons for parsing and generation are found in [Riesbeck and Martin, 1985] [Hovy, 1988].

new features to existing features in the A-Marker. Given the fact that unification is a computationally expensive operation, aggregation is an efficient mechanism for propagating features because it ensures only minimal features are aggregated when features are unified. This is different from another marker-passing scheme which carries an entire feature [Tomabechi and Levin, 1989]. When an entire feature is carried, whole features are involved in the unifiction operation even through some of features are not necessary.

The feature aggregation is applied in order to interface with different levels of knowledge. At the phonological level, only a probability measure and a phoneme sequence are involved. Thus, when an A-Marker hits a CC node representing a certain concept, i.e. *female-person-3sg* for *she*, the A-Marker does not contain any linguistically significant information. However, when the A-Marker is passed up to more abstract CC nodes, i.e. *person*, linguistically significant features are contained in the A-Marker and unnecessary information is discarded. When a sentence is analyzed at the syntactic/semantic-level, a propositional content is established and is passed up to the discourse-level by an A-Marker, and some linguistic information which is necessary only within the syntactic/semantic-level is discarded.

4.2. Constraint Satisfaction

Constraint is a central notion in modern syntax theories. Each CSC has *constraint equations* which define the constraints imposed for that CSC depending on their level of abstraction. CSCs representing specific cases do not have contraint equations since they are already instanciated and the CSCs are indexed in the memory network. The more abstract the knowledge is the more they contain constraint equations. Feature structures and constraint equations interact in two stages. At the prediction stage, if a P-Marker placed on the first element of the CSC already contains a feature structure that is non-nil, the feature structure determines, according to the constraint equations, possible feature structures of A-Markers that subsequent elements of the CSC can accept. At an A-P-Collision stage, a feature structure in the A-Marker is tested to see if it can meet what was anticipated. If the feature structure passes this test, information in the A-Marker and the P-Marker is combined and more precise predictions are made on what can be acceptable in the subsequent element. For *She runs*, we assume a constraint equation *(AGENT NUM = ACTION NUM)* associated with a CSC, for example, <AGENT ACTION>. When a P-Marker initially has a feature structure that is nil, no expectation is made. In this example, at an A-P-Collision, an A-Marker has a feature structure containing (NUM = 3s) constraints for the possible verb form which can follow, because the feature in the A-Marker is assigned in the constraint equation so that *(AGENT NUM 3s)* requires *(ACTION NUM 3s)*. This guarantees that only a verb form *runs* can be legitimate[6]. When predicting what comes as a *ACTION*, P-Markers can be passed down via IS-A links and only lexical entries that meet *(ACTION NUM 3s)* can be predicted. When we need to relax grammatical constraints, P-Markers can be placed on every verb form, but assign higher a priori probabilities for those which meet the constraint. A unification operation can be used to conduct operations described in this section. As a result of parsing at the syntactic/semantic-level, the propositional content of the utterance is established. Since our model is a memory-based parsing model, the memory network is modified to reflect what was understood as a result of previous parsing.

4.3. Prediction

From the viewpoint of predicting the next hypothesis at the phonological level, case-based parsing provides the most specific prediction and gives high a priori probability. Prediction by more abstract knowledge provides less specific predictions and gives weaker a priori probability compared to case-based prediction. Thus, we have a set of hypotheses with strong preferences predicted by the case-based process and a set of hypotheses (this includes hypotheses predicted by the case-based process) predicted by the constraint-based process. Of course, the strength of the preference is dependent on the level of abstraction the parsing has required. Even in the constraint-based process, if the level of abstraction is low, the prediction has strength comparable to the case-based prediction.

5. Integration of Discourse Knowledge

At the discourse-level, the focus is on how to recognize the intention of the utterance, interpret discourse phenomena and predict next possible utterances. ΦDMDIALOG uses discourse knowledge such as (1) discourse plans, and (2)

[6] When we use abstract notation such as NP or VP, the same mechanism applies and captures linguistic phenomena.

discourse entities and their relations. We use hierarchical discourse plan sequences, represented by CSCs[7], to represent and provide specificity as well as productivity of discourse plans. Hierarchical discourse plan sequences represent possible sequences of utterance plans which may be actually performed by each speaker. Plan hierarchies are organized for each participant of the dialog in order to capture complex dialog often taking place in a mixed-initiative dialog. Each element of the plan sequence represents a domain-specific instance of a plan or an utterance type [Litman and Allen, 1987] which can be dynamically derived from abstract dialog knowledge and domain knowledge. Abstract plan sequences are close to plan schemata described in [Litman and Allen, 1987] since they represent very generic constraints as well as the relationship between an utterance and a domain plan. There is also knowledge for the discourse structure[Cohen and Fertig, 1986] [Grosz and Sidner, 1985]. When an element of the plan sequence of this abstraction is activated, the rest of the elements of the plan sequence have constraints imposed which are derived from the information given to the activated elements. This ensures coherence of the discourse. When a plan sequence case is activated, it simply predicts the next plan elements because these specific plan sequences are regarded as records of past cases and, thus, most constraints are already imposed and the sequence is indexed according to the specific constraints. In addition, use of order constraints of CSC representations allows us to handle order-freeness of subdialog conversations. Furthermore, unlike scripts or MOPs[Schank, 1982], a plan sequence has an internal structure which enables our model to impose constraints which ensure coherency of the discourse processing.

As a result of the discourse understanding, possible next utterances can be predicted. P-Markers are passed down to nodes representing these utterances. Eventually, they reach the phonological level and give a priori probability to each hypothesis. Similar to predictions from syntactic/semantic-level, the strength of the prediction is dependent upon the level of abstract knowledge involved.

6. A Cost-based Ambiguity Resolution Scheme

A cost-based disambiguation scheme is a method of evaluating each hypothesis based on the cost assigned to it. Costs are added when (1) phonemes are replaced, inserted, or dropped during recognition of noisy speech inputs (we use a cost converted from a probability measure at the phonological-level), (2) a new instance is created, (3) a concept without contextual priming is used, or (4) constraints are assumed when using CSCs. Costs are subtracted when (1) a concept with discourse prediction is used, or (2) a concept with contextual priming is used. Basic equations are:

$$CSC_i = \sum_j CC_{ij} + \sum_k constraints_k + bias_i \quad (7)$$

$$CC_j = LEX_j + instantiateCI - priming_j \quad (8)$$

$$LEX_l = -C \log P \quad (9)$$

where CC_{ij}, $constraints_k$, $bias_i$ denote a cost of the j-th element of CSC_i, a cost of assuming the k-th constraints, and the lexical preference of CSC_i, respectively. LEX_j, $instantiateCI$, $priming_j$ denote a cost of the lexical node LEX_j, a cost of creating new CI by referential failure, and contextual priming, respectively. LEX_j is a cost converted from the probability measure at the phonological level as described earlier. The accumlated acoustic cost, computed by the equation (6), can be used instead of converting probability by equation (9). Then, the cost-based scheme is adopted at every level of processing. In the cost-based disambiguation scheme, we choose the least costly hypothesis based on the above equations.

Our model parses utterances under a given context. Thus, the cost assigned to a certain hypothesis is not always the same. It is dependent on the context; that is, the initial conditions of the system when the utterance is entered. The initial condition of the system is determined based on the previous course of discourse. The major factors are the state of the memory network modified as a result of processing previous utterances, contextual priming, and predictions from discourse plans. The memory network is modified based on the knowledge conveyed by the series of utterances in the discourse as described briefly in the previous section. Contextual priming is imposed either by using a C-Marker passing or by a connectionist network. The mechanism of assigning preference is based on top-down prediction using discourse knowledge. Such prediction provides a priori probability π_l at the phonological-level.

[7]This means that order-strict or order-free constraints apply in determining the order of the plan sequence.

The cost-based ambiguity resolution scheme is applied to the reference problem including definite and indefinite reference, pronoun reference, etc. We use activation/cost-based reference where each reference hypothesis incures cost and the least-cost hypothesis will be selected. The cost for each hypothesis is computed based of activation levels of each discourse entities and semantic restrictions. The method does not assume a layered network [Tomabechi and Levin, 1989] and, thus, we can coherently handle problems including the reference to the related objects.

7. Preliminary Evaluations and Discussions

Currently, ΦDMDIALOG is being tested on the conference registration domain based on simulated telephone conversation experiments by ATR. The use of dialog-level knowledge has proven to be effective in in reducing the perplexity of the task. We took as an example a small test set from the ATR corpus, and the perplexity of this task with no prediction knowledge was 247.0. Using sentential level knowledge this figure was reduced to 19.7, and using dialog level knowledge it was reduced to 2.4. However, the problem is that (1) the domain of our experiment is relatively small, and (2) when we cover more complex discourse, prediction from the discourse-level may be less specific. We are now evaluating our model with larger test sets.

We employ the probabilistic model for the following reason: the use of phonological knowledge alone, such as phonological rules and distinctive feature theory, cannot sufficiently cope with the stochastic nature of speech recognition. However, phonological knowledge would be useful for analyzing and estimating probability matrices. By contrasting feature types, such as voicing, instead of collecting all the phonemic data, we would reduce the amount of data needed for building the probability matrices[Church, 1987].

The hierarchical organization of the memory network is a key feature in integrating constraint-based and case-based processing. Although we suffer from some overhead by concurrently parsing one sentence at different levels of abstraction, the capability of handling both specific and abstract knowledge in a consistent manner seems more significant. The feature aggregation method is a useful technique to keep overhead to a minimum.

The implementation of ΦDMDIALOG on a parallel machine is an interesting topic. We believe the benefits of our model can be best explored with parallel machines and that its implementation may be relatively straightforward. Actually, a part of our model has been implemented on a custom VLSI[Kitano, 1988].

8. Related Works

Several efforts have been made to integrate speech and natural language processing. [Tomabechi et. al., 1988] attempts to extend the marker-passing model to speech input. Their model uses *environment* without probabilistic measure which would allow environmental rules to be applied. Since misrecognitions are somewhat stochastic, lack of the probability measure seems a shortcoming in their model. The MINDS system [Young et. al., 1989] is an attempt to integrate speech and natural language processing implementing layered prediction. They reported that use of layered prediction involving discourse knowledge reduced the perplexity of the task. This is consistent with our claim. [Church, 1987] discusses speech recognition using phonetic knowledge such as environment and a distinct feature matrix. We share similar motivations, but we try to incorporate this knowledge in a probabilistic model. [Saito and Tomita, 1988] [Kita et. al., 1989] and [Chow and Roukos, 1989] are examples of approaches to integrate speech with unification-based parsing, but, unfortunately, discourse processing has not been incorporated. Marker-passing models of parsing such as [Riesbeck and Martin, 1985] and [Tomabechi and Levin, 1989] captured only one side of parsing (case-based or constraint-based), in contrast to our model which incorporates both aspects in one scheme.

9. Conclusion

This paper describes a method of speech-natural language integration in ΦDMDIALOG. The probability/cost-based model is used to capture the stochastic nature of speech inputs. The language model in our model is a parser itself and directly connected to the phoneme processing by means of cost measures, a priori probability, and constraints to limit search space. Addition of the discourse understanding scheme further improved the power of the language model to constrain and predict phonological processes. As a result, reduction of the perplexity was observed and the recognition rate was improved. Feature aggregation in the hierarchically organized memory network was a useful scheme to integrate case-based and constraint-based parsing. The parallel marker-passing approach seems a viable alternative for designing an integrated architecture for parsing speech inputs.

Acknowledgement

We would like to thank members of the Center for Machine Translation for useful discussions. Especially, Hideto Tomabechi, Hiroaki Saito and Jaime Carbonell helped us with insightful advice. Discussions on speech recognition with Sheryl Young and Wayne Ward were especially useful. Lyn Jones was patient enough to proofread this paper for us. We also would like to thank ATR Interpreting Telephony Laboratories for allowing us to use a corpus of the conference registration domain for our research. Matsushita Research Institute is allowing us to use their speech recognition system.

Appendix: Implementation

ΦDMDIALOG has been implemented on IBM-RT-PC which runs CMU-CommonLisp on the Mach operating system and HP-9000 runs HP-CommonLisp. Speech recognition and synthesis devices (Matsushita Research Institute's Japanese speech recognition device and DECTalk) are connected to perform real-time speech-to-speech translation.

References

[Becker, 1975] Becker, J. D., *The Phrasal Lexicon*, Bolt, Beranek and Newman Technical Report 3081, 1975.

[Chow and Roukos, 1989] Chow, Y.L. and Roukos, S., "Speech Understanding using a Unification Grammar," In *Proc. of ICASSP- IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1989.

[Church, 1987] Church, K., *Phonological Parsing in Speech Recognition*, Kluwer Academic Publishers, 1987.

[Cohen and Fertig, 1986] Cohen, P. and Fertig, S., "Discourse Structure and the Modality of Communication," *International Symposium on Prospects and Problems of Interpreting Telephony*, 1986.

[Grosz and Sidner, 1985] Grosz, B. and Sidner, C., "The Strucure of Discourse Structure," *CSLI Report No. CSLI-85-39*, 1985.

[Hovy, 1988] Hovy, E. H., *Generating Natural Language Under Pragmatic Constraints*, Lawrence Erlbaum Associates, 1988.

[Kaplan and Bresnan, 1982] Kaplan, R. and Bresnan, J., "Lexical-Functional Grammar: A Formal System for Grammatical Representation," In Bresnan (Ed.), *The Mental Representation of Grammatical Relations*, MIT Press, 1982.

[Kita et. al., 1989] Kita, K., Kwabata, T. and Saito, H., "HMM Continuous Speech Recognition using Predictive LR Parsing," In *Proc. of ICASSP - IEEE International Conference on Acoustic, Speech, and Signal Processing*, 1989.

[Kitano, 1988] Kitano, H., "Multilingual Information Retrieval Mechanism using VLSI," In *Proceedings of RIAO-88*, 1988.

[Kitano, 1989a] Kitano, H., "A Massively Parallel Model of Natural Language Generation for Interpreting Telephony: Almost Concurrent Processing of Parsing and Generation," In *Proceedings of the Second European Conference on Natural Language Generation*, 1989.

[Kitano, 1989b] Kitano, H., "A Model of Simultaneous Interpretation: A Massively Parallel Model of Speech-to-Speech Dialog Translation," In *Proceedings of the Annual Conference of the International Association for Knowledge Engineers*, 1989.

[Kitano et. al., 1989a] Kitano, H., Tomabechi, H. and Levin, L., "Ambiguity Resolution in DMTRANS PLUS," In *Proceedings of the Fourth Conference of the European Chapter of the Association for Computational Linguistics*, 1989.

[Kitano et. al., 1989b] Kitano, H., Tomabechi, H., Mitamura, T. and Iida, H., "A Massively Parallel Model of Speech-to-Speech Dialog Translation: A Step Toward Interpreting Telephony," In *Proceedings of the European Conference on Speech Communication and Technology (EuroSpeech-89)*, 1989.

[Kitano et. al., ms.] Kitano, H., Iida, H., Mitamura, T. and Tomabechi, H., Manuscript, "An Integrated Discourse Understanding Model for Interpreting Telephony under a Direct Memory Access Paradigm," Carnegie Mellon University, 1989.

[Litman and Allen, 1987] Litman, D. and Allen, J., "A Plan Recognition Model for Subdialogues in Conversation," *Cognitive Science 11* (1987), 163-200.

[Morii et. al., 1985] Morii, S., Niyada, K., Fujii, S. and Hoshimi, M., "Large Vocabulary Speaker-Independent Japanese Speech Recognition System," In *Proceedings of ICASSP - IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1985.

[Pollard and Sag, 1987] Pollard, C. and Sag, I., *Information-based Syntax and Semantics*, volume 1, CSLI, 1987.

[Riesbeck and Martin, 1985] Riesbeck, C. and Martin, C., "Direct Memory Access Parsing," *Yale University Report 354*, 1985.

[Saito and Tomita, 1988] Saito, H. and Tomita, M., "Parsing Noisy Sentences," In *Proceedings of COLING-88*, 1988.

[Schank, 1982] Schank, R., *Dynamic Memory: A theory of learning in computers and people*, Cambridge University Press, 1982.

[Tomabechi et. al., 1988] Tomabechi, H., Mitamura, T. and Tomita, M., "Direct Memory Translation for Speech Input: A Massively Parallel Network for Episodic/Thematic and Phonological Memory," In *Proceedings of the International Conference on Fifth Generation Computer Systems*, 1988.

[Tomabechi and Levin, 1989] Tomabechi, H. and Levin, L., "The Head-driven Massively-parallel Constraint Propagation: Head-features and subcategorization as interacting constraints in associative memory," In *Proceedings of CogSci-89*, 1989.

[Viterbi, 1967] Viterbi, A.J., "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," In *IEEE Transactions on Information Theory* IT-13(2): 260-269, April, 1967.

[Webber, 1983] Webber, B., "So What Can We Talk About Now?" In *Computational Models of Discourse*, The MIT Press, 1983.

[Young et. al., 1989] Young, S., Ward, W. and Hauptmann, A., "Layering Predictions: Flexible use of Dialog Expectation in Speech Recognition," In *Proceedings of IJCAI-89*, 1989.

Parallel Parsing Strategies in Natural Language Processing

Anton Nijholt

Faculty of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

ABSTRACT

We present a concise survey of approaches to the context-free parsing problem of natural languages in parallel environments. The discussion includes parsing schemes which use more than one traditional parser, schemes where separate processes are assigned to the 'non-deterministic' choices during parsing, schemes where the number of processes depends on the length of the sentence being parsed, and schemes where the number of processes depends on the grammar size rather than on the input length. In addition we discuss a connectionist approach to the parsing problem.

1. Introduction

In the early 1970's papers appeared in which ideas on parallel compiling for programming languages and parallel executing of computer programs were investigated. In these papers parallel lexical analysis, syntactic analysis (parsing) and code generation were discussed. At that time various multi-processor computers were introduced (CDC 6500, 7600, STAR, ILLIAC IV, etc.) and the first attempts were made to construct compilers which used more than one processor when compiling programs. Slowly, with the advent of new parallel architectures and the ubiquitous application of VLSI, interest increased and presently research on parallel compiling and executing is widespread. Although more slowly, a similar change of orientation occurred in the field of natural language processing. However, unlike the compiler construction environment with its generally accepted theories, in natural language processing no generally advocated – and accepted – theory of natural language analysis and understanding is available. Therefore it is not only the desire to exploit parallelism for the improvement of speed but it is also the assumption that human sentence processing is of an inherently parallel nature which makes computer linguists and cognitive scientists turn to parallel approaches for their problems.

Parallel parsing methods have been introduced in the areas of theoretical computer science, compiler construction and natural language processing. In the area of compiler construction these methods sometimes refer to the properties of programming languages, e.g. the existence of special keywords, the frequent occurrence of arithmetic expressions, etc. Sometimes the parsing methods that have been introduced were closely related to existing and well-known serial parsing methods, such as LL-, LR-, and precedence parsing. Parallel parsing has often been looked upon as deterministic parsing of sentences with more than just a single serial parser. However, with the massively parallel architectures that have been designed and constructed, together with the possibility to design special-purpose chips for parsing and compiling in mind, also the well-known methods for general context-free parsing have been re-investigated in order to see whether they allow parallel implementations. Typical results in this area are $O(n)$-time parallel parsing algorithms based on the Earley or the Cocke-Younger-Kasami parsing methods. In order to study complexity results for parallel recognition and parsing of context-free languages theoretical computer scientists have introduced parallel machine models and special subclasses of the context-free languages (bracket languages, input-driven languages). Methods that have been introduced in this area aim at obtaining lower bounds for time and/or space complexity and are not necessarily useful from a more practical point of view. A typical result in this area tells us that context-free language recognition can be done in $O(\log^2 n)$ time using n^6 processors, where n is the length of the input string.

In the area of natural language processing many kinds of approaches and results can be distinguished. While some researchers aim at cognitive simulation, others are satisfied with high performance language systems. The first-mentioned researchers may ultimately ask for numbers of processors and connections between processors that approximate the number of neurons and interconnections in the human brain. They model human language processing with connectionist models and therefore they are interested in massive parallelism and methods which allow low degradation in the face of local errors. In connectionist and related approaches to parsing and natural language analysis the traditional methods of language analysis are often replaced by strongly interactive distributed processing of word senses, case roles and semantic markers. A more modest use of parallelism may also be useful. For any system which has to understand natural language sentences it is necessary to distinguish different levels of analysis (see e.g. Nijholt[1988], where we distinguish the morphological, the lexical, the syntactic, the semantic, the referential and the behavioral level) and at each level a different kind of knowledge has to be invoked. Therefore we can distinguish different tasks: the application of morphological knowledge, the application of lexical knowledge, etc. It is not necessarily the case that the application of one type of knowledge is under control of the application of any other type of knowledge. These tasks may interact and at times they can be performed simultaneously. Therefore processors which can work in parallel and which can communicate with each other may be assigned to these tasks in order to perform this interplay of multiple sources of knowledge. Finally, and independent of a parallel nature that can be recognized in the domain of language processing, since operating in parallel with a collection of processors can achieve substantial speed-ups, designers and implementers of natural language processing systems will consider the application of available parallel processing power for any task or subtask which allows that application.

In this paper various approaches to the problem of parallel parsing will be surveyed. We will discuss examples of parsing schemes which use more than one traditional parser, schemes where 'non-deterministic' choices during parsing lead to separate processes, schemes where the number of processes depends on the length of the sentence being parsed, and schemes where the number of processes depends on the grammar size rather than on the input length. Our aim is not to give a complete survey of methods that have been introduced in the area of parallel parsing. Rather we present some approaches that use ideas that seem to be characteristic for many of the parallel parsing methods that have been introduced.

2. From One to Many Traditional Serial Parsers

Introduction

As mentioned in the introduction, many algorithms for parallel parsing have been proposed. Concentrating on the ideas that underlie these methods, some of them will be discussed here. For an annotated bibliography containing references to other methods see Nijholt et al[1989]. Since we will frequently refer to *LR-parsing* a few words will be spent on this algorithm. The class of LR-grammars is a subclass of the class of context-free grammars. Each LR-grammar generates a *deterministic* context-free languages and each deterministic context-free language can be generated by an LR-grammar. From an LR-grammar an LR-parser can be constructed. The LR-parser consists of an LR-table and an LR-routine which consults the table to decide the actions that have to be performed on a pushdown stack and on the input. The pushdown stack will contain symbols denoting the *state* of the parser. As an example, consider the following context-free grammar:

1. S → NP VP	4. PP → *prep NP
2. S → S PP	5. VP → *v NP
3. NP → *det *n	

With the LR-construction method the LR-table of Fig. 1 will be obtained from this grammar. It is assumed that each input string to be parsed will have an endmarker which consists of the $-sign.

An entry in the table of the form 'sh *n*' indicates the action 'shift state *n* on the stack and advance the input pointer'; entry 're *n*' indicates the action 'reduce the stack using rule *n*'. The

state	*det	*n	*v	*prep	$	NP	PP	VP	S
0	sh3					2			1
1				sh5	acc		4		
2			sh6					7	
3		sh8							
4				re2	re2				
5	sh3					9			
6	sh3					10			
7				re1	re1				
8			re3	re3	re3				
9				re4	re4				
10				re5	re5				

Fig. 1 LR-parsing table for the example grammar.

entry 'acc' indicates that the input string is accepted. The right part of the table is used to decide the state the parser has to enter after a reduce action. In a reduce action states are popped from the stack. The number of states that are popped is equal to the length of the right hand side of the rule that has to be used in the reduction. With the state which becomes the topmost symbol of the stack (0–10) and with the nonterminal of the left hand side of the rule which is used in the reduction (S, NP, VP, or PP) the right part of the table tells the parser what state to push next on the stack. In Fig. 2 the usual configuration of an LR-parser is shown.

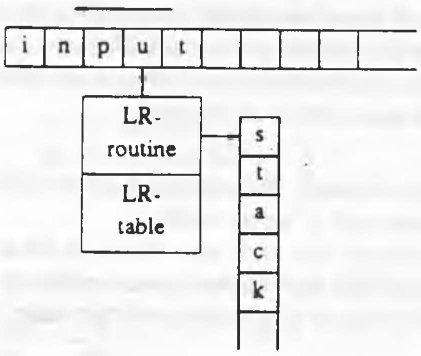


Fig. 2 LR-parser.

More than One Serial Parser

Having more than one processor, why not use two parsers? One of them can be used to process the input from left to right, the other can be used to process the input from right to left. Each parser can be assigned part of the input. When the parsers meet the complete parse tree has to be constructed from the partial parse trees delivered by the two parsers. Obviously, this idea is not new. We can find it in Tseytlin and Yushchenko[1977] and it appears again in Loka[1984]. Let $G = (N, \Sigma, P, S)$ be a context-free grammar. For any string $\alpha \in V^*$ let α^R denote the reversal of α. Let $G^R = (N, \Sigma, P^R, S)$ be the context-free grammar which is obtained from G by defining $P^R = \{i.\ A \rightarrow \alpha^R \mid i.\ A \rightarrow \alpha \in P\}$. It is not difficult to see that, when we start a left-to-right top-down construction of a parse tree with respect to G at the leftmost symbol of a string w and a bottom-up right-to-left construction of a parse tree with respect to G^R at the rightmost symbol of w, then – assuming the grammar is unambiguous – the resulting partial parse trees can be tied together and a parse tree of w with respect to G is obtained. If the grammar is ambiguous all partial trees have to be produced before the correct combinations can be made. Similarly, we can start with a bottom-up parser at the left end of the string and combine it with a top-down parser starting from the right end of the string. Especially when the grammar G allows a combination of a deterministic top-down (or LL-) parser and a deterministic bottom-up (or LR-) parser this might be a useful idea. However, in general we can not expect that if G is an LL-grammar, then G^R is an LR-grammar and conversely.

Rather than having one or two parsers operating at the far left or the far right of the input, we would like to see a number of parsers, where the number depends on the 'parallelism' the input string allows, working along the length of the input string. If there is a natural way to segment a string, then each segment can have its own parser. Examples of this strategy are the methods described in Lincoln[1970], Mickunas and Schell[1975], Fischer[1975], Carlisle and Friesen[1985] and Lozinskii and Nirenburg[1986]. Here we confine ourselves to an explanation of Fischer's method. Fischer introduces 'synchronous parsing machines' (SPM) that LR-parse part of the input string. Each of the SPM's is a serial LR-parser which is able to parse any sentence of the grammar in the usual way from left to right. However, at least in theory, Fischer's method allows any symbol in the input string as the starting point of each SPM. For practical applications one may think of starting at keywords denoting the start of a procedure, a block, or even a statement. One obvious problem that emerges is, when we let a serial LR-parser start somewhere in the input string, in what state should it start? The solution is to let each SPM carry a set of states, guaranteed to include the correct one. In addition, for each of these states the SPM carries a pushdown stack on which the next actions are to be performed. An outline of the parsing algorithm follows.

For convenience we assume that the LR-parser is an LR(0) parser. No look-ahead is necessary to decide a shift or a reduce action. In the algorithm M denotes the LR-parsing table and for any state s, $R(s)$ denotes the set consisting of the rule which has to be used in making a reduction in state s. By definition, $R(s) = \{0\}$ if no reduction has to be made in state s.

(1) *Initialization.*
Start one SPM at the far left of the input string. This SPM has a single stack and it only contains s_0, the initial state. Start a number of other SPM's. Suppose we want to start an SPM immediately to the left of some symbol a. In the LR-parse table M we can find which states have a non-empty entry for symbol a. For each of these states the SPM which will be started, possesses a stack containing this state only. Hence, the SPM is started with just those states that can validly scan the next symbol in the string.

(2) *Scan the next symbol.*
Let a be the symbol to be scanned. For each stack of the SPM, if state s is on top, then
(a) if $M(s, a) = \text{sh } s'$, then push s' on the stack;
(b) if $M(s, a) = \varnothing$, then delete this stack from the set of stacks this SPM carries.
In the latter case the stack has been shown to be invalid. While scanning the next input symbols the number of stacks that an SPM carries will decrease.

(3) *Reduce?*
Let $Q = \{s_1, \cdots, s_n\}$ be the set of top states of the stacks of the SPM under consideration. Define

$$R(Q) = \bigcup_{s \in Q} R(s).$$

(a) if $R(Q) = \{0\}$, then go to step (2); in this case the top states of the stacks agree that no reduction is indicated;
(b) if $R(Q) = \{i\}$, $i \neq 0$, and $i = A \rightarrow \gamma_i$, then, if the stacks of the SPM are deep enough to pop off $|\gamma_i|$ states and not be empty, then do reduction i;
(c) otherwise, if we have insufficient stack depth or not all top states agree on the same reduction, we stop this SPM (for the time being) and, if possible, we start a new SPM to the immediate right.

An SPM which has been stopped can be restarted. If an SPM is about to scan a symbol already scanned by an SPM to its immediate right, then a merge of the two SPM's will be attempted. The following two situations have to be distinguished:

- If the left SPM contains a single stack with top state s, then s is the correct state to be in and we can select from the stacks of the right SPM the stack with bottom state s. Pop s from the left stack and then concatenate the two. All other stacks can be discarded and the newly obtained SPM can continue parsing.
- If the left SPM contains more than one stack, then it is stopped. It has to wait until it is restarted by an SPM to its left. Notice that the leftmost SPM always has one stack and it will always have sufficient stack depth. Therefore there will always be an SPM coming from the left which can restart a waiting SPM.

In step (3c) we started a new SPM immediate to the right of the stopped SPM. What set of states and associated stacks should it be started in? We cannot, as was done in the initialization, simply take those states which allow a scan of the next input symbol. To the left of this new SPM reductions may have been done (or will be done) and therefore other states should be considered in order to guarantee that the correct state is included. Hence, if in step (3) $|R(Q)| > 1$, then for each s in Q, provided $R(s) = \{0\}$, we add s to the set of states of the new SPM and in case $R(s) = \{i\}$ we add to the set of states that have to be carried by the new SPM also the states that can become topmost after a reduction using production rule i (perhaps followed by other reductions).

This concludes our explanation of Fischer's method. For more details and extensions of these ideas the reader is referred to Fischer[1975].

'Solving' Parsing Conflicts by Parallelism?

To allow more efficient parsing methods restrictions on the class of general context-free grammars have been introduced. These restrictions have led to, among others, the classes of LL-, LR- and precedence grammars and associated LL-, LR- and precedence parsing techniques. The LR-technique uses, as discussed in the previous section, an LR-parsing table which is constructed from the LR-grammar.

If the grammar from which the table is constructed is not an LR-grammar, then the table will contain conflict entries. In case of a conflict entry the parser has to choose. One decision may turn out to be wrong or both (or more) possibilities may be correct but only one may be chosen. The entry may allow reduction of a production rule but at the same time it may allow shifting of the next input symbol onto the stack. A conflict entry may also allow reductions according to different production rules. Consider the following example grammar G:

1. S → NP VP
2. S → S PP
3. NP → *n
4. NP → *det *n
5. NP → NP PP
6. PP → *prep NP
7. VP → *v NP

The parsing table for this grammar, taken from Tomita[1985], is shown in Fig. 3.

state	*det	*n	*v	*prep	$	NP	PP	VP	S
0	sh3	sh4				2			1
1				sh6	acc		5		
2			sh7	sh6			9	8	
3		sh10							
4			re3	re3	re3				
5				re2	re2				
6	sh3	sh4				11			
7	sh3	sh4				12			
8				re1	re1				
9			re5	re5	re5				
10			re4	re4	re4				
11			re6	re6, sh6	re6		9		
12				re7, sh6	re7		9		

Fig. 3 LR-parsing table for grammar G.

Tomita's answer to the problem of LR-parsing of general context-free grammars is 'pseudo-parallelism'. Each time during parsing the parser encounters a multiple entry, the parsing process is split into as many processes as there are entries. Splitting is done by replicating the stack as many times as necessary and then continue parsing with the actions of the entry separately. The processes are 'synchronized' on the shift action. Any process that encounters a shift action waits until the other processes also encounter a shift action. Therefore all processes look at the same input word of the sentence.

Obviously, this LR-directed breadth-first parsing may lead to a large number of non-interacting stacks. So it may occur that during parts of a sentence all processes behave in exactly the same way. Both the amount of computation and the amount of space can be reduced considerably by unifying processes by combining their stacks into a so-called 'graph-structured' stack. Tomita does not suggest a parallel implementation of the algorithm. Rather his techniques for improving efficiency are aimed at efficient serial processing of sentences. Nevertheless, we can ask whether a parallel implementation might be useful. Obviously, Tomita's method is not a 'parallel-designed' algorithm. There is a master routine (the LR-parser) which maintains a data structure (the graph-structured stack) and each word that is read by the LR-parser is required for each process (or stack). In a parallel implementation nothing is gained when we weave a list of stacks into a graph-structured stack. In tact, when this is done, Tomita's method becomes closely related to Earley's method (see section 4) and it seems more natural – although the number of processes may become too large – to consider parallel versions of this algorithm since it is not restricted in advance by the use of a stack. When we want to stay close to Tomita's ideas, then we rather think of a more straightforward parallel implementation in which each LR conflict causes the creation of a new LR-parser which receives a copy of the stack and a copy of the remaining input (if it is already available) and then continues parsing without ever communicating with the other LR-parsers that work on the same string. On a transputer network, for example, each transputer may act as an LR-parser. However, due to its restrictions on interconnection patterns, sending stacks and strings through the network may become a time-consuming process. When a parser encounters a conflict the network should be searched for a free transputer whereas the stack and the remainder of the input should be passed through the network to this transputer. This will cause other processes to slow down and one may expect that only a limited 'degree of non-LR-ness' will allow an appropriate application of these ideas. Moreover, one may expect serious problems when on-line parsing of the input is required.

3. Translating Grammar Rules into Process Configurations

A simple 'object-oriented' parallel parsing method for ε-free and cycle-free context-free grammars has been introduced by Yonezawa and Ohsawa[1988]. The method resembles the well-known Cocke-Younger-Kasami parsing method, but does not require that the grammars are in Chomsky Normal Form (CNF). Consider again our example grammar G:

1. S → NP VP	5. NP → NP PP
2. S → S PP	6. PP → *prep NP
3. NP → *n	7. VP → *v NP
4. NP → *det *n	

The parsing table for this grammar, taken from Tomita[1985], is shown in This set of rules will be viewed as a network of computing agents working concurrently. Each occurrence of a (pre-)terminal or a nonterminal symbol in the grammar rules corresponds with an agent with modest processing power and internal memory. The agents communicate with one another by passing subtrees of possible parse trees. The topology of the network is obtained as follows. Rule 1 yields the network fragment depicted in Fig. 4.

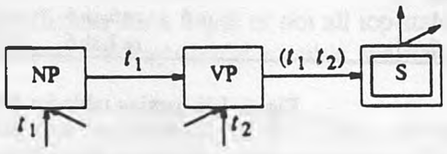


Fig. 4 From rules to configuration.

In the figure we have three agents, one for NP, one for VP and a 'double' agent for S. Suppose the NP-agent has received a subtree t_1. It passes t_1 to the VP-agent. Suppose this agent has received a subtree t_2. It checks whether they can be put together (the 'boundary adjacency test') and, if this test succeeds, it passes $(t_1\ t_2)$ to the S-agent. This agent constructs the parse tree $(S\ (t_1\ t_2))$ and distributes the result to all computing agents in the network which correspond with an occurrence of S in a right hand side of a rule. The complete network for the rules of G is shown in Fig. 5. As can be seen in the network, there is only one of these S-agents. For this agent $(S\ (t_1\ t_2))$ plays the same role as t_1 did for the NP-agent. If the boundary adjacency test is not successful, then the VP-agent stores the trees until it has a pair of trees which satisfies the test.

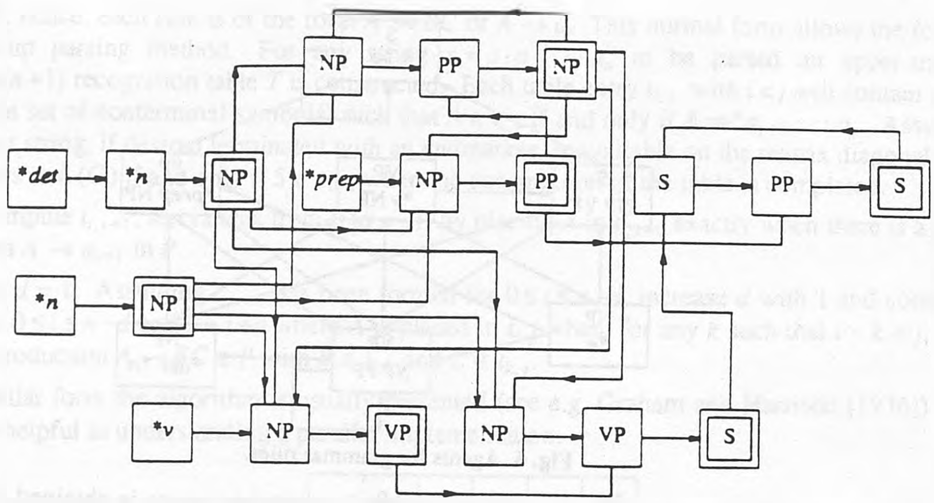


Fig. 5 Computing agents for grammar *G*.

As an example, consider the sentence *The man saw a girl with a telescope*. For this particular sentence we do not want to construct from a subtree t_1 for *a telescope* and from a subtree t_2 for *saw the girl* a subtree for *a telescope saw a girl*, although the rule $S \rightarrow NP\ VP$ permits this construction. Therefore, words to be sent into the network are provided with tags representing positional information and during construction of a subtree this information is inherited from its constituents. For our example sentence the input should look as

(0 1 the)(1 2 man)(2 3 saw)(3 4 a)(4 5 girl)(5 6 with)(6 7 a)(7 8 telescope).

Combination of tokens and trees according to the grammar rules and the positional information can yield a subtree *(3 5 (NP ((*det a)(*n girl))))* but not a subtree in which *(0 1 the)* and *(4 5 girl)* are combined. Each word accompanied with its tags is distributed to the agents for its (pre-)terminal(s) by a *manager agent* which has this information available.

If the context-free grammar which underlies the network is ambiguous, then all possible parse trees for a given input sentence will be constructed. It is possible to pipe-line constructed subtrees to semantic processing agents which filter the trees so that only semantically valid subtrees are distributed to other agents. Another useful extension is the capability to unparse a sentence when the user of a system based on this method erases ('backspaces to') previously typed words. This can be realized by letting the agents send anti-messages that cancel the effects of earlier messages. It should be noted that the parsing of a sentence does not have to be finished before a next sentence is fed into the network. By attaching another tag to the words it becomes possible to distinguish the subtrees from one sentence from those of an other sentence. The method as explained here has been implemented in the object-oriented concurrent language ABCL/1. For the experiment a context-free English grammar which gave rise to 1124 computing agents has been used. Sentences with a length between 10 and 30 words and a parse tree height between .0 and 20 were used for input. Parallelism was simulated by time-slicing. From this simulation it followed that a parse tree is produced from the network in $O(n \times h)$ time, where n is the length of the input string and h is the height of the parse tree. Obviously, simple examples of grammars and their sentences can be given which cause an explosion in the number of adjacency tests and also in the number of subtrees that will be stored without ever being used. Constructs which lead to such explosions do not usually occur in context-free descriptions of natural language.

There are several ways in which the number of computing agents can be reduced. For example, instead of the three double *NP*-agents of Fig. 5 it is possible to use one double *NP*-agent with the same function but with an increase of parse trees that have to be constructed and distributed. The same can be done for the two *S*-agents. A next step is to eliminate all double agents and give their tasks to the agents which correspond with the rightmost symbol of a grammar rule. It is also possible to have one computing agent for each grammar rule. In this way we obtain the configuration of Fig. 6. It will be clear what has to be done by the different agents.

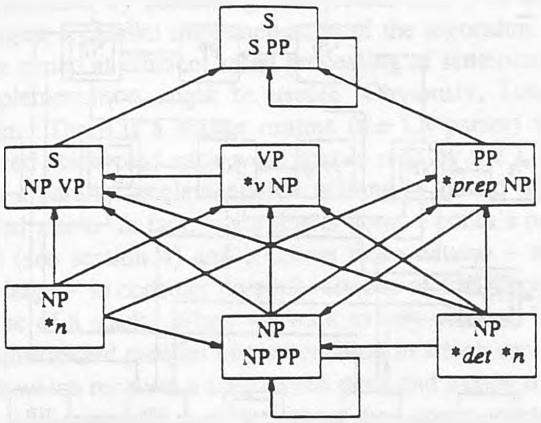


Fig. 6 Agents for grammar rules.

Another configuration with a reduced number of computing agents is obtained if we have an agent for each nonterminal symbol of the grammar. For our example grammar we have four agents, the *S*-, the *NP*-, the *VP*-, and the *PP*-agent. We may also introduce agents for the pre-terminals or even for each word which can occur in an input sentence. We confine ourselves to agents for the nonterminal symbols and discuss their roles. In Fig. 7 we have displayed the configuration of computing agents which will be obtained from the example grammar.

The communication between the agents of this network is as follows.

(1) The *S*-agent sends subtrees with root *S* to itself; it receives subtrees from itself, the *PP*-agent, the *NP*-agent, and the *VP*-agent.

(2) The *NP*-agent sends subtrees with root *NP* to itself, the *S*-agent, the *VP*-agent and the *PP*-agent; it receives subtrees from itself and from the *PP*-agent; moreover, input comes from the manager agent.

(3) The *VP*-agent sends subtrees with root *VP* to the *S*-agent; it receives subtrees from the *NP*-agent; moreover, input comes from the manager agent.

(4) The *PP*-agent sends subtrees with root *PP* to the *S*-agent and to the *NP*-agent; it receives subtrees from the *NP*-agent; moreover, it receives input from the manager agent.

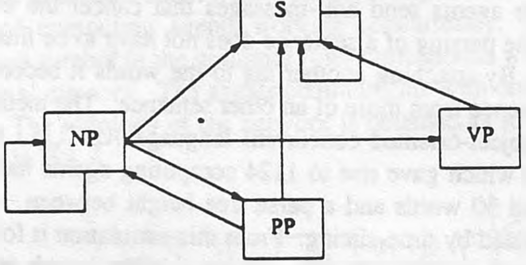


Fig. 7 Agents for nonterminal symbols.

The task of each of these nonterminal agents is to check whether the subtrees it receives can be put together according to the grammar rules with the nonterminal as left-hand side and according to positional information that is carried along with the subtrees. If possible, a tree with the nonterminal as root is constructed, otherwise the agent checks other trees or waits until trees are available.

4. From Sentence Words to Processes

Cocke-Younger-Kasami's Algorithm

Traditional parsing methods for context-free grammars have been re-investigated in order to see whether they can be adapted to a parallel processing view. In Chu and Fu[1982] parallel aspects of the tabular Cocke-Younger-Kasami algorithm have been discussed. The input grammar should be in CNF, hence, each rule is of the form $A \rightarrow BC$ or $A \rightarrow a$. This normal form allows the following bottom-up parsing method. For any string $x = a_1 a_2 \cdots a_n$ to be parsed an upper-triangular $(n+1)\times(n+1)$ recognition table T is constructed. Each table entry $t_{i,j}$ with $i<j$ will contain a subset of N (the set of nonterminal symbols) such that $A \in t_{i,j}$ if and only if $A \Rightarrow^* a_{i+1} \cdots a_j$. Assume that the input string, if desired terminated with an endmarker, is available on the matrix diagonal. String x belongs to $L(G)$ if and only if $S \in t_{0,n}$ when the construction of the table is completed.

(1) Compute $t_{i,i+1}$, as i ranges from 0 to $n-1$, by placing A in $t_{i,i+1}$ exactly when there is a production $A \rightarrow a_{i+1}$ in P.

(2) Set $d = 1$. Assuming $t_{i,i+d}$ has been formed for $0 \le i \le n-d$, increase d with 1 and compute $t_{i,j}$ for $0 \le i \le n-d$ and $j = i+d$ where A is placed in $t_{i,j}$ when, for any k such that $i<k<j$, there is a production $A \rightarrow BC \in P$ with $B \in t_{i,k}$ and $C \in t_{k,j}$.

In a similar form the algorithm is usually presented (see e.g. Graham and Harrison [1976]). Fig. 8 may be helpful in understanding a parallel implementation.

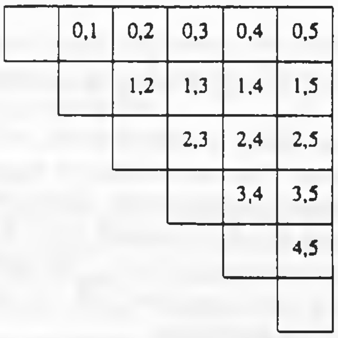


Fig. 8 The upper-triangular CYK-table.

Notice that after step (1) the computation of the entries is done diagonal by diagonal until entry $t_{0,n}$ has been completed. For each entry of a diagonal only elements of preceding diagonals are used to compute its value. More specifically, in order to see whether a nonterminal should be included in an element $t_{i,j}$ it is necessary to compare $t_{i,k}$ and $t_{k,j}$, with k between i and j. The amount of storage that is required by this method is proportional to n^2 and the number of elementary operations is proportional to n^3. Unlike Yonezawa and Oshawa's algorithm where positional information needs an explicit representation, here it is in fact available (due to the CNF of the grammar) in the indices of the table elements. For example, in $t_{1,4}$ we can find the nonterminals which generate the substring of the input between positions 1 and 4. The algorithm can be extended in order to produce parse trees.

From the recognition table we can conclude a two-dimensional configuration of processes. For each entry $t_{i,j}$ of the upper-triangular table there is a process $P_{i,j}$ which receives table elements (i.e., sets of nonterminals) from processes $P_{i,j-1}$ and $P_{i+1,j}$. Process $P_{i,j}$ transmits the table elements it receives from $P_{i,j-1}$ to $P_{i,j+1}$ and the elements it receives from $P_{i+1,j}$ to $P_{i-1,j}$. Process $P_{i,j}$ transmits the table element it has constructed to processes $P_{i-1,j}$ and $P_{i,j+1}$. Fig. 9 shows the interconnection structure for $n = 5$. As soon as a table element has been computed, it is sent to its right and upstairs neighbor. Each process should be provided with a coding of the production rules of the grammar. Clearly, each process requires $O(n)$ time. It is not difficult to see that like similar algorithms suitable for VLSI-implementation, e.g. systolic algorithms for matrix multiplication or transitive closure computation (see Guibas et al[1979] and many others) the required parsing time is also $O(n)$. In Chu and Fu[1982] a VLSI design for this algorithm is presented (see also Tan[1983]).

Earley's Algorithm

The second algorithm we discuss in this section is the well-known Earley's method. It is not essentially different from the CYK algorithm. Since the method maintains information in the table entries about the righthand sides of the productions that are being recognized, the condition that the grammar should be in CNF is not necessary. For general context-free grammars Earley parsing takes $O(n^3)$ time. This time can be reduced to $O(n^2)$ or $O(n)$ for special subclasses of context-free

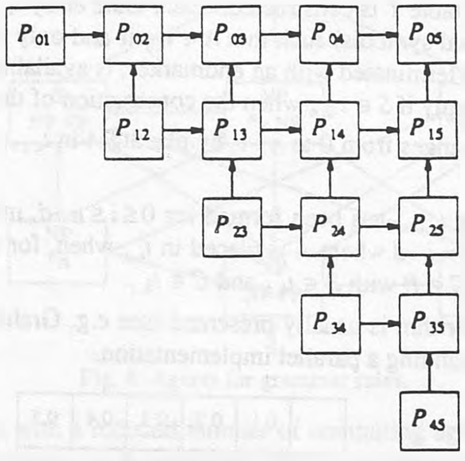


Fig. 9 Process configuration for CYK's algorithm.

grammars. Many versions of Earley's method exist. In Graham and Harrison[1976] the following tabular version can be found. For any string $x = a_1 a_2 \cdots a_n$ to be parsed an upper-triangular $(n+1)\times(n+1)$ recognition table T is constructed. Each table entry $t_{i,j}$ will contain a set of items, i.e., a set of elements of the form $A \rightarrow \alpha\cdot\beta$ (a dotted rule), where $A \rightarrow \alpha\beta$ is a production rule from the grammar and the dot $\cdot$ is a symbol not in $N \cup \Sigma$. The computation of the table entries goes column by column. The following two functions will be useful. Function PREDICT:$N \rightarrow 2^D$, where $D = \{A \rightarrow \alpha\cdot\beta \mid A \rightarrow \alpha\beta \in P\}$, is defined as

$$\text{PREDICT}(A) = \{B \rightarrow \alpha\cdot\beta \mid B \rightarrow \alpha\beta \in P,\ \alpha \Rightarrow^* \varepsilon \text{ and } \exists\ \gamma \in V^* \text{ with } A \Rightarrow^* B\gamma\}.$$

Function PRED:$2^N \rightarrow 2^D$ is defined as

$$\text{PRED}(X) = \bigcup_{A \in X} \text{PREDICT}(A).$$

Initially, $t_{0,0} = \text{PRED}(\{S\})$ and all other table entries are empty. Suppose we want to compute the elements of column j, $j > 0$. In order to compute $t_{i,j}$ with $i \neq j$ assume that all elements of the columns of the upper-triangular table to the left of column j have already been computed and in column j the elements $t_{k,j}$ for $i < k < j$ have been computed.

(1) Add $B \rightarrow \alpha a \beta\cdot\gamma$ to $t_{i,j}$ if $B \rightarrow \alpha\cdot a\beta\gamma \in t_{i,j-1}$, $a = a_j$ and $\beta \Rightarrow^* \varepsilon$.

(2) Add $B \rightarrow \alpha A\beta\cdot\gamma$ to $t_{i,j}$, if, for any k such that $i < k < j$, $B \rightarrow \alpha\cdot A\beta\gamma \in t_{i,k}$, $A \rightarrow \omega\cdot \in t_{k,j}$ and $\beta \Rightarrow^* \varepsilon$.

(3) Add $B \rightarrow \alpha A\beta\cdot\gamma$ to $t_{i,j}$ if $B \rightarrow \alpha\cdot A\beta\gamma \in t_{i,i}$, $\beta \Rightarrow^* \varepsilon$ and there exists $C \in N$ such that $A \Rightarrow^* C$ and $C \rightarrow \omega\cdot \in t_{i,j}$.

After all elements $t_{i,j}$ with $0 \le i \le j-1$ of column j have been computed then it is possible to compute $t_{j,j}$.

(4) Let $X_j = \{A \in N \mid B \rightarrow \alpha\cdot A\beta \in t_{i,j}, 0 \le i \le j-1\}$. Then $t_{j,j} = \text{PRED}(X_j)$.

It is not difficult to see that $A \rightarrow \alpha\cdot\beta \in t_{i,j}$ if and only if there exists $\gamma \in V^*$ such that $S \Rightarrow^* a_1 \cdots a_i A\gamma$ and $\alpha \Rightarrow^* a_{i+1} \cdots a_j$. Hence, in $t_{0,n}$ we can read whether the sentence was correct. The algorithm can be extended in order to produce parse trees.†

Various parallel implementations of Earley's algorithm have been suggested in the literature (see e.g. Chiang and Fu[1982], Tan[1983] and Sijstermans[1986]). The algorithms differ mainly in details on the handling of ε-rules, preprocessing, the representation of data and circuit and layout design. The main problem in a parallel implementation of the previous algorithm is the computation of the diagonal elements $t_{i,i}$, for $0 \le i \le n$. The solution is simple. Initially all elements $t_{i,i}$, $0 \le i \le n$,

† When Earley's algorithm was introduced, it was compared with the exponential time methods in which successively every path was followed whenever a non-deterministic choice occurred. Since in Earley's algorithm a 'simultaneous' following of paths can be recognized, it was sometimes considered as a parallel implementation of the earlier depth-first algorithms (see e.g. Lang[1971]).

are set equal to PREDICT(N), where N is the set of nonterminal symbols. The other entries are defined according to the steps (1), (2) and (3). As a consequence, we now have $A \rightarrow \alpha \cdot \beta \in t_{i,j}$ if and only if $\alpha \Rightarrow^* a_{i+1} \cdots a_j$. In spite of weakening the conditions on the contents of the table entries the completed table can still be used to determine whether an input sentence was correct. Moreover, computation of the elements can be done diagonal by diagonal, similar to the CYK algorithm.

(1) Set $t_{i,i}$ equal to PREDICT(N), $0 \le i \le n$.

(2) Set $d = 0$. Assuming $t_{i,i+d}$ has been formed for $0 \le i \le n-d$, increase d with 1 and compute $t_{i,j}$ for $0 \le i \le n-d$ and $j = i+d$ according to:

(2.1) Add $B \rightarrow \alpha a \beta \cdot \gamma$ to $t_{i,j}$ if $B \rightarrow \alpha \cdot a \beta \gamma \in t_{i,j-1}$, $a = a_j$ and $\beta \Rightarrow^* \varepsilon$.

(2.2) Add $B \rightarrow \alpha A \beta \cdot \gamma$ to $t_{i,j}$ if, for any k such that $i < k < j$, $B \rightarrow \alpha \cdot A \beta \gamma \in t_{i,k}$, $A \rightarrow \omega \cdot \in t_{k,j}$ and $\beta \Rightarrow^* \varepsilon$.

(2.3) Add $B \rightarrow \alpha A \beta \cdot \gamma$ to $t_{i,j}$ if $B \rightarrow \alpha \cdot A \beta \gamma \in t_{i,i}$, $\beta \Rightarrow^* \varepsilon$ and there exists $C \in N$ such that $A \Rightarrow^* C$ and $C \rightarrow \omega \cdot \in t_{i,j}$.

VLSI designs or process configurations which implement this algorithm in such a way that it takes $O(n)$ time (with $O(n^2)$ cells or processes can be found in Chiang and Fu[1982], Tan[1983] and Sijstermans[1986] (see also Fig. 9 and its explanation).

5. Connectionist Parsing Algorithms

Only few authors have considered parsing in connectionist networks. It is possible to distinguish a dynamic programming approach based on the CYK algorithm (Fanty[1985]), a Boltzmann machine approach (Selman and Hirst[1985,1987]) and an interactive relaxation approach (Howells[1980]). We confine ourselves to an explanation of Fanty's method since it fits rather naturally in the framework of parsing strategies we have considered in the previous sections. A connectionist Earley parsing algorithm can be found in the full version of the present paper.

Fanty's strategy is that of the CYK parser. The nodes that will be part of the connectionist network are organized according to the positions of the entries of the upper-triangular recognition table. For convenience we first assume that the grammar is in CNF. The table's diagonal will be used for representing the input symbols. This representation will be explained later. For each nonterminal symbol each entry in the table which is not on the diagonal will represent a configuration of nodes. These nodes allow top-down and bottom-up passing of activity. We first explain the bottom-up pass. Consider a particular entry, say $t_{i,j}$ with $j-i \ge 2$, of the upper-triangular matrix. In the traditional algorithm a nonterminal symbol X is added to the set of nonterminal symbols associated with the entry if there are symbols $Y \in t_{i,k}$ and $Z \in t_{k,j}$ such that $X \rightarrow YZ$ is in P. In the connectionist adaptation of the algorithm we already have a node for each nonterminal symbol in entry $t_{i,j}$. Therefore, rather than adding a symbol, here node X at position $t_{i,j}$ is made active if node Y at position $t_{i,k}$ and node Z at position $t_{k,j}$ are active. In general there will be more ways to have a realization of the production $X \rightarrow YZ$ at position $t_{i,j}$. For example, a node for X at entry $t_{1,5}$ can be made active for a production $X \rightarrow YZ$ if there is an active node for Y at $t_{1,2}$ and for Z at $t_{2,5}$, or for Y at $t_{1,3}$ and for Z at $t_{3,5}$, or for Y at $t_{1,4}$ and for Z at $t_{4,5}$. This separation is realized with the help of match nodes in the configuration of each entry of the table. The use of match nodes is illustrated in Fig. 10 for a node for X at position t_{15} of a CYK-table. Here we have shown the three match nodes, one for each possible realization of $X \rightarrow YZ$, for this node at this particular position. For these match nodes to become active all of their inputs must be on. The node for X becomes active when at least one of its inputs (coming from its match nodes) is on. In the figure only match nodes for separate realizations of the same production are included. Obviously, match nodes should also be included at this position for all possible realizations of the other productions with lefthand side X. In this way all the inputs that can make the node for X at this particular position active can be received in a proper way. Observe that if during the recognition of a sentence in an entry more than one match node for a nonterminal is active then the sentence is ambiguous.

In our explanation the assumption $j-i \ge 2$ for entry $t_{i,j}$ was made. Since the grammar is in CNF we have realizations of productions of the form $X \rightarrow a$ in the entries $t_{i,j}$ with $j-i = 1$. In these entries no match nodes are needed since in each entry there can be only one realization of a production with a given lefthand side. We assume that there is a node for each terminal symbol in each

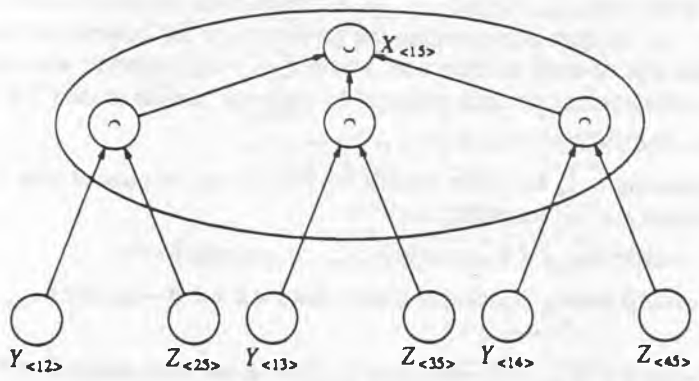


Fig. 10 Bottom-up passing of activity.

position at the diagonal of the matrix. Parsing starts by activating the nodes which correspond with the input symbols. Then activation passes bottom-up through the network, first with realizations of productions of the form $X \rightarrow a$, next with realizations of productions of the form $X \rightarrow YZ$. The input is accepted as soon as the node for the start symbol in the topmost entry of the column of the last input symbol becomes active.

Until now we have discussed a network which accepts (or rejects) an input string. In order to obtain a representation of the parse tree or parse trees a second, top-down, pass of activity is necessary. To perform this top-down pass we assume that each node mentioned so far consists of a bottom-up and a top-down unit. The bottom-up units are used as explained above. In Fig. 11 both bottom-up and top-down passing of activity is illustrated in a configuration of nodes for an entry $t_{i,j}$ with $j-i \geq 2$. Each node is represented as consisting of a leftmost or bottom-up and a rightmost or top-down unit.

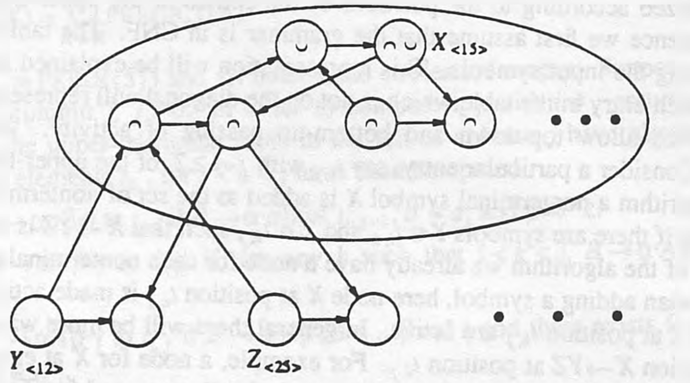


Fig. 11 Top-down and bottom-up passing of activity.

A top-down unit becomes active when it receives input from its bottom-up counterpart and at least one external source. In order to activate the top-down unit of the node for the start symbol in the upper right corner of the table we assume that it receives input from its bottom-up counterpart and from the node at position $t_{n,n}$, where n is the length of the input, which is used to represent end-marker $\$$ of the input and which is made active when parsing starts. Hence, when the input is recognized this unit becomes active and it passes activity top-down. All top-down units which receive this activation and which receive activation from their bottom-up counterparts become active. In this way activity is passed down to the terminal nodes and the active top-down nodes of the network represent the parse tree(s). The parse in the connectionist network completes in $O(n)$ time.

Above our assumption was that grammars are in CNF. This is not a necessary condition, but it facilitates the present discussion. See Fanty[1985] or the full version of this paper for possible relaxations of this condition and the consequences for the time complexity.

6. Conclusions

A survey of some ideas in parallel parsing has been presented. In the field of natural language processing the Earley and CYK method are well known. Sometimes closely related methods such as (active) chart parsing are used. Because of this close relationship a parallel implementation along the lines sketched above is possible. Chart parsing (and Earley parsing) can be done with a more modest number of processors if an agenda approach is followed (see e.g. Grisham and Chitrao[1988]). Earley's algorithm can be modified to transition networks and extended to ATN's (see e.g. Chou and Fu[1975]). Therefore it is worthwhile to investigate a similar parallel approach to the parsing of ATN's. No attention has been paid to ideas aimed at improving upper bounds for the recognition and parsing of general context-free languages. An introduction to that area can be found in Chapter 4 of Gibbons and Rytter[1988]. Neither have we been looking here at the connectionist approaches in parsing and natural language processing as they are discussed in the papers of Cottrell and Small[1984], Waltz and Pollack[1985], McClelland and Kawamoto[1986] and Small[1987]. More references to papers on parallel parsing can be found in Nijholt et al[1989].

Acknowledgements.
I am grateful to P.R.J. Asveld for his comments on an earlier version of this paper and to Theo Vosse for drawing my attention to some papers on connectionist parsing. Some discussions with Bart Van Acker and Bart De Wolf have improved my understanding of parallel parsing methods.

7. References

Carlisle, W.H. and D.K. Friesen [1985]. Parallel parsing using Ada. Proceedings 3rd Annual National *Conference on Ada Technology*, March 1985, 103–106.

Chiang, Y.T. and K.S. Fu [1982]. A VLSI architecture for fast context-free language recognition (Earley's algorithm). Proceedings Third International *Conf. on Distributed Comp. Systems*, 1982, 864–869.

Chou, S.M. and K.S. Fu [1975]. Transition networks for pattern recognition. School for Electrical Engineering, Purdue University, West Lafayette, Indiana, TR-EE 75–39, 1975.

Chu, K.-H. and K.S. Fu [1982]. VLSI architectures for high-speed recognition of context-free languages and finite-state languages. Proceedings of the Ninth Annual *Symposium on Computer Architectures, SIGARCH Newsletter* 10 (1982), No.3, 43–49.

Cottrell, G.W. and S.L. Small [1984]. Viewing parsing as word sense discrimination: A connectionist approach. In: *Computational Models of Natural Language Processing*, B.G. Bara and G. Guida (eds.), Elsevier Science Publishers, North-Holland, 1984, 91–119.

Fischer, C.N. [1975]. Parsing context-free languages in parallel environments. Ph.D. Thesis, Tech. Report 75-237, Dept. of Computer Science, Cornell University, 1975.

Gibbons, A. and W. Rytter [1988]. Parallel recognition and parsing of context-free languages. Chapter 4 in *Efficient Parallel Algorithms*. Cambridge University Press, Cambridge, 1988.

Graham, S.L. and M.A. Harrison [1976]. Parsing of general context-free languages. *Advances in Computers*, Vol. 14, M. Yovits and M. Rubinoff (eds.), Academic Press, New York, 1976, 76–185.

Grisham, R. and M. Chitrao [1988]. Evaluation of a parallel chart parser. In: Proceedings of the *Second Conference on Applied Natural Language Processing*, Association for Computational Linguistics, 1988, 71–76.

Guibas, L.J., H.T. Kung and C.D. Thompson [1979]. Direct VLSI implementation of combinatorial algorithms. *Proc. Conf. on VLSI*, Caltech, January 1979, 509–526.

Lang, B. [1971]. Parallel non-deterministic bottom-up parsing. In: Proc. *Int. Symposium on Extensible Languages*. Grenoble, 1971, *SIGPLAN Notices* 6, Nr. 12, December 1971.

Lincoln, N. [1970]. Parallel programming techniques for compilers. *SIGPLAN Notices* 5 (1970), No.10, 18–31.

Loka, R.R. [1984]. A note on parallel parsing. *SIGPLAN Notices* 19 (1984), No.1, 57–59.

Lozinskii, E.L. and S. Nirenburg [1986]. Parsing in parallel. *Computer Languages* 11 (1986), 39–51.

McClelland J.L. and A.H. Kawamoto [1986]. Mechanism of sentence processing: assigning roles to constituents of sentences. Chapter 19 in *Parallel Distributed Processing*. Vol.2: *Psychological and Biological Models*, D.E. Rumelhart, J.L. McClelland and the PDP Research Group, The MIT Press, Cambridge, Mass., 1986, 272-325.

Mickunas, M.D. and R.M. Schell [1978]. Parallel compilation in a multiprocessor environment. Proceedings *ACM Annual Conf.*, 1978, 241–246.

Nijholt, A. [1988]. *Computers and Languages: Theory and Practice*. Studies in Computer Science and Artificial Intelligence. North-Holland, Elsevier Science Publishers, Amsterdam, 1988.

Nijholt, A. et al [1989]. An annotated bibliography on parallel parsing. Twente University, Internal Memorandum, in preparation, 1989.

Selman, B. and G. Hirst [1987]. Parsing as an energy minimization problem. Chapter 11 in *Genetic Algorithms and Simulated Annealing*. Research Notes in A.I., Morgan Kaufmann Publishers, Los Altos, California, 1987.

Small, S.L. [1987]. A distributed word-based approach to parsing. In: *Natural Language Parsing Systems*. L. Bolc (ed.), Springer-Verlag, Berlin, 1987, 161–201.

Srikant, Y.N. and P. Shankar [1987]. Parallel parsing of programming languages. *Information Sciences* 43 (1987), 55–83.

Sijstermans, F.W. [1986]. Parallel parsing of context-free languages. Doc. No. 202, Esprit Project 415, Subproject A: Object-oriented language approach, Philips Research Laboratories, Eindhoven, 1986.

Tan, H.D.A. [1983]. VLSI-algoritmen voor herkenning van context-vrije talen in lineaire tijd. Rapport IN 24/83, Stichting Mathematisch Centrum, Amsterdam, Juni 1983.

Tomita, M. [1985]. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, Boston, Dordrecht, 1985.

Tseytlin, G.E. and E.L. Yushchenko [1977]. Several aspects of theory of parametric models of languages and parallel syntactic analysis. In: *Methods of Algorithmic Language Implementation*. A. Ershov and C.H.A. Koster (eds.), Lect. Notes Comp. Sci. 47, Springer-Verlag, Berlin, 1977, 231–245.

Waltz, D.L. and J.B. Pollack [1985]. Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science* 9 (1985), 51-74.

Yonezawa, A. and I. Ohsawa [1988]. Object-oriented parallel parsing for context-free grammars. In: *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88)*, Budapest, 1988, 773–778.

COMPLEXITY AND DECIDABILITY IN LEFT-ASSOCIATIVE GRAMMAR[1]

ROLAND HAUSSER

1. Formal Rule Schemata of Generative Grammar

Left-associative grammar (LA-grammar) is a comparatively new formalism. By way of introduction, let us compare it with more widely known systems, namely phrase structure grammar (PS-grammar) and categorial grammar (C-grammar).

The formalism of PS-grammar is based on the rewriting systems of Post (1936). Rewriting rules have the following form:

(1.1) The Schema of a Phrase-Structure Rewriting Rule

$$A \rightarrow B\ C$$

By replacing (rewriting) the symbol A with B and C, this rule generates a tree structure with A dominating B and C. Conceptually, the derivation order of rewriting rules is top-down.

The formalism of C-grammar is based on the categorial-canceling rules of Leśnieswki (1929) and Ajdukiewicz (1935). Categorial-canceling rules have the following form:

(1.2) The Schema of a Categorial Canceling Rule

$$\alpha_{(Y|X)} \bullet \beta_{(Y)} \Rightarrow \alpha\beta_{(X)}$$

This rule schema combines α and β into $\alpha\beta$ by canceling the Y in the category of α with the corresponding category of β. The result is a tree structure with $\alpha\beta$ of category X dominating α and β. Conceptually, the derivation order of categorial-canceling rules is bottom-up.

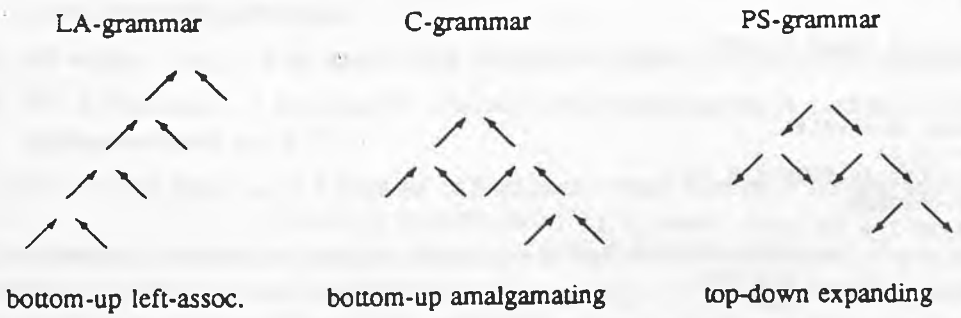
(1.3) The Schema of a Left-Associative Rule

$$r_i\text{: [CAT-1 CAT-2]} \Rightarrow [rp_i \text{ CAT-3}]$$

A left-associative rule r_i maps a sentence start (represented by its category CAT-1) and a next word (represented by its category CAT-2) into the rule package rp_i and a new sentence start (represented by its category CAT-3). A *state* in LA-grammar is defined as an ordered pair, consisting of a rule package and a sentence start. In the next composition, the rules in the rule package are applied to the sentence start resulting from the last composition and a new next word.

The different rule schemata result in three different conceptual derivation orders.

(1.4) Three Grammatical Derivation Orders



LA-grammars are input-output equivalent to their parsers and generators in that (i) they take the same input (i.e., an unanalyzed surface string), (ii) generate the same output (a left-associative syntactic analysis),

[1]The results reported in this paper are published in Hausser, R. (1989) *Computation of Language*, Springer-Verlag Berlin-New York (Symbolic Computation – *Artificial Intelligence*), June 1989.

and (iii) use the same rules in the same derivation order. In other words, LA-grammar achieves "absolute type transparency"[2] because it is strongly input-output equivalent to its parsers and generators.

PS-grammar and C-grammar, on the other hand, are unsuitable for direct parsing. Parsers for context-free PS-grammars, for example, cannot possibly apply the rules of the grammar directly because the rules rewrite an initial start symbol, while the parser takes sentences as input. The standard solution to this dilemma consists in computional routines which reconstruct the grammatical analysis in an indirect way by building large intermediate structures (e.g., "state sets", "charts", "tables") which are not part of the grammar.

2. Syntax and Semantics

The tree structures generated by PS-grammar and C-grammar are semantically motivated *constituent structures*. Constituent structures are based on *substitution* and *movement tests* which are intended to reveal which parts of the sentence belong most closely together. The completely regular tree structure of LA-grammar, on the other hand, is based on the notion of *possible continuations* and reflects the time-linear nature of language.

As an example of a left-associative parse consider (2.1).

(2.1) A Sample Derivation

```
NEWCAT> (z Fido found a bone.)
Elapsed real time = 779 milliseconds
User cpu time = 660 milliseconds
System cpu time = 20 milliseconds
Total cpu time = 680 milliseconds
   Linear Analysis:

   *START_0
   1
      (NA) FIDO
      (N SC V) FOUND
   *NOM+FVERB_3
   2
      (SC V) FIDO FOUND
      (SQ) A
   *FVERB+MAIN_4
   3
      (SQ V) FIDO FOUND A
      (SN) BONE
   *DET+NOUN_2
   4
      (V) FIDO FOUND A BONE
      (V DECL) .
   *CMPLT_13
   5
      (DECL) FIDO FOUND A BONE .


    Hierarchical Analysis:
 (PROPOSITION-5_6_13
        (MOOD (DECLARATIVE-5_6_13))
        (PROP-CONTENT
          ((SENT-2_6_13 (SUBJ ((NP-1_6_13 (NAME (FIDO-1_6_13)))))
                        (VERB (FIND-2_6_13))
                        (DIR-OBJ
                        ((NP-3_6_13 (REF (A-3_6_13 SG-4_6_13))
                                    (NOUN ((BONE-4_6_13))))))))))
```

[2] Berwick & Weinberg (1984), p. 41.

```
          PROPOSITION
               |
      -----------------
      |               |
    MOOD         PROP-CONTENT
      |               |
      |               |
 DECLARATIVE         SENT
                      |
                ---------------
                |      |      |
              SUBJ VERB   DIR-OBJ
                |      |      |
                |      |      |
                |      |      |
               NP   FIND     NP
                |             |
                |          ------
                |          |    |
              NAME        REF NOUN
                |          |    |
                |        ----   |
                |        |  |   |
              FIDO       A SG BONE
```

The algorithm of left-associative grammar (LA-grammar) always combines a sentence start and a next word into a new sentence start. In a semantically interpreted LA-grammar, a homomorphic semantic hierarchy is constructed simultaneously with the linear syntactic parse. The semantic hierarchy expresses many of the intuitions central to constituent structure and may be displayed as a structured list or, equivalently, as a tree structure. The following discussion of LA-grammar is limited to the formal properties of the linear syntax.

3. The Mathematical Definition

(3.1) Formal Definition of Left-Associative Grammar[3]

An LA-grammar is defined as a 7-tuple $<W, C, LX, CO, RP, ST_S, ST_F>$, where

1. W is a finite set of *word surfaces.*
2. C is a finite set of *category segments.*
3. $LX \subset (W \times C^+)$ is a finite set comprising the *lexicon.*
4. $CO = (co_0 \ldots co_{n-1})$ is a finite sequence of total recursive functions from $(C^* \times C^+)$ into $C^* \cup \{\perp\}$, called *categorial operations.*
5. $RP = (rp_0 \ldots rp_{n-1})$ is an equally long sequence of subsets of *n* called *rule packages.*
6. $ST_S = \{(rp_s\ cat_s), \ldots\}$ is a finite set of *initial states*, where each rp_s is a subset of *n* called a start rule package and each $cat_s \in C^+$.
7. $ST_F = \{(rp_f\ cat_f), \ldots\}$ is a finite set of *final states*, where each $rp_f \in RP$ and each $cat_f \in C^*$.

For theoretical reasons, the categorial operations are defined as total functions. In practice, the categorial operations are defined on easily-recognizable subsets of $(C^* \times C^+)$, where anything outside these subsets is mapped into the arbitrary "don't care" value $\{\perp\}$, making the categorial operations total.

[3]Let us recall some notation from set theory needed in this definition. If X is a set, then X^+ is the "positive closure," i.e., the set of all concatenations of elements of X. X^* is the Kleene closure of X, defined as $X^+ \cup \epsilon$, where ϵ is the "empty sequence." The power set of X is denoted by 2^X. If X and Y are sets, then $(X \times Y)$ is the Cartesian product of X and Y, i.e., the set of ordered pairs consisting of an element of X and an element of Y. For convenience, we also identify integers with sets, i.e., $n = \{i \mid 0 \leq i < n\}$.

An LA-grammar is specified by (i) a lexicon LX, (ii) a set of start states ST_S, (iii) a sequence of rules, each defined as an ordered pair (co_i rp_i), and (iv) a set of final states ST_F. This general format of LA-grammars is illustrated below with the context-sensitive language $a^kb^kc^k$.

(3.2) The Definition of $a^kb^kc^k$

LX $=_{def}$ {[a (bc)], [b (b)], [c (c)]}
ST_S $=_{def}$ {({r-1, r-2} (bc))}
r-1: [(X) (bc)] $\Rightarrow$ [{r-1, r-2} (bXc)],
r-2: [(bXc) (b)] $\Rightarrow$ [{r-2, r-3} (Xc)],
r-3: [(cX) (c)] $\Rightarrow$ [{r-3} (X)]
ST_F $=_{def}$ {[rp-3 ϵ]}.

LA-grammar is equally suitable for parsing and generation. The only difference is that in parsing the next word is provided by the input string, while in generation the next word is chosen from the lexicon. The grammatical analysis provided by LA-parsers and LA-generators is simply a trace of the computation. The declarative linguistic analysis and the computation are merely different aspects of the same left-associative structure.

(3.3) Parsing aaabbbccc with Active Rule Counter

```
NEWCAT> (z a a a b b b c c c)
;  1: Applying rules (RULE-1 RULE-2)
;  2: Applying rules (RULE-1 RULE-2)
;  3: Applying rules (RULE-1 RULE-2)
;  4: Applying rules (RULE-2 RULE-3)
;  5: Applying rules (RULE-2 RULE-3)
;  6: Applying rules (RULE-2 RULE-3)
;  7: Applying rules (RULE-3)
;  8: Applying rules (RULE-3)
; Number of rule applications: 14.

      *START-0
      1
         (B C) A
         (B C) A
      *RULE-1
      2
         (B B C C) A A
         (B C) A
      *RULE-1
      3
         (B B B C C C) A A A
         (B) B
      *RULE-2
      4
         (B B C C C) A A A B
         (B) B
      *RULE-2
      5
         (B C C C) A A A B B
         (B) B
      *RULE-2
      6
         (C C C) A A A B B B
         (C) C
      *RULE-3
      7
         (C C) A A A B B B C
         (C) C
      *RULE-3
      8
         (C) A A A B B B C C
```

```
        (C) C
    *RULE-3
    9
        (NIL) A A A B B B C C C
```

The parse is called with the function "(z ...)". Note that categories precede the surfaces in (3.3). Each left-associative composition is characterized by a word number (e.g., 4), a sentence start consisting of a category and a surface (e.g., (B B C C) A A A B), a next word (e.g., (B) B), and a rule (e.g., *RULE-2). The result of the composition is shown in the first line of the next "history section" (e.g., (B C C C) A A A B B).

As an illustration of the relation between an LA-grammar and its generator, consider the following derivation of well-formed expressions up to length 12 using the grammar for $a^k b^k c^k$ defined in (3.2).

(3.4) Generating the Representative Sample of $a^k b^k c^k$

```
NEWCAT> (gram-gen 3 '(a b c))

Parses of length 2:
 A B
   2    (C)
 A A
   1    (B B C C)

Parses of length 3:
 A B C
   2 3    (NIL)
 A A B
   1 2    (B C C)
 A A A
   1 1    (B B B C C C)

          ...

Parses of length 9:
 A A A B B B C C C
   1 1 2 2 2 3 3 3    (NIL)
 A A A A B B B B C
   1 1 1 2 2 2 2 3    (C C C)

Parses of length 10:
 A A A A B B B B C C
   1 1 1 2 2 2 2 3 3    (C C)

Parses of length 11:
 A A A A B B B B C C C
   1 1 1 2 2 2 2 3 3 3    (C)

Parses of length 12:
 A A A A B B B B C C C C
   1 1 1 2 2 2 2 3 3 3 3    (NIL)
```

After loading the same grammar as used for parsing, the function 'gram-gen' is called with two arguments: the "recursion factor" of the grammar (cf. Section 6), and a list of the words to be used.[4] The output is a systematic generation, starting with well-formed expressions of length 2. Each derivation consists of a surface, a sequence of rules, and a result category. As an example of a single derivation, consider (3.5).

[4]In another version, 'gram-gen' is called with the maximal surface length rather than the recursion factor.

(3.5) A Complete Well-Formed Expression in $a^kb^kc^k$

```
A A A B B B C C C
  1 1 2 2 2 3 3 3    (NIL)
```

The surface and the rule sequence are lined up so that it is apparent which word was added by which rule. Derivation (3.5) characterizes a complete well-formed expression because it represents the rule state (rp-3 ϵ), which is element of the set of complete well-formed expressions of the LA-grammar for $a^kb^kc^k$ defined in (3.2).

4. Generative Capacity and the Chomsky Hierarchy

The most basic formal result in LA-grammar is that it generates all—and only—the recursive languages. That LA-grammar generates all recursive languages follows from the fact that a categorial operation can be any total recursive function.[5] That LA-grammar generates **only** the recursive languages, on the other hand, is due to the linear structure of the derivation: at each left-associative composition there is only a finite number of sentence starts, each with a finite rule package, and a finite number of next word readings, resulting in a finite number of new sentence starts.[6]

Furthermore, we may show that the automata-theoretic hierarchy of regular, context-free, and context-sensitive languages is clearly reflected in the formalism of LA-grammar. Specifically, **regular languages** are generated by LA-Grammars with rules using only empty categorial operations, e.g.,

(4.1) LA-Rule with Empty Categorial Operation

r_i: [ϵ CAT-2] $\Rightarrow$ [rp_i ϵ]

The proof is based on a systematic translation procedure from Finite State Automata into LA-grammars with rules like (4.1).[7]

Context-free languages are generated by LA-grammars with categorial operations which work only on the first segment of CAT-1 or CAT-3, e.g.,

r_i: [(a X)(a)] $\Rightarrow$ [rp_i (X)]

or

r_i: [(X)(a)] $\Rightarrow$ [rp_i (a X)]

where X is a variable for sequences of category segments. The proof is based on the corresponding restrictions on pushdown automata. In particular, the automaton may look only at the top of the stack, represented in the rule by CAT-1, and the automaton may only push or pop one element at a time from the stack (with the corresponding result represented by CAT-3).

Context-sensitive languages are generated by LA-grammars where the length of the categories is bounded by $C \cdot n$, where C is a finite constant and n is the length of a complete well-formed input expression. The proof is based on the corresponding restrictions on linearly bounded automata.

(5) The Hierarchy of A-LAGs, B-LAGs, and C-LAGs

A more natural way of dividing possible languages in LA-grammar than the Chomsky hierarchy is the hierarchy of A-LAGs, B-LAGs, and C-LAGs. This new hierarchy is based on the properties of the categorial operations of the rules of LA-grammar. The crucial formal property of a categorial operation—from a complexity point of view—is whether or not it has to search through indefinitely-long sentence-start categories.

[5]For a detailed proof see Hausser (1989), Theorem 2, p. 135.

[6]For a detailed proof see Hausser (1989), Theorem 1, p. 134.

[7]For a detailed discussion see Hausser (1989), Section 8.2. A more general characterization of the regular languages is given in Theorem 3, p. 138.

(5.1) Definition of the Class of C-LAGs

The class of *constant* LA-grammars, or C-LAGs, consists of grammars where no categorial operation co_i looks at more than k segments in the sentence-start categories, for a finite constant k.[8] A language is called a C-language iff it is recognized by a C-LAG.

LA-grammars for regular and context-free languages are all C-LAGs because in regular languages the length of the sentence-start category is restricted by a finite constant, and in context-free languages the categorial operation may only look at a finite number of segments at the beginning of the sentence-start category. But the LA-grammars for many context-sensitive languages, e.g., $a^kb^kc^k$ (cf. (3.2)), $a^kb^kc^kd^ke^k$, WW, and WWW, are also C-LAGs.

Generally speaking, an LA-grammar is a C-LAG if its rules conform to the following schemas:

r_i: [(seg-1...seg-k X) CAT-2] ⇒ [rp_i CAT-3]

r_i: [(X seg-1...seg-k) CAT-2] ⇒ [rp_i CAT-3]

r_i: [(seg-1...seg-i X seg-i+1...seg-k) CAT-2] ⇒ [rp_i CAT-3]

Thereby, CAT-3 may contain at most one sequence variable (e.g., X).

On the other hand, if an LA-grammar has rules of the form

r_i: [(X seg-1...seg-k Y) CAT-2] ⇒ [rp_i CAT-3]

the grammar is not a constant LA-grammar. In non-constant LA-grammars CAT-3 may contain more than one sequence variable (e.g., X and Y).

Non-constant LA-grammars are divided into the *B-LAGs* and *A-LAGs*.

(5.2) Definition of the Class of B-LAGs

The class of *bounded* LA-grammars or B-LAGs consists of grammars where for any complete well-formed expression E the length of intermediate sentence-start categories is bounded by $C \cdot n$, where n is the length of E and C is a constant. A language is called a B-language if it is recognized by a B-LAG, but not by a C-LAG.

(5.3) Definition of the Class of A-LAGs

The class of A-LAGs consists of *all* LA-grammars because there is no limit on the length of the categories, or on the number of category segments read by the categorial operations. A language is called an A-language if it is recognized by an A-LAG, but not by a B-LAG.

The three classes of LA-grammars defined above are related in the following hierarchy:

(5.4) The Hierarchy of A-LAGs, B-LAGs, and C-LAGs

The class of A-LAGs recognizes all recursive languages, the class of B-LAGs recognizes all context-sensitive languages, and the class of C-LAGs recognizes many context-sensitive languages, all context-free languages, and all regular languages.

(6) Decidability

For arbitrary context-free grammars it is undecidable whether the languages generated are ambiguous, in an inclusion relation, or equivalent. In LA-grammar, on the other hand, questions of ambiguity, emptiness, inclusion, and equivalence are decidable for a large subset of the C-LAGs which includes context-sensitive languages. These results are based on the fact that the derivational structure of LA-grammar clearly exhibits the occurrence of grammatical recursions.

The following definition is based on the notion of "abstract derivations". Two derivations are represented by the same abstract derivation if they differ only in the choice of words, but exhibit the same sequence of rules and the same sequence of categories. In an abstract derivation different words of the same category, e.g., (table (sn)) and (chair (sn)), are represented by one abstract word, e.g., (A (sn)).

[8]This finite constant will vary between different grammars.

(6.1) Definition of a Grammatical Recursion

An abstract derivation exhibits a grammatical recursion if and only if

1. the surface exhibits two or more identical subsequences which are directly adjacent,
2. the rule sequence exhibits two or more identical subsequences which correspond to the surface, and
3. each instance of the recursion affects the sentence-start category in a regular way.

How sentence-start categories are affected by a recursion depends on the type of the recursion. LA-grammar distinguishes between (i) constant, (ii) increasing, (iii) decreasing, and (iv) simultaneously increasing and decreasing grammatical recursions. A recursion is constant if the sentence start categories at the beginning of two adjacent loops are identical. A recursion is increasing if the sentence start category at the beginning of the second loop is longer than the sentence start category at the beginning of the first loop. And correspondingly for the other cases.

Here is how the algorithm recognizes and types recursions: Assume the generator has derived a string of length n, and is in the process of adding the n+1st word—e.g., A—by means of a certain rule, e.g. 1.

(6.2) Example of a Grammatical Recursion

```
.......ABCABC A
.......123123 1 (cat)
```

The algorithm for recognizing recursions checks whether the current rule, i.e., rule 1, has two predecessors. If so, it checks whether the (abstract) surfaces added by the occurrences of rule 1 are all the same. If so, it checks whether the rule sequences and the surface sequences between the occurrences of rule 1 are identical. If all these conditions are satisfied, a recursion has been recognized. Finally, the recursion is typed by comparing the categories of the expression in question with its shorter predecessors ending in surface A and rule 1.

The crucial problem for proving decidability in LA-grammar is to determine how often grammatical recursions have to be applied in order for the set of completions to be a "representative sample". In the class of C-LAGs, the grammatical structure provides a "recursion factor" which specifies how often the increasing recursions of the grammar have to be applied in order to arrive at a representative sample. During the generation of longer and longer expressions, the system keeps track of increasing recursions and stops the recursion as soon as the number specified by the recursion factor has been reached.

In most C-LAGs this procedure results in a finite set of derivations which is representative in the sense that all sentence types generated by the grammar are exemplified in it. Such a representative sample provides the basis for deciding ambiguity, inclusion, equivalence, and (non-)emptiness of C-LAGs.

Not all C-LAGs are decidable, however. In C-LAGs with *simultaneously increasing and decreasing recursions such that the increase is greater than the decrease* the recursion factor does not guarantee the derivation of a representative sample. Those C-LAGs where the process of a systematic derivation, controlled by a grammar-dependent recursion factor, results in finite sets of representative samples are called D-LAGs ("Decidable C-LAGs"). An example of a D-LAG is $a^kb^kc^k$, for which a representative sample is derived in (3.4).

Can the technique of proving the subset and the equality relationship, as well as ambiguity and (non-)-emptiness for a large class of context-free and context-sensitive languages be used for PS-grammars as well? Because PS-grammars have a different derivational structure, the method of deriving longer and longer sentence starts cannot be applied directly in PS-grammar. The only possibility would be a systematic translation of PS-grammars into C-LAGs, and proving the properties in question indirectly by way of the weakly equivalent C-LAGs.

However, this approach requires that there is a general algorithm for translating PS-grammars into LA-grammars. No such algorithm has been found. Furthermore, experience writing LA-grammars for languages described originally as PS-grammars has shown that the construction of the LA-grammar is never based on the PS-grammar for the language, but proceeds from the language directly. Thus, it is unlikely that such an algorithm exists.

(7) The Complexity of Sound C-LAGs

Earley (1970) showed that the Earley algorithm recognizes **unambiguous** context-free grammars in $|G|^2 \cdot n^2$, but **ambiguous** context-free grammars in $|G|^2 \cdot n^3$ (where $|G|$ is the size of the grammar and n the length of the input string). Thus, computational complexity in PS-grammar depends not only on the class of the grammar, e.g., regular, context-free, or context-sensitive, but also on whether or not the grammar is ambiguous.

It is similar in LA-grammar: computational complexity depends not only on whether the grammar is a C-LAG, B-LAG, or A-LAG, but also on whether or not the grammar is ambiguous. LA-grammar distinguishes three levels of ambiguity:

(7.1) Three Levels of Ambiguity in LA-Grammar

1. unambiguous grammars
2. syntactically-ambiguous grammars
3. lexically-ambiguous grammars

Syntactic ambiguity is defined in terms of the input-compatibility of rules.

(7.2) Three Types of Input Conditions

1. **Incompatible input conditions:** Two rules have incompatible input conditions if there exist no input pairs which are accepted by both rules.
2. **Compatible input conditions:** Two rules have compatible input conditions if there exists at least one input pair accepted by both rules, and there exists at least one input pair accepted by one rule, but not the other.
3. **Identical input conditions:** Two rules have identical input conditions if it holds for all input pairs that they are either accepted by both rules, or rejected by both rules.

(7.3) Definition of Unambiguous LA-Grammars

An LA-grammar is unambiguous if and only if (i) it holds for all rule packages that their rules have *incompatible* input conditions, and (ii) there are no lexical ambiguities.

Examples of incompatible input conditions are [(a X)(b)] and [(c X)(b)], as well as [(a X)(b)] and [(a X)(c)].

(7.4) Definition of Syntactically-Ambiguous LA-Grammars

An LA-grammar is syntactically ambiguous if and only if (i) it has at least one rule package containing at least two rules with *compatible* input conditions, and (ii) there are no lexical ambiguities.

For example, [(a X)(b)] and [(X a)(b)] represent compatible input conditions.

(7.5) Definition of Lexically-Ambiguous LA-Grammars

An LA-grammar is lexically ambiguous if its lexicon contains at least two analyzed words with identical surfaces.

Because the categorial operations of C-LAGs look at no more than k sentence-start category segments, for some constant k, the application of a rule may be taken as the "primitive operation" for purposes of complexity analysis. Unambiguous C-LAGs are proven to parse in linear time. This result follows from definition (7.3), and is significant insofar as it applies not only to the (non-deterministic) context-free but also to many context-sensitive languages (e.g., $a^k b^k c^k$ as defined in (3.2)).

In the case of syntactically ambiguous LA-grammars, the crucial source of computational complexity are *recursive ambiguities.* In *sound* LA-grammars recursive ambiguities are restricted by the *single return principle.*

(7.6) The Single Return Principle (SRP)

> If a syntactic ambiguity arises inside a recursion, then only one of the branches resulting from the ambiguity may feed back into the recursion.

As a consequence of the SRP, sound LA-grammars have—at most—$(C \cdot n)$ readings.[9] Furthermore, the SRP does not decrease the generative capacity of an LA-grammar.[10] Because for any LA-grammar there exists a weakly equivalent sound LA-grammar, any syntactically ambiguous C-language can be parsed in n^2.

In the case of systematic lexical ambiguity, finally, there are two choices. One is to eliminate the lexical ambiguities by means of neutral categories. The other is to "pack" the readings, which may be exponential in number, into a single representation. It may be shown that these strategies are always possible within the class of C-LAGs.[11] In summary, if a language can be generated or recognized by a C-LAG then there exists a C-LAG which will parse it in n^2.

References

Ajdukiewicz, K. (1935) *"Die syntaktische Konnexität,"* Studia Philosophica, 1:1-27.

Berwick, R.C., and A.S. Weinberg (1984) *The Grammatical Basis of Linguistic Performance: Language Use and Aquisition.* The MIT-Press, Cambridge, Massachusetts.

Earley, J. (1970) *"An Efficient Context-Free Parsing Algorithm,"* CACM 13(2):94-102.

Hausser, R. (1989) *Computation of Language*, Springer-Verlag Berlin–New York (Symbolic Computation – *Artificial Intelligence*), June 1989.

Hopcroft, J.E., and Ullman, J.D. (1979) *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, Reading, Massachusetts.

Leśniewski, S. (1929) *"Grundzüge eines neuen Systems der Grundlage der Mathematik,"* Fundamenta Mathematicae, Warsaw.

Post, E. (1936) *"Finite Combinatory Processes — Formulation I,"* Journal of Symbolic Logic, I.

[9] C is some finite, grammar dependent constant reflecting the number of rules introducing recursive ambiguities and n is the length of the input.

[10] See Hausser (1989), Theorem 11, p. 224.

[11] Note that the problem of Boolean satisfiability (cf. Hopcroft & Ullman (1979), p. 325) exceeds not only the power of context-free grammars, but also of C-LAGs. An LA-grammar would not only have to build longer and longer categories in order to keep track of the different value assignments, but it would also have to check through the category each time it encounters another propositional variable. It is this second requirement which violates the definition of C-LAGs.

The selection of a parsing strategy for an on-line machine translation system in a sublanguage domain. A new practical comparison.

Patrick Shann
University of Geneva (ISSCO) & University of Zurich[1], Switzerland

1. Introduction

The paper reports the results of a practical comparison of different parsing strategies. The research was carried out in the context of a larger project for the development of a machine translation (MT) system for translating avalanche forecast bulletins from German to French. The design of the MT system requires controlled input and no post-editing of the translated texts. The parsing experiment had as a goal to select the most suitable parsing strategy for a parser that allows the composition of the sentences in on-line fashion with mouse and windowing[2]. In order to guarantee correct translation, the input system accepts only words and sentences that are known by their grammar and dictionary and it refuses wrong input. To minimize input errors, the user can select the possible next words with the mouse from different windows, which display the choices at a particular point in the sentence. The sentences are parsed word by word from left to right so that wrong input is detected immediately. After each word, the input device has to predict, with the help of the parser, all the words that can possibly continue the sentence that is being made. For our type of on-line parser, time is critical. The interface window has to be refreshed immediately after each word chosen by the user.

When we looked for a suitable parser, no comparison existed between Tomita's extended LR parser and enhanced chart parsers (top-down filter, rule compiling and lookahead) using different strategies (CKY, LC, BI[3]) apart from Tomita's own comparison with the Earley parser (TD). Furthermore, practical tests (Wirén 1987) are normally performed by using only simple phrase structure grammars and by measuring pure parse time. In our experiment we were interested in real time performance (what is seen by a user). Since the grammar type can heavily influence the overall processing efficiency, we chose to base our experience on three grammar types in the paradigm of context-free parsing (monadic, simple features and unification). Our parsing experiment is a continuation of the work of J.Slocum (1981a) and M.Tomita (1985) on parsing algorithms and parsing strategies. The emphasis of the research lies on the real-world performance of the parsers in connection with different grammar types rather than on the theoretical space and time complexity of the parsers.

2. Description of the parsers

In our experiment, we have compared the Tomita parser with four chart-parsers[4] that have different rule-invocation strategies. In this section we will introduce the different parsing strategies and the improvements that can be made, i.e. top-down filtering, lookahead and rule compilation.

2.1. Chart parsers

Our four chart parsers can be distinguished in the way they define the two basic operations *Combine* and *Propose*[5]. Combine is the procedure that builds new edges in the chart by combining existing ones, Propose is the rule invocation strategy that predicts new edges on the basis of the grammar. In the next chapter we define the basic algorithms. The improvements of the chart parsers are described in the following chapters on top-down filtering, lookahead and rule compilation.

2.1.1. Four different chart parsers: TD, LC, CKY, BI

Let G be a context-free grammar with S as start symbol. We will represent *terminal symbols* by lowercase letters: a, b, c; *nonterminals* by capitals: A, B, C; strings of *terminals or nonterminals*

[1] This research has been supported by a grant from the University of Zurich.
[2] A system with a similar input facility is reported by H.R.Tennant (1983).
[3] S. Steel & A. De Roeck (1987).
[4] We assume basic familiarity with chart parsing and with Tomita's LR parsing algorithm. For further literature on charts see Wirén (1987), for LR parsing Aho & Ullman (1979).
[5] Our Combine is more general than Winograds (1983) since we use a CKY variant with complete edges only.

with Greek letters: α, β, γ; *vertices* by: i, j, k; *edges* [1] as pairs of the rule in dotted notation and their left and right vertices. We will call the first symbol to the right of the dot in an active edge the *required category*. In the following example of an active edge, <A -> B · C D | i,j>, C is the required category, i the left and j the right vertex. TD, LC are implemented in such a way that they use only active edges, CKY only complete edges and BI active and complete edges.

2.1.1.1. Top-down (TD)

This strategy can be considered as Earley-like since it is very similar to Earley's algorithm apart from the fact that it does not use a lookahead. Some authors describe its Combine as the 'fundamental rule' of chart parsing[2].

> Combine
> Whenever a complete edge E_c <A -> α · | j,k> is added to the chart, combine it with all active edges E_a <B -> β · Cγ | i,j> ending at E_c's starting point j if E_c's category A corresponds to E_a's required category C and build the corresponding new edges <B -> βC · γ | i,k>.
>
> Propose
> Whenever an active edge E_a <A -> α · Bβ | i,j> is added to the chart, if its required category B is a nonterminal, for every rule B -> γ in the grammar G that expands E_a's required category B add an empty active edge E_x <B -> · γ | j,j> .

The parse runs top-down and is triggered by the first active edge <S -> · α | 0,0> expanding α with all the rules that have the start symbol S as left-hand side. It proceeds in a strict left-to-right fashion, the next input word is read when all Proposes and Combines up to the current input point have been executed. Opposed to the TD strategy are the two typical bottom-up parsers LC and CKY. Instead of using the rule selecting mechanism for building new hypotheses or active edges on the basis of required categories, the bottom-up parsers trigger the rules from the categories of complete edges.

2.1.1.2. Left-corner (LC)

As a bottom-up technique new edges are proposed on the basis of complete edges. The corresponding grammar rules are triggered if the first symbol of the right-hand side (RHS) of the rule , the 'left-corner', has the same category as the complete edge. LC and TD have the same 'Combine' and expand active edges from left to right.

> Propose
> Whenever a complete edge E_a <A -> α · | i,j> is added to the chart, for every rule B -> Aβ in the grammar G whose left-corner symbol A has the same category as E_a, add an active edge E_n <B -> A · β | i,j> to the chart.

2.1.1.3. Cocke-Kasami-Younger (CKY)

The second bottom-up parser is a variant of the Cock-Kasami-Younger algorithm. It is similar to CKY in the sense that it is pure bottom-up and combines only complete edges, but the grammar rules are not restricted to Chomsky normal form. To achieve this, Combine works from the right to the left and the rules are proposed on the rightmost symbol of the right-hand side.

> Propose
> Whenever a complete edge E_a <A -> α · | i,j> is added to the chart, propose all rules B -> βA in the grammar G, whose rightmost symbol is A.

[1] Edges correspond to Earley's (1970) 'states' and to 'items' in Aho & Ullman (1977).
[2] H. Thompson (1981). We will describe the two operations in a similar style to Thompson and Wirén (1987).

Combine

Whenever a complete edge E_c <A -> α · | i,j> is added to the chart, for each rule B -> βA that is proposed on A and for each combination of consecutive[1] complete edges starting with E_c and going to the left whose categories satisfy the sequence βA build a new complete edge E_n <B -> βA· | k,j> starting at the vertex k of its left-most edge and ending at the right vertex j of E_c.

2.1.1.4. Bi-directional (BI)

De Roeck (1987) gives the following motivation for bi-directional rule invocation. Form a linguistic point of view, certain phenomena like traces are best analysed top-down whereas others are best discovered from evidence in the string, e.g. in coordination, the conjunction is the best evidence for triggering the rule. But in the two bottom-up chart parsers the rules are triggered by a fixed handle, which is either the left-most or the right-most symbol of the RHS of a rule. In bi-directional chart parsing the linguist can tailor the rule invoking strategy locally by annotating the rules if they are used top-down or bottom-up. For bottom-up rules, one has to indicate which symbol they are triggered on. A rule for coordinating Np's can be annotated for example 'up' on the conjunction: Np -> Np Conj Np {up Conj}. When the complete edge for Conj is added to the chart, this rule will be triggered and it will add an active edge that tries to combine with an NP to the left as well as to the right. The Propose of the bi-directional parser acts accordin.g the the annotation of the rules. In order to avoid duplication Combine has been implemented in such a way that it first combines to the left and only then to the right. We have to expand the dotted rule notation in the sense that a colon marks the beginning of the recognized symbols of an edge and the dot the end of the recognized parts. Symbols to the right of a colon and to the left of a dot have been recognized. Our implementation proposes only to the right. An active edge can be left-active, if it is expecting a symbol to the left.

Propose

Whenever a complete edge E_a <A -> : γ · | i,j> is added to the chart, for every rule B -> αAβ annotated bottom-up on the symbol A, add an active edge E_n <B -> α : A · β | i,j> to the chart. Whenever an active edge E_a <A -> : α · Bβ | i,j> is added to the chart, if its required category B is a nonterminal, add an empty active edge E_x <B -> · δ | j,j> for each rule in the grammar G that is annotated down and that expands E_a's required category B.

Combine

Whenever a left active edge E_a <A -> α : γ · β | i,j> is added to the chart, for each combination of complete edges starting with E_a and going to the left whose categories satisfy the sequence α build a new active edge E_n <A -> : αγ · β | k,j> starting at the vertex k of its left-most edge and ending at the right vertex j of E_a.
Whenever a complete edge E_c <A -> α · | j,k> is added to the chart, combine it with all active edges E_a <B -> : β · Cγ | i,j> ending at E_c's starting point j if E_c's category A corresponds to E_a's required category C and build the corresponding new edges <B -> : β C · γ | i,k>.

The bi-directional chart parser was included in the tests for verifying the hypothesis if triggering annotations of the rules reduce the search space and improve the overall performance.

2.1.2. Top-down filter (tdf)

In general, bottom-up algorithms have a reduced search space by the fact that they are data-driven. On evidence of complete edges, that are present in the string, they are faster in finding the corresponding rules. They do not have to explore the whole search space of the grammar as the TD parser that is over-productive in active edges. On the other hand, bottom-up parsers have problems in dealing with rules that have common right parts as in the following example: 'CD' is the common right string of both rules A -> BCD and A -> CD. Both rules will fire on a string 'BCD'. Bottom-up chart parsers are over-productive in complete edges that do not attach to phrases on the left. The next two chapters deal with filters to reduce over-production of useless edges: top-down-filtering, a method for bottom-

[1] Two complete edges can be combined to the left if the starting vertex of the first edge corresponds to the ending vertex of the second one.

up parsers to reduce the production of useless complete edges and lookahead, a method to reduce the production of unnecessary active edges, useful for TD, LC and BI.

Top-down-filtering is described like running a top-down parser in parallel with a bottom-up parser[1]. The bottom-up parser proposes new edges while the top-down process checks if they can be derived from the root. The tdf rejects all proposed rules that will generate phrases that can't be attached to the left context. The tdf uses a "reachability relation $\mathcal{R}$ where A$\mathcal{R}$B holds if there exists some derivation from A to B such that B is the left-most element in a string derived from A" (Wirén 1987, cf also Pratt 1975). The reachability relation $\mathcal{R}$ can be precompiled so that the tdf can check in constant time if $\mathcal{R}$ holds for a new proposed edge.

In the LC parser, the tdf is implemented in the following way: For each nonterminal category A the transitive closure of the categories that are reachable from A are precalculated . At each vertex, the tdf keeps a list of the reachable categories. Vertex 0 is initialised with the list of the categories that are reachable from the root category. For each new active edge E_n, the tdf adds the categories that are reachable from the new required category to the tdf -list of reachable categories at the ending vertex of E_n. In the function Propose, the tdf checks for every proposed rule if its left-hand side category is in the list of the reachable categories of the current vertex. Only rules that pass the tdf lead to the creation of new active edges.

2.1.3. Lookahead (la)

Top-down-filtering cuts down the production of useless complete edges in bottom-up parsing by checking if they can combine with the left context. The lookahead function verifies if a new edge can be attached to the right context. Wirén (1987) reports an experiment where la was used successfully to reduce the over-production of active edges in TD or LC[2]. La is based on the same reachability relation as tdf but is looking to the right. Each time an active edge is proposed, the la function checks if the new required category C_n can reach the preterminal category of the next input word a_{i+1}, that is if $C_n \mathcal{R} a_{i+1}$ holds. We have tested all our parsers without lookahead.

2.1.4. Rule compilation

The third method for reducing the number of edges in chart parsing is precompiling the grammar rules into decision trees. Assume two rules used by a LC parser, A -> BC and A -> BDE. The two rules have the common left part B and can therefore be merged into a single combined rule with a shared part B: A -> B (C, DE). In parsing, the two rule scan share the common part B which is represented by a single active edge. TD and LC compile the rules by factoring out similar left parts. CKY combines from right to left and does therefore the factoring from the right. BI, based on annotations of single rules, uses both ways of building its rule decision trees. Note that building decision trees for rules is related to the way in which the canonical set of items is built for the construction of LR parsing tables. The first step in making a new canonical LR set is done by taking all the items in a set that have the same category to the right of the dot. Building decision trees from rules also groups them together on the basis of the next category that has to be recognized.

2.2. Tomita's extended LR parser (TOM)

Tomita's Parser (Tomita 1985) is a generalised version of a LR shift-reduce parser. It is based on two data structures: a graph structured stack and a parser forest for representing the result. The graph-structured stack allows nondeterministic parsing of ambiguous grammars with LR shift-reduce technique. Tomita (1988) shows that his graph-structured stack is very similar to the chart in chart parsing. The parse forest allows an efficient representation of the result. While the number of parses can grow exponentially, the parse forest grows polynomially. In order to see which part of the program is responsible for efficiency, we compare two versions of Tomita's parser, one with and one without parse forest.

[1] J. Slocum (1981b), M.Kay (1982), Pratt (1975), Wirén (1987).

[2] Earley uses the lookahead in a different way: The lookahead is in his Completer and not in the Predictor, as in Wiréns programms.

2.3. The grammar types

Each parser can be run with three different types of context-free grammars. This is done by adding annotations to the context-free rule skeleton. Whenever all constituents of a context-free rule are found, before the new edge is constructed, the parser calls for a rule-body procedure (Slocum 1981b) that evaluates the annotations of the rule. Each grammar type has a different module for evaluating the rule-body procedure. If the rule-body procedure returns an error because a test has failed, the new edge is discarded.

The first grammar type uses simple phrase structure rules with monadic categories that have no annotations. The second grammar type has annotations that go with simple sets of attribute-value pairs where the values are atomic. These annotations allow testing and assigning features to particular nodes of the context-free rules. The third grammar type is unification based and uses complex features and annotations in the PATR-II style. The three grammar types vary the rule-body procedure overhead (unification being very time consuming) and therefore show a more realistic picture of the behaviour of the parsers in real context.

3. Previous empirical comparisons

In this section we report the results of three practical comparisons of parsers relevant to our experiment: Slocum who compared particularly LC and CKY with top-down filter, Tomita who compared his extended LR parser with Earley's parser and Wirén who compared TD and LC with top-down filter and lookahead. Each of the comparisons gives an incomplete picture. They usually compare two basic strategies with different refinements like top-down filtering etc.

One of the important points for comparisons is stressed by Slocum (1981b): Theoretical calculations about worst case behaviour of algorithms can be quite inaccurate because they often neglect the constant factors that seem to have a dominant effect in practical situations. He writes: "In order to meaningfully describe performance, one must take into account the complete operational context of the natural language processing system, particularly the expenses encountered in storage management and applying rule-body procedures" since a significant portion of the sentence analysis effort may be invested in evaluating the rule-body procedures. To measure performance accurately he suggests including "everything one actually pays for in real computing world: Paging, storage management, building interpretations, rule-body procedure, etc., as well as parse time".

3.1. Slocum: two bottom-up chart parsers, LC vs. CKY

Slocum has conducted two experiments, one at SRI and the second one at LRC, which is more important for us. In the second experiment, he carefully compared two bottom-up chart parsers: LC and CKY enhanced with top-down filtering and early constituent tests[1]. He used the German analysis grammar (~ 500 rules) of the MT system that was under development at the time at LRC and a corpus of 262 sentences going from 1 - 39 words per sentence (15,6 words/sentence average). The rule-body procedures were rather considerable for a parser test but interesting for realistic performance evaluation. They consisted of "the complete analysis procedures for the purpose of subsequent translation which includes the production of a full syntactic and semantic analysis via phrase-structure rules, feature tests and operations, transformations and case frames".

Given his grammar and test sentences Slocum establishes two things:

1) LC with tdf (without early constituent test) performs best, better than CKY (which is the opposite of the common expectation). He comments that a tdf increases the search space, but that the overhead is balanced in practice by the fact that the tdf reduces the number of phrases and therefore particularly the rule-body procedure overhead, which is considerable in his case. "The overhead for filtering in LC is less than that in CKY. This situation is due to the fact that LC maintains a natural left-right ordering of the rule constituents in its internal representation, whereas CKY does not and must therefore compute it at run time."

1 "The early constituent test calls for the parser to evaluate that protion of the rule body-procedure which tests the first constituent, as soon as it is discovered, to determine if it is acceptable" (Slocum 1981b)

2) "The benefits of top-down filtering are dependent on sentence length: in fact filtering is detrimental for shorter sentences. Averaging over all other strategies, the break-even point for top-down filtering occurs at about 7 words."

We conclude this section with a statement from Slocum about filters: "Filtering always increases pure parse time because the parser sees it as pure overhead. The benefits are only observable in overall system performance, due primarily to a significant reduction in the time/space spent evaluating rule-body procedures." This point will be important in our comparisons since we use three different grammar types with rule-body procedures that take increasingly more time.

3.2. Tomita: The Tomita parser vs. Earley's algorithm

Tomita (1985) compared his parser empirically with two versions of the Earley algorithm (E-I and E-II). In our terminology this would correspond to TD and TD+la. While the Tomita parser was producing a parse forest, E-I and E-II were run as recognizers and produced no parse.

In the comparison, four pure context-free phrase-structure grammars were used, consisting of a varying number of rules: G1 8, G2 40, G3 220 and G4 400 rules. These grammars were tested with two sets of sentences, S1: 40 sentences from texts and S2: 13 artificial sentences that have an increasing number of prepositional phrases (1 to 13). These artificial sentences are useful for testing growing sentence ambiguity since the number of parses grows exponentially (Martin et al. 1981).

Tomita's experiment shows that his algorithm works 5 to 10 times faster than Earley's standard algorithm (TD), and 2 to 3 times faster than Earley's improved algorithm (TD+la). He states that this result is due to the pre-compilation of the grammar into an LR table. Tomita summarizes that his algorithm "is significantly faster than Earley's algorithm, in the context of practical natural language processing. . . Its parsing time and space remain tractable when sentence length, sentence ambiguity or grammar size grows in practical applications."

3.3. Wirén: top-down and bottom-up chart parsers, TD vs. LC

Wirén compared in his experiment two basic chart parsers with several improvements, TD versus LC, both with lookahead, LC with top-down filtering[1]. He tested his parsers with grammars G1 to G3 from Tomita, with a reduced number of the two sentence sets, S21 and S2.

The results of his experiments show that the "directed methods" - based on top-down filtering and lookahead - reduce significantly the number of edges and perform better than undirected parsers. Tested independently, the selectivity filter (lookahead in our terminology) turned out to be much more time efficient than top-down filtering that degraded time performance as the grammar grew larger[2]. "The maximally directed strategy - ... with selectivity and top-down filtering - remained the most efficient one throughout all the experiments, both with respect to edges produced and time consumed." It performed better than TD with lookahead.

Putting the results of the three experiments together, we would expect that improved LC performs best amongst chart parsers. Since the Tomita parser has only been compared with TD, we can expect a different result by comparing it with improved bottom-up chart parsers that compile their rules into decision trees (cf. chap. 2.1.4). Tomita and Wirén measure pure parsing time determined by CPU time minus time for garbage collection. Their grammars are pure CF grammars using little rule-body procedure time and it is therefore difficult to predict what the interaction will be between filtering overhead and rule-body procedure and how this will influence overall performance.

4. The comparison

4.1. The parsers

Our main goal was the selection of a suitable parsing strategy for our on-line MT-system. Since our application is time critical, one of the important questions was what combination of parser and rule-

[1] LC à la Kilbury has already been used by Slocum. What it comes down to is that new active edges subsume the complete edges that have provoked their proposal. Since we use that variant of LC (cf. 2.1.1.2) coming from Slocum (1981a), we don't distinguish between a standard LC and the Kilbury variant.

[2] Wirén explains this puzzle with implementational reasons.

body procedure is best for our purpose. One of the objectives was to verify if the Tomita parser is as efficient as predicted if it is compared to improved bottom-up chart parsers. Since no comparison existed between all the basic rule invocation strategies for chart parsers, we decided to compare the Tomita parser with four chart parsers. To guarantee the comparability of the chart parsers, we chose Slocum's implementation (1981a) as basic design for all chart parsers. We added two supplementary rule invocation strategies to his bottom-up left-corner (LC) and Cocke-Kasami-Younger strategy (CKY), namely a top-down Earley-like strategy (TD) and a bi-directional strategy (BI). The basic chart parsers were augmented by two enhancements, i.e. top-down filtering and compilation of the rules into decision trees. We took the Tomita parser as described by Tomita (1985) and added a second version without the parse forest representation. Since its LR(0) parsing table has no lookahead, we added no lookahead to the chart parsers.

All the programs are implemented in Allegro Common Lisp and tested on a Macintosh II (MC68020 with 5 MB RAM). As main parameters we compared number of edges, number of rule-body procedure executions and over-all time.

4.2. The grammars and sentences

The first test uses small grammars (22 and 80 rules) together with the same 50 artificial sentences. The monadic grammars are tested with all 9 parsing strategies (TOM +/-parse forest; TD, LC, CKY, BI, the bottom-up parsers +/-tdf), for features and unification grammars we use TOM without parse forest and all the chart parsers. The 50 test sentences are constructed artificially to control parameters like sentences ambiguity, sentences length and three linguistic phenomena, i.e. PP-attachment, relative clauses and coordination. They can be classified into two groups, one where ambiguity grows exponentially with increasing sentence length (PP-attachment and coordination), and a second group, where the sentence length does not influence ambiguity (they have 1 to 3 readings). The sentence length varies from 3 to 24 words. Each grammar type has two small grammars with approximately 25 resp. 80 rules.

The second test compares a reduced number of parsers (TOM, TD, LC, CKY, bottom-up +/-tdf) with a bigger monadic grammar based on the German avalanche corpus that has 750 rules and 300 lexical items. The 50 test sentences were taken from the avalanche corpus, their length varies from 6 to 42 words (average 19 words per sentence).

5. Test-results and discussion

Before we comment, we will give a brief outline of how we present the test-results in appendix 1 and 2. The seven tables in appendix 1 summarize the statistics for each grammar and set of sentences. We give the total number of edges and the total time for each parser over all sentences. The figures for time indicate overall time[1] that includes rule-body procedure etc. The reader should be careful in the interpretation of the timings; these figures are dependent on machine, lisp system and the way in which the algorithms are programmed. Nevertheless, we think that they give an indication of relations. Appendix 2 shows a limited number of diagrams to illustrate the figures graphically.

In appendix 1, each table shows three fields, one for the number of edges and two for timings: 1) the total time for all sentences and 2) the time for 26 sentences with low ambiguity. The second group of test sentences includes relative clauses and coordinations. The number of words per sentence goes from 5 to 23 words (13 average) and they have 1 to 5 readings. Time is measured in milliseconds. The column 'diff' indicates the difference of the parsers from Tomita which is set to 1. In the field 'time all', we added the average time per word (ms/word) in order to have a figure that can easily be compared across the different tests. We have listed the number of edges because this figure is often given as measurement for parser performance. But one can observe that the rankings based on the number of edges and the one based on timing do not correspond. This is due to the particular way in which the chart parsers are implemented. As we have mentioned in chap. 2.1.1, TD and LC keep only active edges in the chart, whereas CKY has only complete edges and BI both. For TOM, we counted the number of shift operations.

Since there is limited space for diagrams, most of them show three parsers: TOM, TD and LC +/-tdf. All the diagrams display the time/word relation for a particular grammar and a sentence set. Diagram 1

[1] Since we have forced a garbage collection before each sentence, the garbage collector does not interfere with the timings.

and 2 show PP-attachment (high ambiguity: a 20 word sentence has 132 parses), diagram 3 the time/word relation for the 750 rule grammar and all the avalanche sentences. Diagram 4 represents the times for LC +/-tdf with the three different grammar types for a set of coordinations in high ambiguity. Diagram 5 shows all parsers with a set of relative clauses that have low ambiguity.

5.1. The chart parsers

Our tests confirm Slocum's and Wirén's data: the left-corner parser (LC) with top-down filtering is overall the most efficient chart parser. It ranks highest among the chart parsers with all grammar types and grammar sizes. The only exceptions are monadic and feature grammars of the size of 80 rules with low ambiguity sentences (see below 5.3.). Earley-like top-down (TD) with the two small grammars is highly overproductive in active edges and therefore a bad choice if it is used without lookahead. Diagram 1 and 2 show how TD is influenced negatively by the grammar size, the grammar in diagram 2 has three times more rules. Strangely enough, in the large grammar (table 3 and diagram 3), TD is converging towards LC as the sentences grow longer. In diagram 3, one can see well its initial overhead of active edges .

The bi-directional chart parser (BI) was included in the tests for verifying the hypothesis if triggering annotations on the rules reduce the search space and improve the overall performance. None of our tests could confirm such a hypothesis. It seems that top-down filtering or lookahead influence performance to a greater extent than linguistic triggering annotations. BI did not perform better with any particular set of test sentences or grammars.

5.2. The Tomita parser and chart parsers

Diagram 1 and 2 show how the Tomita parser (to+) performs best in situations of high ambiguity. Taking the overall timings in table 1 and 2, TD is 4.75 to 6.53 times slower than TOM (and our comparison stops at sentences with 20 words with 132 readings). The situation is less dramatic if we take LC+tdf. Here the difference is 1.67 to 1.9. But, if we take our grammar of 750 rules with its low ambiguity sentences, the gap is much smaller: 1.38 for LC+tdf and 2.15 for TD. A closer look at diagram 1 and 2 shows that TOM without parse forest (to-) is roughly equivalent to LC+tdf (lc+). We therefore think that the major speed gain of TOM comes from its parse forest, which is an efficient way of packing the parse trees. But, this representation could be used with any parser and is not specific of TOM. In diagram 3, TOM and LC+tdf show a constant time difference. Precompiling the grammar rules into a LR parsing table or precompiling them into decision trees does not make a crucial difference, even with very long sentences of up to 42 words and a large grammar of 750 rules.

5.3. Filters, grammar size and rule-body procedures

This chapter tries to address the complex interaction between parsing strategy, grammar size, sentence ambiguity and overheads for top-down filtering and rule-body procedure. There is no standard grammar size. According to the grammar type, the size varies. We estimate that unification grammars, which are highly lexical, might have 50 to 100 rules, grammars with simple features around 500[1], and monadic grammars several thousand rules.

In general, a TD parser is disadvantaged if the grammar has a high branching factor because of its overproduction of active edges (cf. chap. 2.1.3.). Bottom-up parsers suffer from rules with common right factoring in the right-hand side of the rules (cf. chap. 2.1.2.). A grammar might produce different results about TD overproduction or top-down filters according to its branching factor or right factoring. The effect of a top-down filter is not always a good one. We have contradicting results about top-down filtering. In the test with the monadic grammar of 750 rules, the two chart parsers with top-down filter (lc+ and cky+) perform better than their counterparts without filter. Diagram 3 also shows a converging TD and a diverging LC-tdf (lc-) as the sentence length increases. This is due to the high right factoring of that grammar. The opposite result is shown by monadic and feature grammars with 75 rules together with the sample of low ambiguity sentences. In these cases, the overhead from the top-down filter deteriorates the efficiency of the chart parsers with top-down filter. Unfiltered parsers with sentences up to 19 words are faster than the filtered ones. This result is influenced by the nature of the grammar as well as its size since the top-down filter with the small grammars (22 or 30 rules) shows a positive effect.

[1] The Metal German analysis grammar, which is based on simple features, has 500-600 rules.

Another tradeoff is between top-down filter and rule-body procedure. In our tests we compare three different types of rule-body procedures: no annotations in monadic grammars or simple features and unification. Monadic grammars and simple feature grammars have a small rule-body procedure whereas the overhead for unification is considerable (2/3 for unification and 1/3 for pure parsing). Diagram 4 shows optically that the top-down filter has a positive effect as the rule-body procedure grows. With a time consuming rule-body procedure, a top-down filter becomes vital for the overall efficiency. This statement should not be interpreted as a generalization about simple feature grammars versus unification. Our point is independent of a particular grammar type but has to do with the relation between pure parse time and rule-body procedure time.

5.4. Sentence length

As we reported in chap. 3.1., Slocum claims that the benefits of top-down filtering are dependent on the sentence length and that the break-even point for top-down filtering (averaged over LC and CKY) occurs at about 7 words. As we have shown above, the question is more complex and influenced furthermore by the number of parses as well as by the nature and by the size of the grammar (right factoring and branching factor). Some of our tests show clearly that the length of the sentence is not necessarily the main parameter. We believe that no generalization is possible unless all the mentioned factors are taken into account.

5.5. Final choice

The choice of the parsing strategy for our MT-system was guided by the following ideas: Possible candidates for an on-line parser that parses strictly from left to right are TOM, LC+tdf and TD. Given the performance, TD was ruled out. The question of the grammar type was more difficult to solve. The grammar has to predict all the sentences but only the correct ones, no overproduction is allowed. We therefore have to subclassify heavily by using a system of about 100 grammatical and semantic features. The worst cases for an empirical efficiency test are sentences with high ambiguity. Diagram 4 shows the performance of the three grammar types where the 20 word sentence has the highest ambiguity. The average time per word varies heavily according to the grammar type: monadic - 70 ms, features - 160 ms and unification -1267 ms. Unification is slower by a factor of about 20. This factor would be increased by the search for possible next words because it is not a simple matching of categories but a complicated search that has to take into account all the instantiated variables from constituents that have already been found. Given this poor expectation for unification grammar in on-line parsing, we were left with two grammar types, and we opted for simple monadic grammars, rather as a matter of computational simplicity. Together with monadic grammars, we chose the Tomita parser, because it was slightly more performant with the large grammar for the avalanche corpus, and last but not least, because of its elegance. We like the idea of precompiling the grammar into a LR table.

We have come to the conclusion that it is very difficult to test empirically the performance of algorithms or better of programs and to find good generalizations[1]. Nevertheless, we believe that we have shown that the parse forest representation is to a large extent responsible for the good performance of the Tomita parser, and second, that the difference in efficiency between the Tomita parser without the parse forest representation and an enhanced left-corner parser with top-down filtering and compiled rules is small. Two points of empirical research have not been addressed in our tests, which could also help the practitioners of computational linguistics when they have to select their parsing strategies: 1) We have excluded the use of a lookahead. We think that this point needs further investigation (i.e. TOM with an LALR table versus LC+tdf with la). 2) Since the parse forest representation is highly efficient, its benefits in combination with unification grammars need more clarification.

6. Acknowledgement

I would like to thank Anne De Roeck and Tony Lawson for all the theoretical and practical discussions as well as for the contribution of the unification grammar from Anne and the unifyer from Tony. Thanks also to Mike Rosner, Rod Johnson, Dominique Petitpierre and Thomas Russi for their comments and useful hints.

[1] On a different machine with a different lisp system the same programs might behave differently.

7. References

A. Aho and J. Ullman (1979), *Principles of compiler design*, Addison Wesly.

J. Earley (1970), An efficient context-free parsing algorithm, *Communications of the ACM 13(2)*, 94-102.

M. Kay (1982), *Algorithmic schemata and data structures in syntactic processing*, CSL-80-12, Xerox Parc, Palo Alto.

W. Martin, K.Church & R. Patil (1981), Preliminary analysis of a breadth-first parsing algorithm: Theoretical and experimental results, MIT LCS Technical report.

V. Pratt (1975), LINGOL - A progress report, *Proc. 4th IJCAI*, Tbilisi, 422-428.

J. Slocum (1981a), *A practical comparison of parsing strategies for machine translation and other natural language processing purposes*, PhD University of Texas, Austin.

J. Slocum (1981b), A practical comparison of parsing strategies, *Proc. 19th ACL*, Standford.

S. Steel & A. De Roeck (1987), Bi-directional parsing, in: Hallam & Mellish (eds.), *Advances in AI, Proc. of the 1987 AISB Conference*, J. Wiley, London.

H.R. Tennant et al. (1983) , Menu-based natural language understanding, *Proc. 21st ACL*, 151-158.

H. Thompson (1981), Chart parsing and rule schemata in GPSG, *Proc. 19th ACL*, Stanford .

M. Tomita (1985), *An efficient context-free parsing algorithm for natural languages and its applications*, PhD CMU Pittsburg. Also as: *Efficient parsing for natural language. A fast algorithm for practical purposes.* Kluwer, Boston1986.

M. Tomita (1987), An efficient augmented-context-free parsing algorithm, *Computational Linguistics* 13(1/2).

M. Tomita (1988), Graph-structured stack and natural language parsing, *Proc. 26th ACL*, Buffalo.

J. Winograd (1983), *Language as a cognitive process, Syntax*, Addison-Wesley.

M. Wirén (1987), A comparison of rule-invocation strategies in context-free chart parsing, *Proc. 3rd European chapter ACL*, 226-233.

Appendix 1a

Table 1 Monadic grammar: 22 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6532	5	3.08	24049	2	1.67	38	8000	3	1.07
lc-	13332	8	6.28	41418	7	2.88	66	12301	6	1.64
cky+	3449	2	1.62	33219	4	2.31	53	12901	7	1.72
cky-	6886	6	3.24	34634	5	2.41	55	9599	4	1.28
bi+	6497	4	3.06	44483	8	3.10	71	15850	9	2.11
bi-	9655	7	4.55	36265	6	2.52	58	10033	5	1.34
td	19766	9	9.31	68217	9	4.75	109	12985	8	1.73
tom	2124	1	1.00	14364	1	1.00	23	7498	1	1.00
to-2	3881	3	1.83	25756	3	1.79	41	7940	2	1.06

Table 2 Monadic grammar: 75 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6224	5	3.38	24418	3	1.90	39	8417	6	1.36
lc-	9020	8	4.90	27834	6	2.16	44	7318	5	1.18
cky+	2803	2	1.52	38980	7	3.03	62	16481	7	2.66
cky-	4884	6	2.65	25650	4	1.99	41	6150	1	0.99
bi+	7476	4	4.06	56649	8	4.40	90	20084	8	3.24
bi-	8277	7	4.50	25815	5	2.00	41	6730	4	1.09
td	33028	9	17.95	84130	9	6.53	134	20665	9	3.33
tom	1840	1	1.00	12883	1	1.00	21	6200	2	1.00
to-2	3117	3	1.69	21899	2	1.70	35	6382	3	1.03

Table 3 Monadic grammar: 750 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word
lc+	3693	3	1.94	20132	2	1.28	21
lc-	18411	6	9.67	51132	6	3.26	54
cky+	1923	2	1.01	36715	4	2.34	39
cky-	6662	4	3.50	40785	5	2.60	43
td	16951	5	8.90	33717	3	2.15	35
tom	1904	1	1.00	15684	1	1.00	16

Abbreviations

+	+ top-down filter (tdf)
-	- tdf
tom	Tomita + parse forest
to-2	Tomita - parse forest

Table 4 Feature grammar: 30 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6067	4	2.10	38669	1	0.93	62	11434	2	1.06
lc-	12666	7	4.39	76415	6	1.85	122	18483	8	1.71
cky+	2661	1	0.92	47015	3	1.14	75	15233	4	1.41
cky-	5304	3	1.84	69844	5	1.69	111	16213	5	1.50
bi+	6165	5	2.14	64901	4	1.57	103	17517	7	1.62
bi-	10266	6	3.56	81869	7	1.98	130	14050	3	1.30
td	21669	8	7.52	114548	8	2.77	182	16982	6	1.57
to-2	2883	2	1.00	41368	2	1.00	66	10818	1	1.00

Table 5 Feature grammar: 80 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	6232	4	2.02	41265	1	0.96	66	12383	3	1.04
lc-	8963	7	2.91	60867	6	1.42	97	14649	5	1.23
cky+	2831	1	0.92	57884	4	1.35	92	20668	6	1.73
cky-	4871	3	1.58	59248	5	1.38	94	13983	4	1.17
bi+	7459	5	2.42	80217	7	1.87	128	26467	8	2.22
bi-	8198	6	2.66	49901	3	1.16	79	11967	2	1.00
td	32792	8	10.64	135650	8	3.16	216	24985	7	2.10
to-2	3083	2	1.00	42985	2	1.00	68	11917	1	1.00

Table 6 Unification grammar: 30 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	5449	3	1.79	144349	1	0.91	251	23433	1	0.93
lc-	11446	6	3.75	525250	7	3.32	913	74750	8	2.97
cky+	2813	1	0.92	153500	2	0.97	267	27233	3	1.08
cky-	7068	4	2.32	533515	8	3.38	928	73167	7	2.91
bi+	8675	5	2.85	210300	5	1.33	366	29034	5	1.16
bi-	14247	7	4.67	307949	6	1.95	536	37866	6	1.51
td	18795	8	6.16	169684	4	1.07	295	27983	4	1.11
to-2	3049	2	1.00	158032	3	1.00	275	25132	2	1.00

Table 7 Unification grammar: 80 rules

	edges	rank	diff	time all (ms)	rank	diff	ms/word	time 2	rank	diff
lc+	5519	3	2.00	108382	1	0.95	188	20531	2	1.03
lc-	12527	7	4.54	272181	8	2.39	473	42549	8	2.14
cky+	2483	1	0.90	122618	3	1.08	213	27603	3	1.39
cky-	5700	4	2.07	268468	7	2.36	467	41485	7	2.09
bi+	6770	5	2.45	135834	4	1.19	236	29918	4	1.51
bi-	12232	6	4.43	189650	6	1.67	330	30217	5	1.52
td	34093	8	12.36	146203	5	1.29	254	31551	6	1.59
to-2	2759	2	1.00	113750	2	1.00	198	19866	1	1.00

Appendix 2

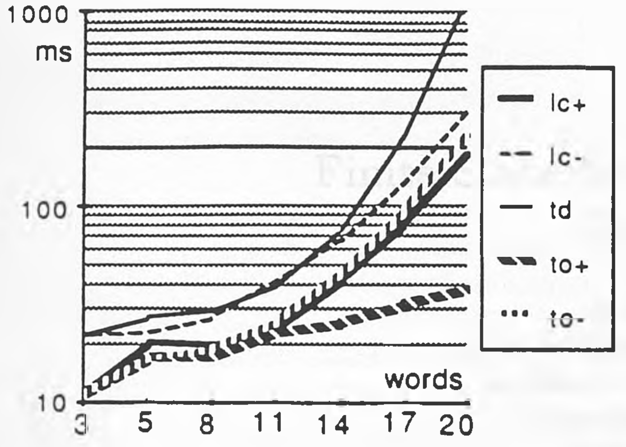


Diagram 1 Monadic-22 PP-attachment

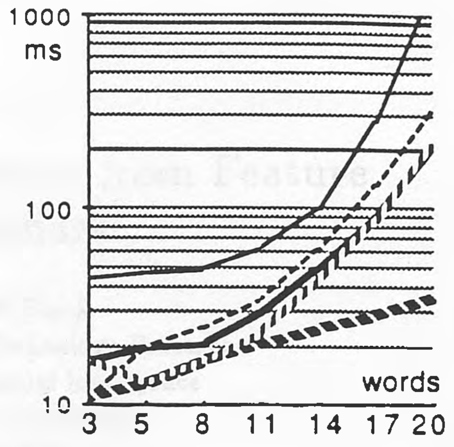


Diagram 2 Monadic-75 PP-attachment

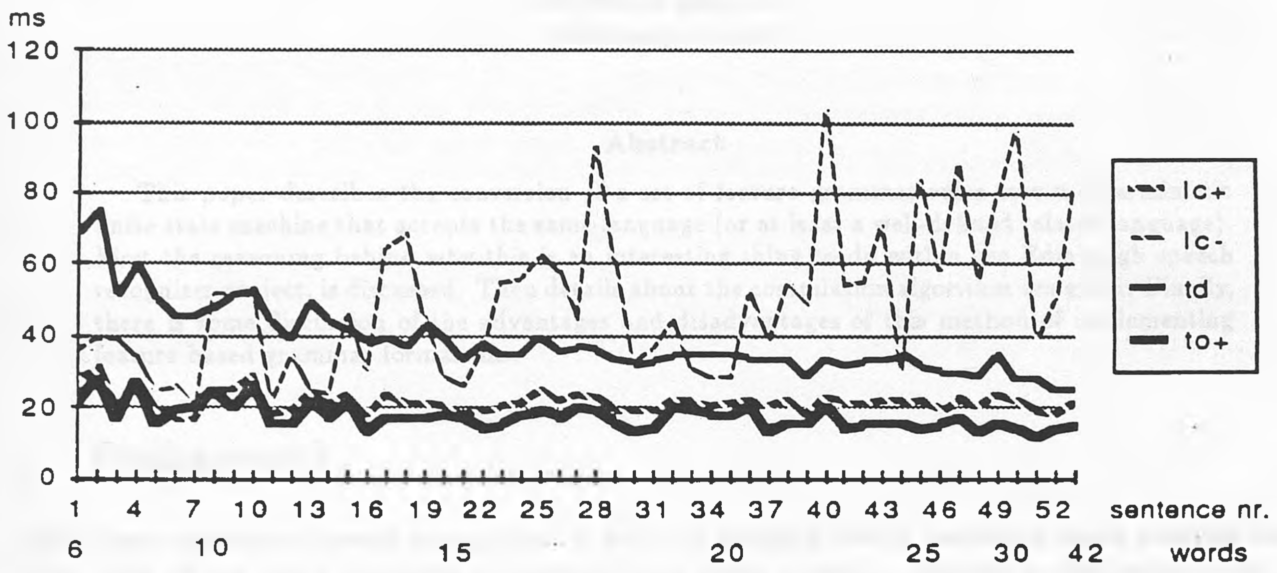


Diagram 3 Monadic-750 (all sentences)

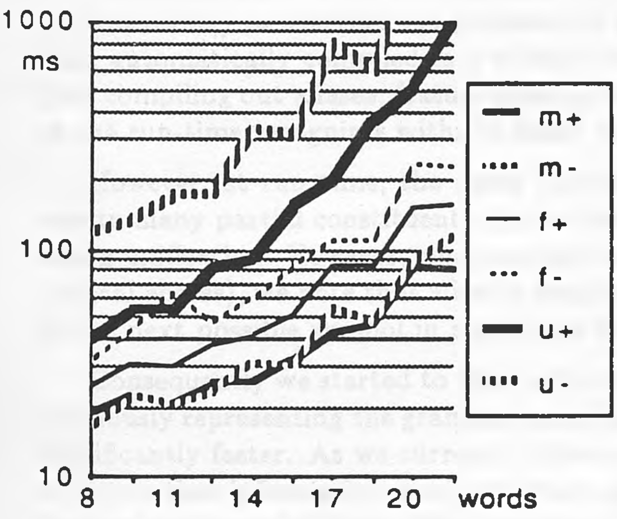


Diagram 4 LC +/-tdf (30 rules)
3 grammar types, coordination
high ambiguity

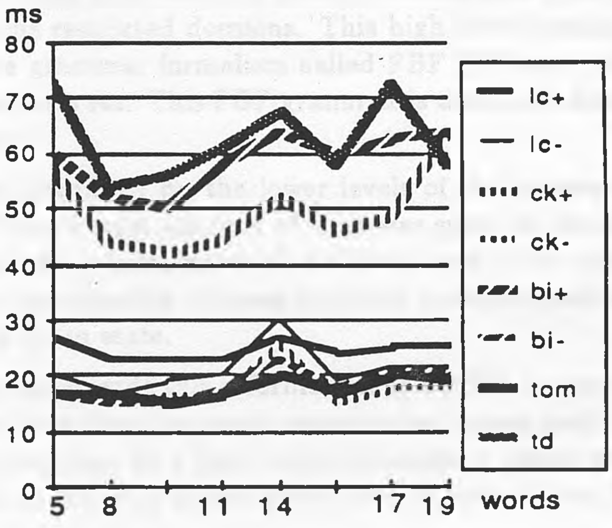


Diagram 5 Monadic-75, relative clauses
low ambiguity

Finite State Machines from Feature Grammars

Alan W Black
Centre for Speech Technology Research
and Dept of Artificial Intelligence
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
awb@eusip.ed.ac.uk

Abstract

This paper describes the conversion of a set of feature grammar rules into a deterministic finite state machine that accepts the same language (or at least a well-defined related language). First the reasoning behind why this is an interesting thing to do within the Edinburgh speech recogniser project, is discussed. Then details about the compilation algorithm are given. Finally, there is some discussion of the advantages and disadvantages of this method of implementing feature based grammar formalisms.

1 Background

Real-time continuous speech recognition is still not possible but is becoming more possible each year. One of the many problems in recognition is doing symbolic analysis in the higher levels of the system in a reasonable time.

Within CSTR, we are investigating analyses using high level GPSG-type formalisms (like that in [Gazdar85]) to describe the grammar of various restricted domains. This high level notation is then automatically compiled into a basic feature grammar formalism called FBF ([Thompson89]) thus compiling out aliases, feature passing conventions etc. This FBF grammar is then used directly in the run-time recogniser within a chart parser.

However, at run time, the many hypotheses predicted by the lower levels of the system give rise to many partial constituents in the chart. Thus a large amount of time was spent in the chart doing unification. However, when we look at the real requirements of the lower level of the system (lexical access), we note that what is required in the majority of cases is merely a simple prediction of the next possible symbol in a sentence from a given state.

Consequently we started to think about ways to provide this information as quickly as possible. Obviously representing the grammar as a Finite State Machine would make lexical access prediction significantly faster. As we currently write our grammars in a high level formalism it seems wrong to throw that information away and start again, so we hope to find some form of compilation from feature grammars to finite state grammars.

Of course, the first theoretical point to note is that feature grammars are, in essence, context-free thus allowing more complex languages to be described than FSGs. For example, there does not exist an equivalent finite state grammar for the (context-free) grammar

$$S \rightarrow a\ S\ b$$
$$S \rightarrow a\ b$$

Which describes the language $a^n b^n$ where n is greater than or equal to 1. However if we set a finite limit on n then there does exist a (possibly very large but *finite*) FSM. Thus we could accept $a^n b^n$ only where n is greater than or equal to one but less than some finite number d.

In terms of natural language, an equivalent example is the restriction that you can only have up to n levels of centre embedding within a language. This seems to be no less a restriction on a language than the restrictions you are imposing on that language when you try to write a grammar for it in the first place, irrespective of the grammar formalism.

Practically, there may be other problems in writing a compilation function from feature grammars to finite state grammars. There is of course the problem of the *size* of FSM created, as well as the *time* that is needed to generate it. Both these question were open at the start of our investigation.

Because we hoped that this compilation need only be run occasionally and that the high level formalism could be debugged using a conventional chart parser, we feel that compilation time can be up to 12 hours without any problem. As for the resulting FSM, it seems that with today's workstations up to 100,000 transitions might be acceptable. But the question still remained: how big a feature grammar can be compiled within these constraints?

2 The Initial Structures

The grammarian first writes a grammar in the high level GPSG-like notation which is then translated to FBF. This translation is relatively simple, it merely converts the user-written form into an internal Lisp form, expanding aliases, feature passing conventions etc. The FBF formalism seemed like a good input to the FSM compiler as it is well defined and quite fixed within our system.

FBF is effectively an *assembly language* for feature grammars. It is much in the spirit of PATR-II ([Shieber86]) but differs in that it uses term unification rather than graph unification as its basic operation, though that distinction if not important here.

The inputs to the FSM compilation are:

- a distinguished category
- a set of feature grammar rules.
- a set of lexical entries

The lexicon consists of a mapping of atomic symbols to categories. In actual fact within our system these atoms are not words but preterminals. It is these preterminals which label the arcs of the generated finite state machine.

It should be added that FBF is not a prerequisite for this technique. Any feature grammar notation would be suitable (though the code would have to be changed).

3 The Compilation Process

The compilation takes place in five stages:

- conversion into internal structures for fast access. This consists of the conversion of categories in the grammar and lexicon into an internal form, consisting of an atomic type and a list of feature values, thus unification can be done more efficiently. Also, two indexes are created – one for the grammar and one for the lexicon – both indexed by category type, allowing efficient access to them.
- conversion of the grammar to a non-deterministic finite state machine. This is the main part — see the the next section for details about this.
- removal of error states from the non-deterministic finite state machine. States can be created which cannot lead to final states, these are removed as well as all arcs pointing to them.
- determinising. Standard determinising of the finite state machine (as described in [Hopcroft79, p. 22])
- analysis to produce statistics, this finds the size, average and maximum branching rates.

4 The Actual Conversion

The conversion is done by building "agenda states" on an agenda and processing them until the agenda is empty. An "agenda state" consists of the following:

- A depth — the number of rewrites that are required to get the first category in the remainder
- a list of remaining categories — these are the categories (preterminal or otherwise) that have yet to be found before the end of a sentence is reached
- A set of variable bindings
- a state in the non-determinised machine

The basic loop starts with an initial "agenda state" with the following settings:

- a depth of 0
- a list containing only the distinguished category
- a set of empty bindings
- the initial state of the (non-deterministic) FSM

The processing is as follows:

Take an "agenda state" from the agenda and take its remainder. Rewrite the first category in the remainder, using the grammar, in all ways, recursively until either the depth limit is met or a lexical category is found (i.e. a category which is in the lexicon).

Rewrites are made by replacing the first category with the right hand side of a grammar rule, whose left hand side unifies with the first category. Thus a rewrite changes the first category, increments the depth, and possibly binds some variables[1]. Also, in addition to the right hand side, a special "end-subrule" marker (*em*) is added so that we can tell when to decrease the depth count. For example: S may rewrite as follows[2]

$$\begin{array}{ll} \text{S} & \Longrightarrow \\ \text{NP VP } \mathit{em} & \Longrightarrow \\ \text{Det Noun } \mathit{em} \text{ VP } \mathit{em} & \end{array}$$

Then for each rewrite, check the lexicon and find all entries that can match the first category. Add a transition to the state in the current "agenda state", labelled with that lexical item, to a new state, in the non-deterministic FSM.

This may be a (truly) new state or an already existing state. Each state in the non-deterministic FSM has a "state descriptor" which symbolizes which categories from this state would lead to a final state. The state descriptor is constructed by taking the remaining categories list and dereferencing the variables, removing the "end-subrules" markers, and replacing any unbound variables with a unique atom name representing a variable[3]. Thus no unification is required in searching, a simple Lisp `EQUAL` is adequate (actually a more complex indexing system is used).

When looking for a "new state", the state descriptor of the required state is constructed and a (rather large) index is checked to find if such a state already exists, if so the new transition points to the state related to that "state descriptor".

If a truly new state is required a corresponding new "agenda state" is created. The "cdr" of the remaining categories list is taken: that is the next category is found in the remainder list, any "end-subrule" markers which precede it are removed and the depth is decremented.

5 An Example

For the sake of brevity the example grammar used here is only a standard context-free grammar with atomic categories rather than a feature grammar. Thus we use `EQUAL` as our test operator, while with feature grammars we would use unification, and record any resulting bindings.

Given the following grammar:

S → NP VP
NP → Det Noun
NP → PropNoun
VP → Verb NP

And a lexicon as follows:

the → Det
boy → Noun
Hanako → PropNoun
saw → Verb

[1] Because variables are "uniquified" at each instantiation of a rule the correct bindings are ensured throughout the conversion.

[2] Atomic symbols are used here as categories for brevity

[3] This is actually over-general, as variables which have been bound to one variable, and hence co-referenced, but not (yet) bound to a literal, will still be treated as distinct by this method.

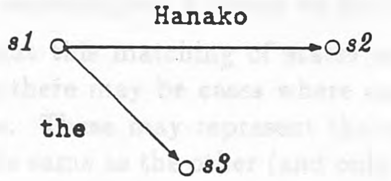
Let us go through some of the steps. The first stage is an agenda state of the form[4]:

```
depth: 0   remainder: (S)      state: s1
```

There are two possible rewrites

```
depth: 2   remainder: (PropNoun em VP em)
depth: 2   remainder: (Det Noun em VP em)
```

We then add transitions from *s1* to two new states labelled with "the" and "Hanako" like so:



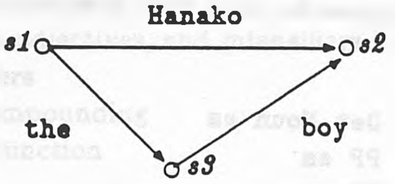
We then create two new "agenda states" and add them to the agenda

```
depth: 1   remainder: (VP em)            state: s2
depth: 2   remainder: (Noun em VP em) state: s3
```

Now consider the second one. As Noun is already a lexical category, there is no need to rewrite it. We can add a transition from *s3* to a "new state". To find the "state descriptor" of this "new state" we first remove the first category, and then remove any "em" markers, decrementing the depth accordingly. The resulting remainder and depth is

```
depth: 1   remainder: (VP em)
```

Then we create the "state descriptor" from this new remainder, which will give simply (VP), which is the same as the descriptor of *s2*. Thus this new arc labelled with "boy" will go from *s3* to *s2*. Like this:



Thus we only need one occurrence of the VP despite there being two "types" of NP. Of course in larger grammars, we would probably have two parts of the FSM representing VPs, one dealing with singular subject VPs, and the other with plural VPs (actually there may be more depending on the distinctions made in the grammar). This of course means building a large FSM, but that is, in part, the object of this exercise, trading space (i.e. the size of the FSM) with time (reducing the number of unifications required).

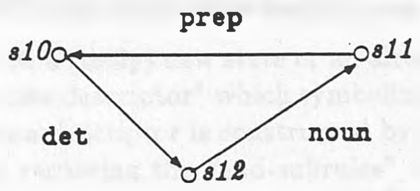
[4]no bindings are shown as we dealing with a simple atomic CFG

5.1 Getting Loops from Recursion

Consider the following three rules in isolation:

NP → NP PP
NP → Det Noun
PP → Prep NP

If we can collapse recursion into loops, we can represent these three rules by the very simple FSM



We have two problems to deal with here, left recursion, and right recursion. Left recursion is a lot harder to deal with than right recursion. With left recursion, during the rewrite stage we must check to see if we have already used the rule during this rewrite. If we detect this, we construct the new rewrite in a different way.

Instead of replacing the first category with its expansion, we find: what the non-recursive rewrites are; and the rules which introduce the rewrites. For the sake of description we will consider the case where there is only one non-recursive and one recursive rule, as in this example. Thus we have a "non-recursive rewrite" (`Det Noun em`) and a "non-recursive part of a recursive rule" (`PP em` — from the rule NP → NP PP). We then construct a new remainder (for an "agenda state")

```
(    "non-recursive rewrite"
     ( "non-recursive part of a recursive rule" )
     "top remainder"
)
```

When there are multiple occurrences of the first two parts we must form remainders for the cross-product of them. However in our example, suppose we start with the remainder (`NP VP em`), the three parts are

non-recursive rewrite	`Det Noun em`
non-recursive part of recursive rule	`PP em`
top remainder	`VP em`

Thus the complete rewrite is

```
(Det Noun em (PP em) VP em)
```

The "looping part" in brackets, (`PP em`), does not appear in the "state descriptor" and hence this state is treated the same as (`Det Noun em VP em`). The important feature is this: when the categories before the bracketed part have been dealt with and we have remainder of the form (`(PP em) VP em`), we construct *two* new "agenda states", one with remainder (`PP em VP em`) and the other (`VP em`).

This of course is too general as we are now treating the states with the "state descriptors" (Det Noun em VP em) and (Det Noun em (PP em) VP em) as the same, which may not be true. What we need to do is ensure that after the "looping part" we can get back to the same state which did not follow that part. (Assuming no variable bindings have made that join inappropriate).

Right recursion is a lot easier, having generated a state with the remainder (PP em VP em), we rewrite to (prep NP em em VP em). After removing the prep we will be left with a remainder of (NP em em VP em). Because we ignore "depth" and the "end-subrule" markers in generating "state descriptors", the "state descriptor" of (NP em em VP em) is the same as that of (NP em VP em), despite the different depths and number of "end-subrule" markers. Thus after the preposition we can return to the point in the FSM where we require an NP followed by a VP.

It is true that this NP is "different" from the other. One is an NP within a PP the other is the subject of a sentence, but because we are merely doing *recognition* this is all we need.

Notice that this matching of states by a state descriptor is not *guaranteed* to merge similar states, since there may be cases where one remainder does not start with a lexical category and another does. These may represent the same state if the first category can be written to the a remainder the same as the other (and only that remainder). This means that we will not guarantee the most minimal FSM during compilation, but will collapse many states.

6 Complexity Results

It is not surprising that this is possible. The really interesting part is whether useful grammars can be converted to reasonably sized finite state machines in reasonable time.

The code is written in Common Lisp and runs on a number of different machines. It had to be re-written a number of times to get the performance we wished. It has been true that the spectre of unacceptable computational complexity has been just round the corner a number of times but so far we have kept it at bay.

Describing the size of a grammar is difficult, but to give some idea of the feasibility of this method of running feature grammars, one of our current grammars, which consists of 31 GPSG-like rules, describes declarative sentences with the following features:

transitive and intransitive verbs
copula sentences
multiple adjectives, and intensifiers in NPs
quantifiers
noun compounding
NP conjunction

The NP conjunction was quite a drastic addition, which increased the size of the resulting FSM by an order of magnitude.

The grammar described above can be converted to a non-deterministic FSM of about 9,000 states[5] in around one hour on a Sun 4/260 with 32Megabytes of memory. We feel this is well within our 12 hour/ 100,000 state limit. But although this grammar is bigger than many "toy grammars", it is still rather small and not really large enough to cover a significant proportion of the domain we wish to cover.

[5]without conjunction the FSM is less than 1,000 states

It should be added that we have had problems in determinising some of the generated FSMs. Though the conversion stage has taken around an hour, determinising has failed to finish in 75 hours, producing a much larger FSM than its non-determinised equivalent. This does suggest that perhaps we should only produce non-deterministic FSMs as output.

7 Comment

So the basic question is, "is it worth it?"

The major loss in moving from a chart parser using a feature grammar to a finite state machine is the loss of a parse tree. One of the reasons for adding a sentence grammar to a speech recogniser is to enable (eventually) some form of semantic analysis. There is an argument that because vast numbers of hypotheses have to be dealt with by a speech recogniser, perhaps running with a FSM as a grammar would be effective during recognition, and that post-processing of the few sentences found could be done with a chart parser.

Then again perhaps speed is not the real thing to worry about, a fast chart parser and unification algorithm might work almost as well (especially if machines are doubling in speed every year).

It is true that the technique is practically limited, no matter how fast machines get there will always be grammars which cannot be converted in reasonable time and/or produce finite state machines with too many states.

And as noted before, the algorithm does produce a FSM which accepts the subset of the language described by the feature grammar where the "depth" less than the given limit, *plus* some extra sentences not originally accepted by the feature grammar. These extras are because of two faults in the conversion algorithm, namely in joining the end of left recursive rules and not constraining where variables have been co-indexed by another variable (and not an atomic value).

This over-generation seems to encourage the idea of using a real chart parser to post-process and correct the sentences accepted by the FSM (though the types of grammars which cause these problems are not common in our domain, so far).

Within our working framework (speech *recognition*) this method does produce useful results. As [illegible] can still allow our grammarians to write a high level description, but still have a fast implementation of their grammar. So in spite of the short comings we will probably use this technique for the foreseeable future.

8 Acknowledgements

This work was supported by the UK Information Engineering Directorate/ Science and Engineering Research Council as part of the IED/SERC Large Scale Integrated Speech Technology Demonstrator Project (SERC grants D/29604, D/29611, D/29628, F/10309, F/10316), in which Marconi Speech and Information Systems are the industrial partner.

References

[Gazdar85] G. Gazdar, E Klein, F. Pullum and I. Sag *Generalized Phrase Structure Grammar* Blackwell, Oxford, 1985

[Hopcroft79] J. Hopcroft and J. Ullman *An Introduction to Automata Theory, Languages and Computation* Addison Wesley, Reading 1979.

[Shieber86] S. Shieber *An Introduction to Unification-based Approaches to Grammar* CSLI Lecture notes Number 4, 1986

[Thompson89] H. Thompson *FBF - A Micro-formalism for grammar: Syntax, Semantics and Metatheory* Dept of AI, University of Edinburgh, forthcoming

An Efficient Enumeration Algorithm of Parses for Ambiguous Context-Free Languages

Nariyoshi YAMAI[†], Tadashi SEKO[†], Noboru KUBO[††] and Toru KAWATA[††]

† Department of Information Engineering, Nara National College of Technology, Yamatokoriyama, Nara 639-11, Japan

†† Computer Systems Laboratories, Corporate Research and Development Group. SHARP Corporation, Tenri, Nara 632, Japan

Abstract

An efficient algorithm that enumerates parses of ambiguous context-free languages is described, and its time and space complexities are discussed.

When context-free parsers are used for natural language parsing, pattern recognition, and so forth, there may be a great number of parses for a sentence. One common strategy for efficient enumeration of parses is to assign an appropriate weight to each production, and to enumerate parses in the order of the total weight of all applied production. However, the existing algorithms taking this strategy can be applied only to the problems of limited areas such as regular languages; in the other areas only inefficient exhaustive searches are known.

In this paper, we first introduce a hierarchical graph suitable for enumeration. Using this graph, enumeration of parses in the order of acceptablity is equivalent to finding paths of this graph in the order of length. Then, we present an efficient enumetation algorithm with this graph, which can be applied to arbitrary context-free grammars. For enumeration of k parses in the order of the total weight of all applied productions, the time and space complexities of our algorithm are $O(n^3 + kn^2)$ and $O(n^3 + kn)$, respectively.

1 Introduction

Context-free parsers are commonly used for natural language parsing, pattern recognition, and so forth. In these applications, there may be a great number of parses (or derivations) for a sentence, only a few of which would be needed in later processes. Therefore, we look up only a few promising parses and do not make an inefficient exhaustive search of parses. In order to find a few promising parses efficiently, we often take a strategy that an appropriate weight is assigned to each production and parses are looked up in the order of the total weight of all applied productions. If the assigned weight is selected carefully to have strong correlation to whether a parse is accepted or not, looking up parses in the order of the total weight is equivalent to enumeration of parses in the order of acceptability. For example, in the punctuation problem of Japanese sentences, the number of the phrases of the sentence is known to be an excellent candidate for the weight of parses. However, the algorithms proposed so far that took this strategy are applied only to the problems of the limited areas such as regular languages, and they are not applied to general context-free languages.

In this paper, we present an efficient enumeration algorithm based on this strategy, which can be applied to general context-free grammars. We introduce a data structure suitable for enumeration of parses named a *parse graph*, and present how to construct a parse graph in section 3. With a parse graph, a path between two special vertex, some of whose arcs are replaced iteratively by the path denoted by their labels, represents a right parse of the parsed sentence. Because the length of paths represents the total weight of all applied productions for parses, enumeration of parses in the order of the total weight of all applied productions is equivalent to finding paths on the parse graph in the order of length. In section 4, we show the outline of how to enumerate the parses of the ambiguous sentence in the order of their weight, using the parse graph. We also discuss the time and space complexities of the algorithm in that section.

2 Context-Free Parsing Algorithm

Several general context-free parsing algorithms have been proposed so far, namely Cocke–Younger–Kasami algorithm[2, 3], Earley's algorithm[4], Valiant's algorithm[5], Graham–Harrison–Ruzzo algorithm[6, 8], and so forth. The features of these algorithms are the following. Cocke–Younger–Kasami algorithm (CYK algorithm for short) is a kind of the bottom up parsing algorithms, and has $O(n^3)$ time complexity, where n is the length of the sentence. In this algorithm, the grammar is required to be written in Chomsky normal form. Earley's algorithm is a kind of the top down parsing algorithms, and has $O(n^3)$ time complexity. By contrast with CYK algorithm, no special production form is required in Earley's algorithm. Valiant's algorithm and Graham–Harrison–Ruzzo algorithm (GHR algorithm for short) are the modified versions of CYK algorithm and Earley's algorithm, respectively. Both of them use the technique of matrix multiplication in order to reduce the time complexity, The time complexity of Valiant's algorithm is $O(n^{2.81})$ and that of GHR algorithm is $O(n^3/\log n)$. However, in both algorithms, the overhead for matrix multiplication is so large that these algorithms don't seem suitable for the practical use.

In this paper, we adopt Earley's algorithm as the base of our algorithm because of the following two reasons:

(1) No special production form is required.

(2) Earley's algorithm seems more suitable than Valiant's algorithm and GHR algorithm because the overhead of these two algorithms is quite large.

Let $G = (V_N, V_T, P, S)$ be a grammar, where V_N is the set of nonterminal symbols, V_T is the set of terminal symbols, P is the set of productions, and $S \in V_N$ is the start symbol. In Earley's algorithm, the *item lists* $I_0, I_1, \ldots, I_n, I_{n+1}$ are created, where n is the length of the parsed sentence. Each item list consists of several *items* $[A \rightarrow \alpha \cdot \beta\ (p), f]$, where $A \rightarrow \alpha\beta \in P$, p is the index number of the production, "$\cdot$" is the meta symbol that shows how much of the right side of the production has been recognized so far, and f is an integer which denotes the position in the input string at which we began to look for that instance of the production. The set of item lists $\{I_0, I_1, \ldots, I_n, I_{n+1}\}$ is called the *parse list*.

As for the time and space complexities for Earley's algorithm, the following are known[1].

(e–1) The time and space complexities for parsing a sentence by Earley's algorithm are $O(n^3)$ and $O(n^2)$, respectively, where n is the length of the parsed sentence.

(e–2) The time complexity for deriving a parse from the parse list is $O(n^2)$, where n is the length of the parsed sentence.

3 Parse Graphs

3.1 The features of parse graphs

The parse graph is a directed graph which consists of several connected components. Each connected component is called a *layer* of the parse graph. Each layer is an acyclic graph that has only one source, and it corresponds to either a nonterminal symbol or an integer. An layer corresponding to a nonterminal symbol has only one sink. With this graph, we can extract parses more efficiently than with a parse list of Earley's algorithm. As shown in the next section, a path between two special vertex, some of whose arcs are replaced iteratively by the path denoted by their labels, represents a right parse of the parsed sentence.

In the remainder of this paper, we use the following notations.

$L(f)$	The layer corresponding to an integer f.
$L(A)$	The layer corresponding to a nonterminal symbol A.
$L(v)$	The layer containing a vertex v.
$L(e)$	The layer containing an arc e.
$v_s(X)$	The source of the layer $L(X)$, where X is either an integer, a nonterminal symbol, a vertex, or an arc.
$v_t(A)$	The sink of the layer $L(A)$, where A is a nonterminal symbol. Note that the layer corresponding to a nonterminal symbol has only one sink.

In the parse graph, each arc has one of the following labels.

(1) An index number of the production p, which denotes the derivation by $A \rightarrow \alpha\ (p)$.

(2) A nonterminal symbol A, which denotes the derivation $A \overset{+}{\Rightarrow} \epsilon$.

(3) The index of a vertex v, which denotes the path from $v_s(v)$ to v.

When we describe the arc $e = (u, v)$ with its label of each kind, we use the notations $e(p)$, $e\langle A\rangle$, $e[v]$, or the alternative notations $(u, v; (p))$, $(u, v; \langle A\rangle)$, $(u, v; [v])$, respectively.

Instead of an item of the form $[A \rightarrow \alpha\cdot\beta\ (p), f]$ in Earley's algorithm, we use the triplet $[A \rightarrow \alpha\cdot\beta\ (p), f, v]$ as an item of our algorithm for constructing a parse graph, where v is the index of a vertex.

For example, we parse the sentence xx of the grammar shown in Figure 1. The parse list and the parse graph generated from this sentence are shown in Figure 2 and Figure 3, respectively.

In Figure 3, the label "(2)" of the arc from vertex #8 to vertex #9 indicates the derivation by $S \rightarrow SJ\ (2)$, the label "$\langle S\rangle$" of the arc from vertex #0 to vertex #1 indicates the derivation $S \overset{+}{\Rightarrow} \epsilon$, and the label "[7]" of the arc from vertex #2 to vertex #8 indicates the paths from vertex #0 to vertex #7.

3.2 An algorithm for constructing a parse graph

Our algorithm for constructing a parse graph is based on Earley's algorithm. In Earley's algorithm, one of three operations is performed on each item, depending on its form, to add more items to the item lists. In our algorithm, these operations not only add more items to item lists but also add new vertices and arcs to the parse graph, shown as follows.

$$\begin{array}{ll} S \rightarrow \epsilon & (1) \\ S \rightarrow SJ & (2) \\ J \rightarrow F & (3) \\ J \rightarrow I & (4) \\ F \rightarrow x & (5) \\ I \rightarrow x & (6) \end{array}$$

Figure 1: An ambiguous context-free grammar

I_0 :

[$S' \rightarrow \cdot S\$$	(0),	0,	0]
[$S \rightarrow \cdot$	(1),	0,	0]
[$S \rightarrow \cdot SJ$	(0),	0,	0]
[$S' \rightarrow S \cdot \$$	(0),	0,	1]
[$S \rightarrow S \cdot J$	(2),	0,	2]
[$J \rightarrow \cdot F$	(3),	0,	0]
[$J \rightarrow \cdot I$	(4),	0,	0]
[$F \rightarrow \cdot x$	(5),	0,	0]
[$I \rightarrow \cdot x$	(6),	0,	0]

I_1 :

[$F \rightarrow x \cdot$	(5),	0,	0]
[$I \rightarrow x \cdot$	(6),	0,	0]
[$J \rightarrow F \cdot$	(3),	0,	4]
[$J \rightarrow I \cdot$	(4),	0,	6]
[$S \rightarrow SJ \cdot$	(2),	0,	8]
[$S' \rightarrow S \cdot \$$	(0),	0,	10]
[$S \rightarrow S \cdot J$	(2),	0,	11]
[$J \rightarrow \cdot F$	(3),	1,	12]
[$J \rightarrow \cdot I$	(4),	1,	12]
[$F \rightarrow \cdot x$	(5),	1,	12]
[$I \rightarrow \cdot x$	(6),	1,	12]

I_2 :

[$F \rightarrow x \cdot$	(5),	1,	12]
[$I \rightarrow x \cdot$	(6),	1,	12]
[$J \rightarrow F \cdot$	(3),	1,	14]
[$J \rightarrow I \cdot$	(4),	1,	16]
[$S \rightarrow SJ \cdot$	(2),	0,	18]
[$S' \rightarrow S \cdot \$$	(0),	0,	20]
[$S \rightarrow S \cdot J$	(2),	0,	21]
[$J \rightarrow \cdot F$	(3),	2,	22]
[$J \rightarrow \cdot I$	(4),	2,	22]
[$F \rightarrow \cdot x$	(5),	2,	22]
[$I \rightarrow \cdot x$	(6),	2,	22]

I_3 :

[$S' \rightarrow S\$ \cdot$	(0),	0,	20]

Figure 2: A parse list for the sentence xx of the grammar in Figure 1

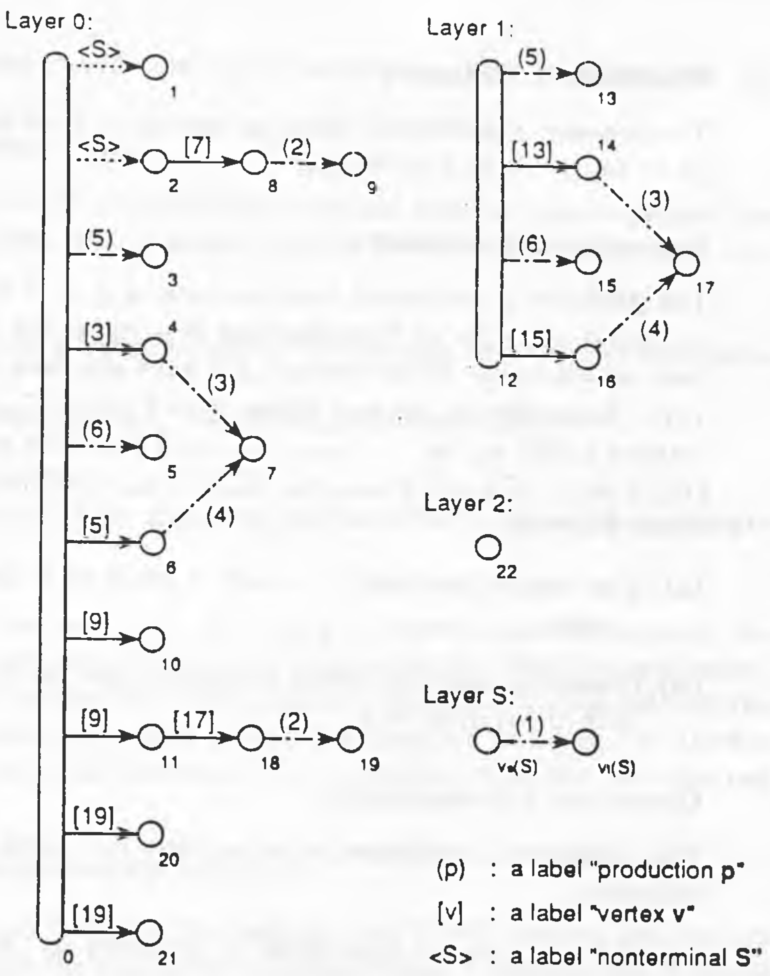


Figure 3: A parse graph for the sentence xx of the grammar in Figure 1

Operation 1 (scanner)

The *scanner* is performed when an item in I_j is of the form $[A \rightarrow \alpha \cdot a_{j+1}\beta\ (p), f, v]$. It puts the item $[A \rightarrow \alpha a_{j+1} \cdot \beta\ (p), f, v]$ to I_{j+1}.

Operation 2 (predictor)

The *predictor* is performed when an item in I_j is of the form $[A \rightarrow \alpha \cdot B\beta\ (p), f, v]$. It adds items $[B \rightarrow \cdot\gamma_k\ (p_k), j, v_s(j)]$ for all B-productions $B \rightarrow \gamma_k\ (p_k)$ to I_j, except in the case where these items have already been added to I_j. If the vertex $v_s(j)$ have not been created yet, the predictor creates $v_s(j)$ to the layer $L(j)$. Especially, in the case where $B \Rightarrow C_1C_2\cdots C_m \overset{*}{\Rightarrow} \epsilon$ and $C_1C_2\cdots C_m \in V_N^*$, the predictor adds the vertices $v_s(B)$, v_1, v_2, $\ldots$, v_{m-1}, v_m, $v_t(B)$, and the arcs $(v_s(B), v_1; \langle C_1\rangle)$, $(v_1, v_2; \langle C_2\rangle)$, $(v_2, v_3; \langle C_3\rangle)$, $\ldots$, $(v_{m-2}, v_{m-1}; \langle C_{m-1}\rangle)$, $(v_{m-1}, v_m; \langle C_m\rangle)$, $(v_m, v_t(B); (p))$ to $L(B)$ if they are not in $L(B)$, and performs one of the following:

(a) If an item of the form $[A \rightarrow \alpha B \cdot \beta\ (p), f, w]$ is already in I_j, then add the arc $(v, w; \langle B\rangle)$ to the parse graph.

(b) Otherwise, add the vertex w and the arc $(v, w; \langle B\rangle)$ to the parse graph, and add the item $[A \rightarrow \alpha B \cdot \beta\ (p), f, w]$ to I_j.

Operation 3 (completer)

The *completer* is performed when an item in I_j is of the form $[A \rightarrow \alpha \cdot\ (p), f, v]$. It performs one of the following:

(a) If $f = j$, then the item would be processed by the predictor. Therefore, the completer does nothing.

(b) If $f \neq j$, and there exists an item of the form $[A \rightarrow \beta \cdot\ (q), f, u]$ ($p \neq q$, $u \neq v$) in I_j, and the arc $(u, w; (q))$ in the parse graph, then add the arc $(v, w; (p))$ to the parse graph.

(c) Otherwise, add a new vertex x and a new arc $(v, x; (p))$ to the parse graph. Furthermore, for all items of the form $[B_k \rightarrow \gamma_k \cdot A\delta_k\ (p_k), f_k, u_k]$ in I_f, perform one of the following:

(c-1) If there exists an item of the form $[B_k \rightarrow \gamma_k A \cdot \delta_k\ (p_k), f_k, v_k]$ in I_j where $u_k \neq v_k$, then add a new arc $(u_k, v_k; [x])$ to the parse graph.

(c-2) Otherwise, add a new vertex v_k and a new arc $(u_k, v_k; [x])$ to the parse graph, and add a new item $[B_k \rightarrow \gamma_k A \cdot \delta_k\ (p_k), f_k, v_k]$ to I_j.

We describe our algorithm for constructing a parse graph as follows:

Algorithm 1. An algorithm for constructing a parse graph

A context-free grammar $G = (V_N, V_T, P, S)$ and a sentence $a_1a_2\cdots a_n$ are given.

[**step 1**] Add the meta symbol "\$" to the tail of the sentence. Add the production $S' \rightarrow S\$\ (0)$ to P. Create the parse graph consisting of $v_s(0)$. Create the item list I_0 consisting of $[S' \rightarrow \cdot S\$\ (0), v_s(0)]$.

[**step 2**] Create the item lists $I_0, I_1, \ldots, I_{n+1}$ in order, by performing the following operations from $k = 1$ to $k = n$.

(1) Perform the predictor or the completer to add items to the item list I_k, until no more items can be added to I_k.

(2) Then, perform the scanner to add items to I_{k+1}.

[step 3] If I_{n+1} has an item of the form $[S' \rightarrow S\$ \cdot \ (0), v]$, then it means that the parser accepts the sentence, and the algorithm terminates. Otherwise, it means that the parser rejects the sentence, and the algorithm terminates.

Note that this algorithm is the same as Earley's algorithm except the portion for constructing a parse graph.

As for the time and space complexities of this algorithm, the following theorem holds.

Theorem 1. The time and space complexities of our algorithm are both $O(n^3)$, where n is the length of the sentence.

(proof) Consider the number of items in the item lists. According to three operations, namely the scanner, the predictor and the completer, each item list does not have items such that their first and second components are the same. Therefore, each item list has $O(n)$ items, because the number of the kinds of the first component is constant, and that of the second component is not more than $n + 2$. Hence, the number of items of the parse list is $O(n^2)$, because the parse list consists of $n + 2$ item lists. Consider the time and space complexities of the operations per item.

(1) As for the scanner, the time and space complexities are both $O(1)$.

(2) As for the predictor, at most $O(|P|)$ items are added to the item list, and $O(|P|)$ vertices and arcs are added to the parse graph, where $|P|$ denotes the number of the productions. Therefore, the time and space complexities are both $O(|P|) = O(1)$.

(3) As for the completer, if the second component of the performed item is f, the completer scans all items in I_f, adds at most $O(n)$ items to the item list, and adds at most $O(n)$ vertices and arcs to the parse graph. Therefore, the time and space complexities are both $O(n)$.

Consequently, the time and space complexities of the operations per item is $O(n)$. Therefore, the time and space complexities of the parse graph construction algorithm are both $O(n^3)$. □

Compared with (e–1) in section 2, the time complexity for constructing a parse graph is the same as Earley's algorithm, but the space complexity is worse because the number of arcs in a parse graph is $O(n^3)$.

4 Enumeration of Parses

4.1 Extracting parses

In order to extract parses from a parse graph, we introduce a *traversal paths* of a parse graph. The notation $\pi(u, v)$ represents traversal paths from u to v.

A *traversal path* from a vertex u to a vertex v is defined as follows provided that $L(u) = L(v)$.

(1) A null sequence is defined as a traversal path if $u = v$.

(2) The sequence of the arcs $e_1 e_2 \cdots e_n$, where $e_i = (u_i, v_i)$, is defined as a traversal path if $u = u_1, v_1 = u_2, v_2 = u_3, \ldots, v_{n-1} = u_n, v_n = v$.

(3) Let $e_1 e_2 \cdots e_n$ be a traversal path from u to v, in which an arc e_i is labeled with a nonterminal symbol A. The sequence of the arc $e_1 \cdots e_{i-1} \pi(v_s(A), v_t(A)) e_{i+1} \cdots e_n$ in which $e_i\langle A \rangle$ is replaced by a traversal path $\pi(v_s(A), v_t(A))$ is defined as a traversal path.

(4) Let $e_1 e_2 \cdots e_n$ be a traversal path from u to v, in which an arc e_i is labeled with the index of a vertex v. The sequence of the arc $e_1 \cdots e_{i-1} \pi(v_s(v), v) e_{i+1} \cdots e_n$ in which $e_i[v]$ is replaced by a traversal path $\pi(v_s(v), v)$ is defined as a traversal path.

Especially, the traversal path that has only the arcs labeled with the index of the production is called a *proper traversal path*. The notation $\pi^*(u, v)$ represents proper traversal paths from u to v. This notation is also used to represent the sequence of the labels of the proper traversal paths.

As for the relationship between proper traversal paths and parses, the following theorem holds.

Theorem 2. If there exist two items $[A \rightarrow \alpha \cdot \beta\gamma\ (p), f, u] \in I_j$, $[A \rightarrow \alpha\beta \cdot \gamma\ (p), f, v] \in I_k$, where $\alpha, \beta, \gamma \in V^*$, the sequence of the labels $\pi^*(u, v)$ is the reverse order of the sequence of the production numbers used for the rightmost derivation $\beta \overset{*}{\underset{rm}{\Rightarrow}} a_{j+1} \cdots a_k$.

(proof) It is easy to prove this theorem by induction on the length of the derivation sequence. □

Let $v_t(0)$ be the third component v of the item $[S' \rightarrow S\$ \cdot\ (0), 0, v] \in I_{n+1}$. According to theorem 2, the sequences of the labels $\pi^*(v_s(0), v_t(0))$ represent the right parses of the parsed sentence. An example of a proper traversal path of the parse graph in Figure 3 is shown in Figure 4, where $v_s(0)$ is vertex #0 and $v_t(0)$ is vertex #20.

A right parse can be extracted from the parse graph by searching a proper traversal path from $v_t(0)$ toward $v_s(0)$. This extraction can be done without backtracking, because each layer has only one source. Therefore, the following theorem holds.

Theorem 3. If the given grammar is cycle-free, the time complexity for extracting a parse is $O(n)$, where n is the length of the sentence.

(proof) If the grammar is cycle-free, the length of the parse is $O(n)$. Therefore, the time complexity is $O(n)$. □

Compared with (e–2) in section 2, the time complexity for extracting a parse is better than Earley's algorithm.

4.2 An algorithm for parse enumeration

Using a parse graph, enumeration of the parses in the order of the total weight is equivalent to enumeration of the proper traversal paths from $v_s(0)$ to $v_t(0)$ in the order of the length. While many researchers have developed the algorithms for finding the k shortest paths[9, 10, 11, 12, 13], we apply one of them developed by Katoh, Ibaraki and Mine[10] to the parse graph recursively. Because of the lack of the space, we explain

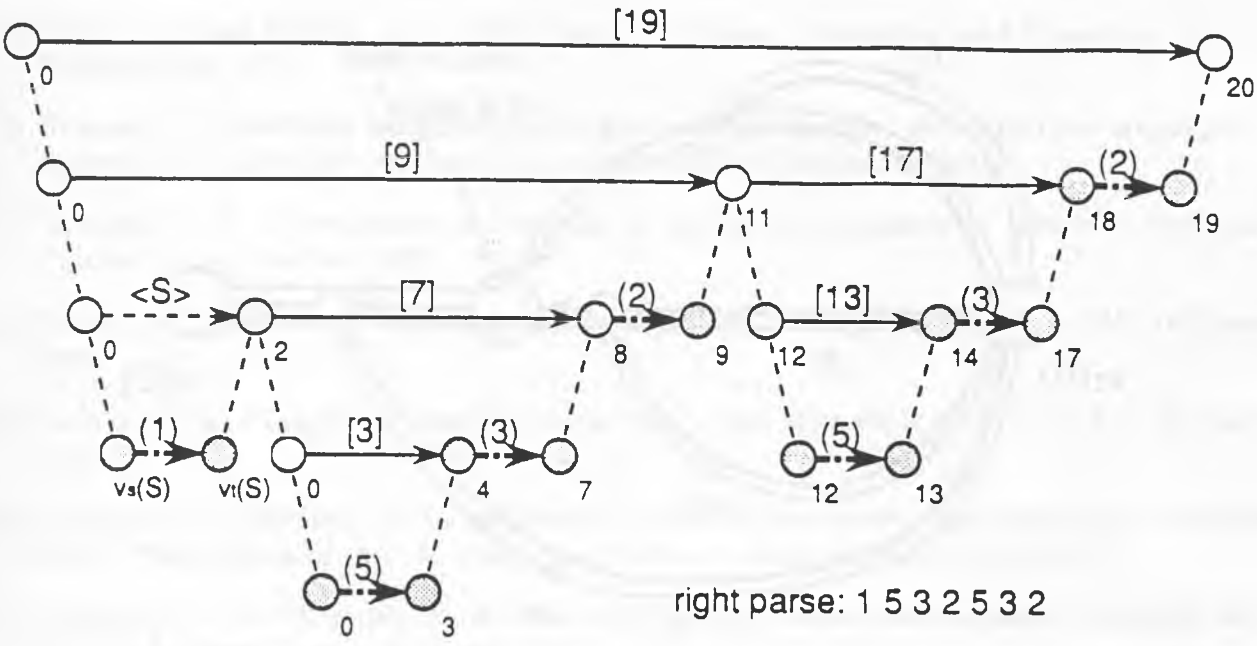


Figure 4: A proper traversal path from vertex #0 to vertex #20

only the outline of the algorithm. The details of the algorithm are described in [14]. In the following discussion, the k-th shortest traversal path from $v_s(0)$ to $v_t(0)$ is referred to as π^k.

First of all, derive the *shortest path tree* for $v_s(0)$, denoted as $T(v_s(0))$, which consists of the arcs of the shortest paths from $v_s(0)$ to all other vertices. The shortest path tree can easily be derived in the algorithm for constructing a parse graph. π^1 can be extracted from $T(v_s(0))$. π^2 consists of the path of $T(v_s(0))$ from $v_s(0)$ to a vertex u, the arc (u, v) where v is one of the vertices on π^1, and the subpath of π^1 from v to $v_t(0)$. Therefore, the number of the candidates of π^2 is the same as the sum of the in-degree of all vertices on the shortest path. As for the parse graph, the length of the shortest path and the in-degree of a vertex are both $O(n)$[14], and hence we can derive π^2 in $O(n^2)$. In order to derive π^3, all paths from $v_s(0)$ to $v_t(0)$ except π^1 and π^2 are divided into three sets as follows (see Figure 5):

(1) The set of paths that join the subpath common to π^1 and π^2. The shortest path in this set is referred to as π_a.

(2) The set of paths that join π^1, and contain the subpath common to π^1 and π^2 as their final subpath. The shortest path in this set is referred to as π_b.

(4) The set of paths that join π^2, and contain the subpath common to π^1 and π^2 as their final subpath. The shortest path in this set is referred to as π_c.

π_a, π_b, and π_c can be derived in the same manner as deriving π^2 in $O(n^2)$, respectively. π^3 is the shortest one of π_a, π_b, and π_c, and the rest of these paths are stored in another set as the candidates of π^4. $\pi^4, \pi^5, \ldots$ are derived by repeating the similar calculation. Therefore, the time and space complexities of the enumeration of the k shortest paths are $O(n^3 + kn^2)$ and $O(n^2 + kn)$, respectively.

In the above discussion, the k shortest paths are derived. However, we can also derive the k longest paths in the same manner.

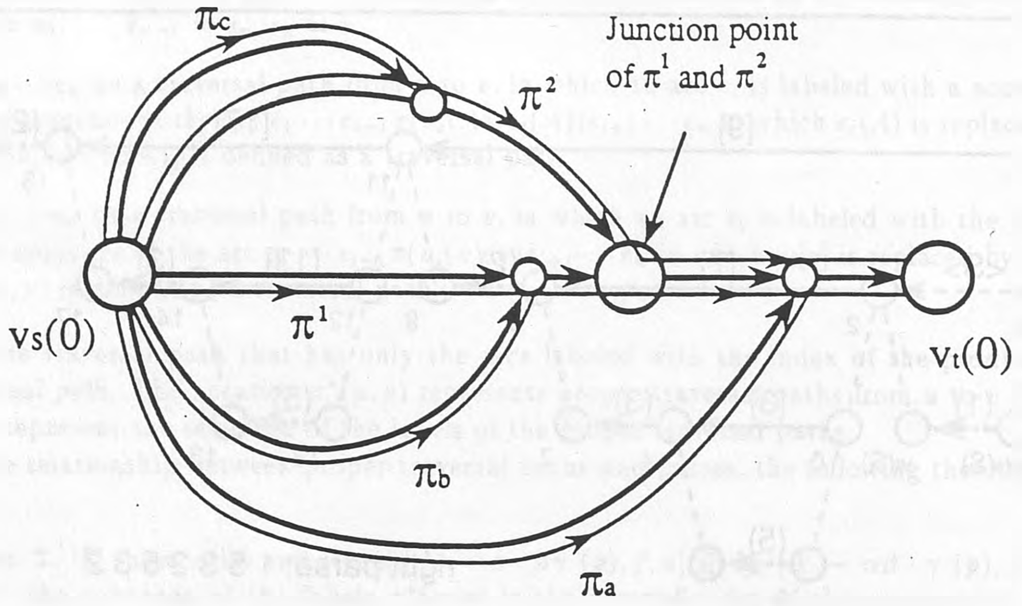


Figure 5: The relation among π^1, π^2, and π^3

Table 1: The time and space complexities of our algorithms (n:the length of the sentence)

Complexity	Construction of parse graph	Enumeration of k parses
Time	$O(n^3)$	$O(n^3 + kn^2)$
Space	$O(n^3)$	$O(n^2 + kn)$

We summarize the time and space complexities of our algorithms in Table 1.

5 Conclusion

In this paper, we have presented an algorithm for the enumeration of the parses in the order of the acceptability. This algorithm can be applied to the general context-free languages. In order to enumerate parses efficiently, we have introduced a data structure suitable for the enumeration called the parse graph. Using a parse graph, we can enumerate k parses in the order of acceptability efficiently in $O(n^3 + kn^2)$.

Acknowledgement

We appreciate Prof. Isao Shirakawa and Prof. Hideo Miyahara of Osaka University for their helpful support. The first author thanks Prof. Toshiro Araki, Dr. Hiroshi Deguchi, Dr. Shinji Shimojo of Osaka University, and Mr. Toshiyuki Masui of SHARP Corporation for their helpful suggestions and comments on this paper.

References

[1] Aho, A. V. and Ullman, J. D., *The Theory of Parsing, Translation, and Compiling, Vol.1:Parsing*, Prentice-Hall, 1972.

[2] Kasami, T., "An efficient recognition and syntax analysis algorithm: for context-free languages", *Science Report*, AF CRL-65-758, Air Force Cambridge Research Laboratory, 1965.

[3] Younger, D. H., "Recognition and parsing of context-free languages in time n^3", *Information and Control*, 10, pp.189–208, 1967.

[4] Earley, J., "An efficient context-free parsing algorithm", *Communication of A.C.M.*, 13-2, pp.94–102, 1970.

[5] Valiant, L. G., "General context-free recognition in less than cubic time", *J.C.S.S.*, 10, pp.308–315, 1975.

[6] Graham, S. L., Harrison, M. A., and Ruzzo, W. L., "On line context-free recognition in less than cubic time", *Proc. 8th Annu. A.C.M. Symp. on Theory of Computing*, pp.112–120, 1976.

[7] Graham, S. L., and Harrison, M. A., "Parsing of general context-free languages", *Advances in Computers*, 14, Academic Press, pp.415–462, 1976.

[8] Graham, S. L., and Harrison, M. A., "An improved context-free recognizer", *A.C.M. Trans. on Programming Languages and Systems*, 2-3, pp.415–462, 1980.

[9] Yen, J. Y., "Finding the K shortest loopless paths in a network", *Management Science*, 17, pp.712–716, 1971.

[10] Katoh, N., Ibaraki, T., and Mine, H., "An efficient algorithm for K shortest simple paths", *Networks*, 12, pp.411–427, 1982.

[11] Fox, B. L., "Data structures and computer science techniques in operations research", *Operations Research*, 26, pp.686–717, 1978.

[12] Denardo, E. V., and Fox, B. L., "Shortest-route methods: 1. reaching, pruning and buckets", *Operations Research*, 27, pp.161–186, 1979.

[13] Lawer, E. L., "A procedure for computing the K best solutions to discrete optimization problems and its application to the shortest path problem", *Management Science*, 18, pp.401–405, 1972.

[14] Yamai, N., "A study for parsing of ambiguous languages using hierarchical graph representation of all derivations", *Master Thesis of Osaka University*, 1986 (in Japanese).

A Morphological Parser for Linguistic Exploration

David Weber
Summer Institute of Linguistics

1. INTRODUCTION

This paper describes AMPLE, a morphological parser (i.e., a program that parses words into morphemes). AMPLE grew out of work in computer assisted dialect adaptation, as described in section 1. It contains no language-specific code, being controlled entirely through external, user-written files, the notations of which were designed for linguists. AMPLE's constructs are linguistic: "allomorph", "morpheme", "conditioning environment", "co-occurrence constraint", etc.

AMPLE's fundamental algorithm is (i) to discover all possible decompositions of a word into allomorphs, and (ii) to eliminate those which fail any conditions, constraints or tests imposed by the user.

This match-and-filter algorithm allows a highly modular approach to morphological parsing. Strong rejection of incorrect analyses is achieved by the combined effect of diverse filters, each expressed simply in a notation appropriate to the phenomena.

AMPLE is a good tool for exploring morphology because of the flexibility resulting from this modularity. And it is usable by computationally naive linguists because its notations are linguistic rather than computational.

2. COMPUTER ASSISTED DIALECT ADAPTATION

Computer assisted dialect adaptation (CADA) attempts to exploit the systematic relationships between closely-related languages to produce drafts of text in target languages from source languages texts. (Initial explorations are described in Weber and Mann, 1979.) CADA works over non-trivial degrees of language difference because, between closely-related languages, most of the differences are systematic. These result from the generalization of regular diachronic changes, thus impacting the language heavily. By contrast, irregular or idiosyncratic changes cannot be generalized, so tend to have a limited impact. So between closely related languages, systematic differences predominate.

Differences are systematic only relative to some analysis. For example, between one dialect of Quechua and another, the character string *ra* might correspond to *ra*, *ri*, *ru* or *rqu*, but the context in which each is appropriate cannot be determined simply by inspecting adjacent character strings (in the source dialect text). However, if one can determine the identity of the morpheme in which *ra* occurs, the differences become systematic: when it is the past tense suffix, then it corresponds to *rqa*; when it is the punctual, it corresponds to *ri* or *ra*, depending on morphological context; when it is the directional 'out', it corresponds to *rqu* or *rqa*, and so forth.

Experience in various language families [Quechua, Tucanoan, Cakchiquel (Mayan), Campa (Arawakan), and the Philippine type] has shown that, for language families with rich morphologies, parsing words into morphemes makes most differences systematic, thereby providing a sufficient analytic base on which to do adaptation.

CADA's analytic engine began as a Quechua-specific morphological parser written in INTERLISP (Weber and Mann, 1979). This parser was re-implemented in C for small systems (Kasper and Weber, 1986a,b). This implementation was subsequently adapted to the Tucanoan language family of Colombia (Reed 1986, 1987), to Campa languages (Arawakan of Peru), and to Philippine languages. Guided by these extensions, a general morphological parser has been developed, called AMPLE (Weber, Black and McConnel, 1988).

AMPLE fits into word-by-word adaptation as indicated in Table 1:

```
                    +-----------------------------+ S
                    | +----------+                | T
 word analyses-->   | | TRANSFER |-->modified     | A
           |        | +----------+ word analyses  | M
A +--------------+  +-------------+      |        | P
M | +----------+ |               | +-----------+  |
P | |          | |               | |           |  |
L | | ANALYSIS | |               | | SYNTHESIS |  |
E | |          | |               | |           |  |
  | +----------+ |               | +-----------+  |
  |  normalized  |               |  normalized    |
  |    words     |               |    words       |
  | +----------+ |               | +-----------+  |
  | |  TEXTIN  | |               | |  TEXTOUT  |  |
  | +----------+ |               | +-----------+  |
  +--------------+               +----------------+
         |                              |
  source dialect text          target dialect text
```

Table 1: THE MAJOR MODULES OF WORD-LEVEL CADA

The following illustrates how each module of Table 1 contributes to adapting from Pachitea Quechua *Aywarkaykargan* 'they were going' to the corresponding Huanca Quechua form, *Liyalkala*:

```
                 Pachitea: Aywarkaykargan
TEXTIN                     |             |
                           aywarkaykarqan
ANALYSIS
                 aywa-   -rka     -yka    -rqa   -n
                *aywa-   -PLIMPF  -IMPF   -PST   -3
                 |           \    /        |      |
                 |            \  /         |      |
TRANSFER         |             X           |      ø
                 |            /  \         |
                 |           /    \        |
                *ri-     -IMPF    -PLIMPF -PST
SYNTHESIS        |         |        |      |
                 li      -ya      -lka    -la

                          liyalkala
TEXTOUT                   |
                   Huanca: Liyalkala
```

In addition to serving as the analytic base for adaptation, AMPLE has been used to automate the glossing of texts (see, e.g., Weber 1987a), to detect spelling errors, and perhaps most significantly, to advance users' understanding of the morphology of various languages.

3. GENERAL AMPLE DESCRIPTION

Various external factors have shaped AMPLE: its constructs, mechanisms and notations must be familiar to linguists; its data files should be useful for other computational and non-computational purposes; it must run effectively on personal computers with small memories; and crucially, it must be able to cope with very diverse phenomena without unduly compromising linguistic integrity.

AMPLE takes text as input. It identifies words and normalizes them according to user-specified rules (e.g., change *b* to *p* before *m*). This allows the internal representation to differ from the external orthography (which might even be a phonetic representation). Each word is subjected to a depth-first, all paths analysis. The text is output as a database--one record per word--with fields for the (possibly ambiguous) analysis, punctuation, white space, format marking, and capitalization information.

AMPLE has various "biases." It is based on the assumption that morphemes exist. It applies directly to concatenative morphology; non-concatenative phenomena usually have to be coerced into concatenative solutions. For example, *took* could be analyzed as *take+PAST* (as suggested by Block 1947). To apply AMPLE to fusional languages generally requires large numbers of fused combinations constrained by declension or conjugation class. Finally, AMPLE takes an item/arrangement rather than an item/process approach (Hockett 1954). There are no "underlying forms" from which surface forms are derived.

AMPLE has main modules: SETUP, TEXTIN and ANALYSIS. SETUP reads files containing information about the language, creating internal structures for TEXTIN and ANALYSIS. Most significantly, SETUP reads one or more dictionaries, creates a trie structure based on allomorphs (character strings) for accessing the information about that allomorph and the morpheme it represents.

TEXTIN identifies the words of the text, putting to one side white space, capitalization information, format markup, and punctuation. User-specified orthographic changes are applied, allowing the internal working representation to differ from the practical orthography of the text.

Analysis parses by (i) discovering all possible sequences of matching allomorphs and (ii) filtering these with the tests that the user writes in various linguistically-oriented constraint languages (as described below). This proceeds bottom-up, left-to-right and exhaustively, i.e., all possible combinations of matching morphemes are discovered, and all which pass the tests are returned in the output. Matching and filtering are integrated so as to abandon false paths as early as possible.

There are two types of test. *Successor tests* apply when a matching allomorph is considered as the next possible morpheme of an analysis. *Final tests*, generally incorporating non-local dependencies, are deferred until an entire decomposition is discovered, one which passes all successor tests.

More specifically, as processing proceeds, a partial analysis is maintained. Whenever a matching allomorph is discovered, successor tests are applied between the partial analysis (usually its last morpheme) and the morpheme under consideration as a successor (for which some allomorph has been matched). For example, in analyzing *rikaykaamaran* 'he was watching me', the following stage would be reached:

```
                          see   IMPFV
                           |      |
    PARTIAL ANALYSIS: rika- -yka:
 POSSIBLE SUCCESSOR:                  -ma 1OBJ
    REMAINING STRING: maran
```

One of the successor tests, to take an example, insures that vocalic length (represented here as a colon) is not followed by syllable-closing suffix (since long vowels cannot occur in a closed syllable).

Successor tests have the advantage of eliminating false paths before they consume more computation, but they can not appeal to following morphemes, since these have not yet been identified. But final tests apply constraints to an entire analysis, so can express forward-referring constraints. For example, a final test might say that a morphophonemically affected unit must be followed (not necessarily adjacently) by a trigger for the process. Also, final tests can impose well-formedness constraints expressed on a particular morpheme; e.g. it might constrain the category of the final morpheme.

4. PHENOMENA

AMPLE can handle a wide variety of phenomena. Units may be prefixes, roots or suffixes, realized, null. or the reduplication of an adjacent segment Morphemes may have multiple allomorphs. AMPLE can handle the reduplication of adjacent segments (although the mechanism may be clumsy in some cases, as discussed below). Infixation is handled, even when obscured by prior or subsequent affixation or reduplication. The compounding of roots is handled (but nothing has been done to treat the compounding of morphologically-complex words).

4.1. Types of unit

AMPLE can deal with roots, suffixes and prefixes (of course!). More interestingly, it can deal with infixes, such as those of Philippine languages, for which an infix may be within a root or within a prefix, and where reduplication may apply after infixation. AMPLE allows compound roots, possibly constrained by the categories of those roots.

AMPLE allows null allomorphs. The occurrence of nulls must be strongly constrained, since they are not constrained by the characters of the word being analyzed. For example, in Napo Quichua, the agentive nominalizer has no phonological realization, due to its lenition and ultimate loss. But there is a strong constraint on its occurrence: it must be at a boundary where an uninflected verb is either word final or followed by suffixes typical of nouns. When adapting to Pastaza Quichua, where the agentive is /h/, it is thus possible to insert /h/ in the appropriate places with considerable accuracy. (For example, *rita* (= ri- 'go' -0 'agentive' -ta 'accusative', meaning 'to the one who goes') can become *ri-j-ta*.

4.2. Phonologically conditioned allomorphy

The occurrence of each allomorph in an analysis may be constrained by its phonological or morphemic environment, either locally or at a distance.

4.2.1. Issues of representation

The practical orthography of the text being analyzed may not be the best representation for doing analysis. (For example, in analyzing Spanish, it might be desirable to eliminate the orthographic alternation between *z* and *c* (cf. *raiz*, *raices*). Likewise, for Latin one might wish to convert *x* into *ks*, so that a morpheme boundary could be posited between the *k* and the *s* (cf. *rex* = /reks/, regis). Orthographic changes such as these can be made by the TEXTIN module.

4.2.2. Conditions on allomorphs

Allomorphs may be restricted by phonological (character string) environment. For example, the following says that *m* may only occur followed by *p*. (\a is the field code for "allomorph".)

```
    \a m / __p
```

Classes of phonological segments can be defined, and then used in constraining environments. For example, the following defines the class of labials and states that *m* must precede one of them:

```
\scl +labial p b f v
\a m / _[+labial]
```

4.2.3. Multiple allomorphs

Any morpheme may have multiple allomorphs. For example, the second person possessive in most Quechua languages have three allomorphs, constrained as follows (where ~[V] _ indicates "not following a vowel):

```
\a niki / ~[V] _ | hatunniki 'your big one'
\a ki   /    i _ | wasiki    'your house'
\a yki  /  [V] _ | umayki    'your head'
```

Reduplication is handled as a special case of multiple allomorphs, where each possibility is enumerated along with the environment in which it could occur (so, e.g., *pa* before *pa...*, *pe* before *pe*, etc. If the reduplicated from is always a precise substring of what precedes or follows, it is possible to state this as a general constraint rather than with each allomorph.

4.3. Morphophonemics

Phenomena involving both altered form (phonology) and morpheme identity present no special challenge because both the character string being analyzed and the posited morphemes are available.

4.3.1. Morpheme environment constraints on allomorphs

It is possible to restrict the occurrence of an allomorph by the identity of a morpheme; e.g., the following says that *an* must be directly followed by the morpheme identified as PQR:

```
\a an +/ _PQR
```

4.3.2. Properties and tests

It is possible to assign properties to allomorphs and morphemes and to use these in a very general constraint language. For example, suppose inherently applicative verbs may never co-occur with the applicative suffix APPL; this can be incorporated by assigning the property "applicative" to applicative verbs and imposing the following test:

```
IF   (current property is applicative)
THEN (FOR_ALL_RIGHT
        NOT (RIGHT morphname is APPL))
```

4.4. Morphotactics

AMPLE has good mechanisms for imposing morphotactic constraints There are three main types: categorial, ordering, and morpheme co-occurrence constraints.

4.4.1. Categorial constraints

Roots are assigned one or more categories, and affixes are assigned one or more category pairs. The left part of a category pair is called the "fromcategory" and corresponds roughly to the affix's "subcategorization frame." The right part is called the "tocategory" and corresponds roughly to its "category".)

In terms of these categories, tests can be imposed which "structure" the verb. To illustrate, consider a language with derivational suffixes (causative, applicative, passive, etc.) and inflectional prefixes. What inflection is permitted and/or required depends on the category after derivation, and "prior" inflection. Likewise, the derivational possibilities depend on the category of the root and any "prior" derivation. Thus, the constraints must propagate first progressively from the root through the suffixes and then regressively through the prefixes to the beginning of the word:

```
          S
         /\
        /  U
       /  /\
      /  /  Z
     /  /  /\
    /  /  X  \
   /  /  /\   \
  /  /  /  \   \
R/S T/U V  W/X Y/Z
pfx pfx root sfx sfx
```

This can be achieved by four tests: (i) for suffixes (whereby V=W and X=Y above):

left tocategory is current fromcategory

(ii) for prefixes (whereby U=R above):

current tocategory is left fromcategory

(iii) to identify the category after derivation with that of the closest prefix (Z=T above):

```
IF (current type is prefix AND right type is root)
THEN (current fromcategory is FINAL tocategory)
```

(iv) to ensure that the category of the whole word (S above) is an acceptable terminal category, we can declare a class of such categories (called "finalcategories") and state:

INITIAL tocategory is member finalcategories

Thus, although AMPLE processes from left to right, it is possible to model the percolation of features from a root through the layers of affixation, to the final resulting category of the word.

4.4.2. Ordering

The use of category along the lines described in the previous section may strongly restrict the order in which affixes occur. However, further ordering constraints may need to be imposed. This can be done by giving each affix a number (not necessarily unique) and imposing a successor test like the following:

```
left orderclass < current orderclass
```

This says that every morpheme's number must be greater than of the preceding morpheme, so insists that the orderclass strictly increase. If "<=" were used instead of "<", the order would be non-decreasing.

The test could also be modified to tolerate morphemes that are not constrained by order, such as Quechua *-lla* 'just'. To do so, we assign *-lla* orderclass 0, and then the following successor test passes it:

```
     (current orderclass = 0)
OR   (left orderclass <= current orderclass)
```

To make ordering constraints apply over one or more "floating" affixes, we give the following final test:

```
IF (     (current orderclass = 0)
     AND (FOR_SOME_LEFT  (LEFT  orderclass ~= 0))
     AND (FOR_SOME_RIGHT (RIGHT orderclass ~= 0)))
THEN (LEFT orderclass <= RIGHT orderclass)
```

4.4.3. Morpheme co-occurrence constraints

AMPLE has a simple but effective constraint language for imposing conditions on the co-occurrence of morphemes. The following, for example, says that PLIMPF can only occur preceding IMPFV:

```
\mcc   PLIMPF  / _ IMPFV
```

The following says that the conditional morpheme CND must be preceded (not necessarily contiguously) by a first, second, or third verbal person suffix (respectively named 1, 2, and 3):

```
\mcc CND / 1 ..._ / 2 ..._ / 3 ..._
```

The first line of the following defines a class of morphemes DIR, and the second says that PLDIR must precede a directional, the reciprocal or the reflexive:

```
\mcl DIR    IN OUT UP DWN
\mcc PLDIR / [DIR] _ / RECIP _ / REF _
```

5. AMPLE AS A TOOL FOR LINGUISTIC EXPLORATION

AMPLE has some features that enhance its usefulness as an exploratory tool:

1. It returns the original word (the \a field), that word's decomposition (\d), and the analysis (\a); for example, the following would be returned for *rirkansapanashi* 'they now went (it is reported)':

```
\a < V1 go > PST 3 PLUR NOW REPORT
\d ri-rka-n-sapa-na-shi
\w rirkansapanashi
```

2. AMPLE reports all analytic failures, indicating how far into the word it was able to proceed and whether or not it matched a root. This often provides a sufficient clue to why the word failed to be analyzed. For example, the following report (for Quechua) makes it clear that (i) the root *fes* (*hwes* after orthography changes) is not available as a root, and (ii) there is an incompatibility between the suffixes *-ri* and *-ma:*:

```
        Root Failure: hwesqa   [ | fesqa ]
    Analysis Failure: roqorimaachun  [ roqori | ma:chun ]
```

3. AMPLE reports on the effectiveness of each test: for both the user-defined and built-in tests, it reports how many times each test was applied (in the order of application) and how many analyses were filtered out by the test:

```
      CATEGORY_ST called 10936 times, failed 7436.
         ORDER_ST called  3500 times, failed 392.
   FORESHORTEN_ST called  3108 times, failed 36.
       MLOWERS_ST called  3072 times, failed 2.
```

4. The user can control which tests are applied and the order of their application. This makes it possible to see the effectiveness of each, and their joint effect.

5. Ambiguity levels are reported as follows:

```
     2 words with  0 analyses.
   620 words with  1 analysis.
    73 words with  2 analyses.
     2 words with  3 analyses.
     3 words with  4 analyses.
```

6. It is possible to trace AMPLE's parsing activity. For example, the following is the first part of the trace for the Quechua word *nimaran*:

```
Parsing nimaran
root: ni, *ni V2
sfx: ma, 10, V2/V1, order: 70, ullong Mlowers, fshrtnd
  sfx: ra, PST, V1/V1, order: 80, foreshortens
    sfx: n, 3P, NO/NO, order: 140   / [V]__
            Suffix test CATEGORY_ST failed.
    sfx: n, 3P, R1/RO, order: 140   / [V]__
            Suffix test CATEGORY_ST failed.
    sfx: n, 3P, N1/NO, order: 140   / [V]__
            Suffix test CATEGORY_ST failed.
    sfx: n, 3, V1/VO, order: 120, foreshortens
            No more suffixes found.
            End of word found; checking final tests
                 Analysis string: < V2 *ni > 10 PST 3
                 Decomposition:   ni-ma-ra-n
```

After achieving this analysis, AMPLE continues considering other possibilities.

A future version of AMPLE will allow selectivity in tracing, more information in the analysis (e.g., the category pairs used in an analysis), and quantifying the contribution of specific morphemes, tests, etc. to analysis.

6. CONCLUDING REFLECTIONS

AMPLE's match-and-filter algorithm permits a highly modular approach to morphological parsing. Strong rejection of incorrect analyses can be achieved by the combined effect of diverse filters, each of which may be quite simple. Direct reporting of these linguistic constraints is possible because they are not compiled into some inaccessible form. And this algorithm has proven to be reasonably efficient.

Our success with the match-and-filter algorithm suggests that morphology has a modular organization. That is, the organization of morphology may resemble the Chomskian approach to syntax, where diverse principles or theories, here expressed as filters, jointly but modularly define acceptability.

Each filter is expressed simply in a notation appropriate to the phenomena and familiar to the users, in this case linguists. This makes it quite straight forward for linguists to set up a morphological parser for a language. Experience has repeatedly shown that doing so leads the user to new insights into the morphology. Because there are various constraint languages and mechanisms, AMPLE can be used to model various conceptions of the morphology, and to quickly test these against large amounts of data.

The modularity afforded by the match-and-filter approach also makes AMPLE very extensible: as other constraint languages are discovered (and notations developed) they can be integrated into AMPLE. For example, we are considering an alternative (or complement) to the category system that would allow categories to be defined as sets of features, incorporating percolation, redundancy rules and feature addition rules; see in Weber 1987b.

We expect AMPLE to be useful in conjunction with various syntactic parsers. In one experiment, a unification-based parser (adapted from an early version of PATR-II) parses sentences (or sentence fragments) using AMPLE output. The morpheme dictionaries, are read once by AMPLE for the morphological information and again by the syntactic parser for the syntactic parser.

We hope that in the next few years AMPLE will be applied to a much wider range of languages.

current address: David Weber
6004 Stanton Ave. A-15
Pittsburgh, PA 15206

REFERENCES

Block, Bernard. 1947. "English verb inflection." **Language**. 23:399-418.

Hockett, Charles. 1954. "Two models of grammatical description." **Word**. 10:210-231.

Kasper, Robert and D. Weber. 1986a (written in 1982). **User's Reference Manual for the C Quechua Adaptation Program.** Occasional Publications in Academic Computing No. 8. Dallas: Summer Institute of Linguistics.

__________ and __________. 1986b (written in 1982). **Programmers Reference Manual for the C Quechua Adaptation Program.** Occasional Publications in Academic Computing No. 9. Dallas: Summer Institute of Linguistics.

Reed, Robert. 1986. **Computer Aided Dialect Adaptation: The Tucanoan Experiment.** PhD dissertation in linguistics. University of Texas at Arlington.

__________. 1987. **The Implementation of a System for Computer Aided Dialect Adaptation.** MA thesis in computer science. University of Texas at Arlington.

Weber, David (editor). 1987a. **Juan del Oso.** Serie Linguistica Peruana No. 26. Yarinacocha: Instituto Linguistico de Verano.

__________. 1987b. "Sobre la morfologia quechua." **Estudios Quechua.** Serie Linguistica Peruana No. 27. Yarinacocha: Instituto Linguistico de Verano.

__________ and William Mann. 1981. "Prospects for Computer Assisted Dialect Adaptation." **AJCL.** 7:165-177.

__________, H.A. Black and S.R. McConnel. 1988. **AMPLE: A Tool for Exploring Morphology.** Occasional Publications in Academic Computing No. 12. Dallas: Summer Institute of Linguistics. (available from the ILC Academic Book Center; 7500 W. Camp Wisdom Rd.; Dallas TX 75236).

THE PARALLEL EXPERT PARSER: A MEANING-ORIENTED, LEXICALLY-GUIDED, PARALLEL-INTERACTIVE MODEL OF NATURAL LANGUAGE UNDERSTANDING

G. ADRIAENS
Siemens NLP Research
& Katholieke Universiteit Leuven
M. Theresiastraat 21
B-3000 Leuven, Belgium
+32 16 285091
(siegeert@kulcs.uucp or
siegeert@blekul60.bitnet or
siegeert@cs.kuleuven.ac.be)

Abstract

The Parallel Expert Parser (PEP) is a natural language analysis model belonging to the interactive model paradigm that stresses the parallel interaction of relatively small distributed knowledge components to arrive at the meaning of a fragment of text. It borrows the idea of words as basic dynamic entities triggering a set of interactive processes from the Word Expert Parser (Small 1980), but tries to improve on the clarity of interactive processes and on the organization of lexically-distributed knowledge. As of now, especially the procedural aspects have received attention: instead of having wild-running uncontrollable interactions, PEP restricts the interactions to explicit communications on a structured blackboard; the communication protocols are a compromise between maximum parallelism and controllability. At the same time, it is no longer just words that trigger processes; words create larger units (constituents), that are in turn interacting entities on a higher level. Lexical experts contribute their associated knowledge, create higher-level experts, and die away. The linguists define the levels to be considered, and write expert processes in a language that tries to hide the procedural aspects of the parallel-interactive model from them. Problems include the possibility of deadlock situations when processes wait infinitely for each other, the way to efficiently pursue different alternatives (as of now, the system just uses don't-care determinism), and testing whether the protocols allow linguists to fully express their needs. PEP has been implemented in Flat Concurrent Prolog, using the Logix programming environment. Current research is oriented more towards the problem of distributed knowledge representation. Abstractions and generalizations across lexical experts could be made using principles from object-oriented programming (introducing generic, prototypical experts; cp. Hahn 1987). Thoughts also go in the direction of an integration of the coarse-grained parallelism with knowledge representation in a fine-grained parallel (connectionist) way.

1. Introduction

In the course of the last decade, interest in parallel machines and applications has steadily been growing in the different disciplines that deal with natural language understanding (NLU). This of course holds in the first place for researchers in computer science and AI, who have always been interested in computational processes (see e.g. Kowalik 1988). Recent developments in (computational) linguistics and cognitive psycholinguistics show that these NLU-related disciplines have also been moving towards parallel models. A major factor influencing this development is the rapidly growing interest in *interactive* approaches to NLU (see e.g. Briscoe 1987, Altmann 1988, Altmann & Steedman 1988). Briefly, these models move away from the traditional linguistics-inspired views of language understanding as non-interactive, i.e. as a serial application of processing modules whose sole means of communication is a unidirectional input-output channel (cp. Forster 1979). There is a growing belief (based on experience in computational linguistics and on psycholinguistic experimentation) that non-interactive models are incorrect, and interactive ones allowing more flexible communications among components (mainly to deal with ambiguity) prove to be superior. Although matters are more complicated than stated here, it will be clear that interactive models lend themselves easily to parallel architectures (see also Adriaens & Hahn forthcoming). For the presentation of the Parallel Expert Parser, I will only briefly distinguish two kinds of approaches to parallel NLU. On the one hand, there is what can be called *fine-grain parallelism;* on the other hand, there is *coarse-grain parallelism.* With *fine-grain parallel NLU* I refer basically to the connectionist approach and its decendants. Connectionist models feature huge networks of small nodes of information; computation is represented by fluctuations of the activation levels of nodes and by (parallel) transmission of excitation and inhibition along connections. (For connectionism in general, see Feldman & Ballard 1982, VanLehn 1984, Hillis 1986, McClelland & Rumelhart 1986; for connectionist models of NLU, see Cottrell & Small 1983, Cottrell 1985, Pollack & Waltz 1985, McClelland & Rumelhart 1986). With *coarse-grain parallel NLU,* I refer to a more modest kind, in which the smallest item of information is more complex than a node in a connectionist model (it may be a rule, for instance), but in which one attempts to keep the parallel computation involving the items of information more under control than can be done in a connectionist model. (For examples of coarse-grain parallel NLU, see Hirakawa 1983, Matsumoto 1987 or Granger, Eiselt & Holbrook 1986).

The research presented here is of the latter type of parallel NLU. A potentially parallel NLU system (the Word Expert Parser, Small 1980) has been drastically revised so as to allow a truly parallel implementation (viz. in Flat Concurrent Prolog, using the Logix environment (Silverman et al. 1986)); we call the resulting system the Parallel Expert Parser (PEP, Devos 1987).

2. The Word Expert Parser (WEP) revisited

The Word Expert Parser (WEP, Small 1980) is a natural language understanding program in the AI tradition of semantic parsing (see also Hirst 1983, Hahn 1986/1987, Cottrell 1985, Adriaens 1986a/b for WEP-inspired or -related work). The organization of the model differs strongly from that of a "classical" NLU system. Rather than having a number of components of rules that are applied (serially) to linguistic input by a general process, WEP considers the words themselves as active agents (word experts) that interact with each other and with other knowledge sources in order to find the meaning of a fragment of text. Words are implemented as coroutines, i.e. processes that run for a while (broadcasting information or performing side-effect operations to refine the representation of the meaning of a text fragment), and suspend when they have to wait for information from other experts. The information they send or wait for are either signals relating to the status of the parsing process (broadcast on a dedicated signal channel) or concepts that represent the meaning of parts of the linguistic input (broadcast on a dedicated concept channel). The experts coordinate the understanding process in turn, eventually converging towards a conceptual structure that represents the meaning of a text fragment.

Although the model inspired several researchers, it has received little attention in the linguistic community (but see Berwick 1983) and has been considered as "an interesting rarity". Moreover, the original researchers have abandoned the model in favor of the connectionist approach mentioned above. Yet, the original model still has some interesting features that are worthy of further consideration; on the other hand, both for linguistic and for computational reasons, some drastic revisions are needed.

3. From WEP to PEP

In general, the idea of parallel interacting processes is a very attractive one if one wants a flexible parser capable of using any type of information at any moment it needs it. This basic principle of WEP has been retained for PEP. Yet, although the design of the system seemed to lend itself easily to a parallel implementation, linguistic and computational flaws in the model have made drastic revisions necessary before this could actually be done.

3.1 True parallelism

Although WEP claimed to be "potentially parallel", it heavily (and implicitly) relied on sequentiality to make its principles work. Especially for the restarting of suspended experts, a last-in first-out regime (stack) took care of contention for messages: the expert that placed an expectation for a message last, mostly got it first. Also, to avoid complications in expert communication, no new experts were initialized before the queue of ready-to-run experts was empty. The adherence to this sequentialization, not to mention the side-effects involved, obviously made WEP's claim of being "potentially parallel" invalid.

In a truly parallel environment, sequentiality can no longer be relied on. PEP uses parallelism whenever possible: for the execution of expert code AND for initializing new experts (initializing all of them as soon as they are read and morphologically analyzed). In order to realize this, the most important departure from the original model is that experts are no longer only associated with words (the only linguistic entities acknowledged by WEP). We will now discuss what experts are associated with, and how the new view of experts leads to clearer and more explicit concepts of waiting and communicating in a parallel environment.

3.2 Word-Experts versus Concept-Experts at different levels

A major item of criticism uttered against WEP has been that it considers the word as the only entity to be turned into an expert process. Linguistically speaking, the existence of larger constituents is undeniable and must be taken into account, whatever model one advocates. From the computational viewpoint, squeezing all interactions into words makes it almost impossible to figure out what is going on in the overall parsing process (This non-transparency is one of the reasons why fully-integrated interactive models are not so popular). Words have to decide on everything, from morphological issues to pragmatic issues, with jammed communication channels as a result.

In PEP, experts are associated with *concepts* rather than with words. It is very natural to do so: words are only used to evoke the concepts that constitute the meaning of a fragment of text. Still, concepts have a concrete link to words and can be regarded as being associated with the group of words that evokes them. E.g. in "the young girl" three concepts can be discovered, associated with the basic word-groups "the", "young" and "girl". At a higher level a compound concept constituting the meaning of the entire construct "the young girl" is invoked.

Concretely, in PEP a specific data structure (the *expert frame*) is associated with every expert. The hierarchy that originates from the concepts is reflected by the interconnection of the expert frames. These are vertically related by level interdependencies, and horizontally by the relative role the concepts of the frames play in the frame that is being built out of them one level higher. Besides its level, an expert frame has three attribute slots: a *function* attribute (stating what the role is the expert concept plays at a specific level), a *concept* attribute (representing the contents of the expert) and a *lexical* attribute (simply corresponding to the group of words associated with the concept). Below, we will see that this definition of an expert frame is crucial for the restricted communication protocol among experts.

The "analysis process" consists of the collection of currently active experts that try to establish new concepts. If a new concept can successfully be formed, the corresponding expert is added to the analysis process, while the combined concept's experts may die. They pass their expert frames, and so the contained information, to the new expert, which will usually incorporate them in its own expert frame. Notice that this view has interesting software engineering aspects not present in WEP: by having a leveled approach expert code becomes more local, modular and adaptable. The dynamic process hierarchy enables the linguist/expert writer to write generic or prototypical experts that can be parameterized with the value of the concept they represent (cp. object-oriented programming).

A final note about the levels. Each level is intended to deal with a more or less independent part in the derivation and composition of meaning. However, we leave it up to the linguists writing the expert processes to declare (1) what levels they want to consider and (2) what the appropriate functions are that they want to use at the respective levels. By combining this flexible filling in of a rigorously defined model, we force the linguist-user to clearly specify the experts and help him to keep the experts relatively small (hence, more readable) and to figure out more easily where things could go wrong in the parsing process. A possible hierarchy of levels might be: morpheme, word, constituent, clause, sentence (each level having its own function attributes). In the somewhat oversimplified example below, three levels will be used (between brackets: the respective function attributes), viz. word_level [article;adjective;substantive], constituent_level [action;agent;object], and sentence_level.

3.3 Broadcasting versus Explicit Communication

Experts are the active components of the analysis system. New concepts come into existence only through their interaction. Since parallelism was a major goal of the PEP approach, the communication protocols have been based on explicit identification of the expert frames involved in some interaction, which allows one to keep communication under control. Two kinds of communication take place:

(1) attribute-refining:

Experts are allowed to refine the attributes of expert frames. The attributes are considered to be information that is accessible by all experts.

(2) attribute-probing:

Basing themselves on the attributes of the probed expert frames, experts decide which way to go in the analysis process. All attribute probing is in the choose_alt predicate, that is described in the next subsection.

3.4 Suspending/Resuming: Explicit Machinery versus Declarative Reading

In the course of the analysis process, experts walk through a discrimination network, gradually refining and constructing the meaning of a text fragment. The predicate that allows experts to decide which way to go in this process on the basis of information they expect to get from other experts is the *choose_alt* predicate:

```
choose_alt([
        alt(frame(frame-specification,
                  attribute_condition),
                     invoke(expert)),
        alt(frame(frame-specification,
                  attribute_condition),
                  invoke(expert)),
         ...
        else(invoke(expert))
        ]).
```

It consists of a number of alternatives and an optional elsative. The alternatives contain a test, which may fail, suspend or succeed. In the last case the corresponding expert may be invoked. If tests from several alternatives succeed, an arbitrary corresponding expert is invoked, whereas the others are not further considered (don't-care committed choice; see also below and Devos 1987, however, for a suggestion of how to realize non-determinism in view of ambiguity). Only after failure of all tests is the elsative-expert executed.

Tests consist of a frame-specification and an attribute-condition. The latter constitutes the actual test on the attribute of the frame selected by "frame-specification". This frame can be referred to with *testframe* in the corresponding invoked expert. One will already have noticed that the choose_alt predicate does not contain any explicit scheduling commands. Indeed, the intention is to entirely mask the program flow by a declarative reading. However, flow control remains necessary and it is realized by suspending an expert routine (or a branch in the choose_alt predicate, since the alternatives in the choose_alt may be executed in parallel), if it requires information that is not yet available. Only after this required information is filled in, does the expert-routine resume. This can cheaply be implemented using read-only unification (Shapiro 1986). Intuitively, predicates that probe for information suspend, if the variable that supplies this information is not yet instantiated. This suspension takes place during unification of the Flat Concurrent Prolog (FCP) predicate (see below), into which expert routines are compiled. Resumption occurs whenever the required variable gets instantiated. Suspension of a choose_alt branch may take place in the following cases:

(1) If the search for the testframe requires information that is not yet available, it simply suspends. As a result the frame-specification always leads to the selection of a frame in a deterministic way. Hence, explicit communication becomes possible.

(2) The attribute-test suspends until the information to be tested is available.

There is one other predicate or command that may cause suspension of an expert, viz. begin_level (a_level). The execution of an expert that specifies begin_level(a_level), is only resumed after all attributes of incorporated expert frames are specified. This filling in of attributes takes place between different expert frames on the same level (intra-level communication). With rigid rules as to which expert fills in which frame, it is possible to prove that the expert code is deadlock free. These rules will further be referred to as the deadlock avoidance rules. It suffices e.g. to prove that every frame that is at the lowest level that still contains unfilled frames, will eventually be filled in. It must then not be difficult to construct a deadlock analyser, that checks whether the deadlock avoidance rules are violated. This has not yet been further elaborated.

However, to ensure flexibility (especially from linguistic considerations) we are forced to allow inter-level communication. e.g. in sentences as "the little girl loved her toy", where "her" is level equivalent to "little", but anaphorically refers to "the little girl", which will probably be at a higher (hence, different) level than "her". In this case deadlock free code is not easy to guarantee, because of the possibility of circular waiting of experts for one another. It is our hope that we can also incorporate restricted and well-specified use of this inter-level communication in the deadlock avoidancy rules.

The system as yet designed, implements a don't-care committed-choice between the alternatives of a choose_alt predicate. This means that an arbitrary alternative that succeeds, will be chosen to determine the expert's behaviour. We are well aware of the fact that don't-care committed-choice is not always what one wants in AI applications. We merely chose this (easy) option here in order not to burden the design and implementation with one more serious problem. Two alternatives to be explored in the future are the following.

The first is intermediate between don't-care committed-choice and full non-determinism. To each alternative in the choose_alt command a priority is assigned. The alternatives are then tried out by descending priority, allowing the more likely ones to succeed first. (These priorities will often reflect frequency of occurrence of specific linguistic structures.) A prioritizing approach like this one will however require more synchronisation among the alternatives of the choose_alt to ensure a unique semantics of the command.

The second is full non-determinism. No priorities are assigned to alternatives, and the system is capable of undoing a wrong choice during the analysis process. It can go back to a choice point and try out another alternative whose test succeeds. A (costly) implementation of this strategy should be based on Concurrent Prolog code (Shapiro 1986) that contains a copy of the global environment for each alternative in the choose_alt command. This Concurrent Prolog code would then have to be flattened to FCP (Codish & Shapiro 1985).

3.5 An Example Analysis

Below we present the code of some sample experts that allow the analysis of the sentence "the little girl eats the apple". The example is simplified, but illustrates well the crucial elements of PEP. First the appropriate levels and functions are declared. Then follows the code of the actual experts. Remember that *expframe* refers to the frame that is associated with the expert and *testframe* refers to the frame that was referred to in the alternative of the preceding choose_alt command. "begin_frame" sets the appropriate level and "refine_function" and "refine_concept" do the filling in of the attributes of the specified frame. The lexical attribute is automatically filled in when beginning the frame. The example restricts itself to choose_alt commands that only require intra-level commur..cation. When the sentence is read, the corresponding experts are initialized and start to run in parallel. The rest of the code is self-explanatory.

```
declare(level[
      word_level
     (function[article,adjective,substantive]),
      constituent_level
     (function[action,agent,object]),
      sentence_level
     (function[])
                ]).

the :-
begin_frame(word_level),
refine_function (expframe, 'article'),
refine_concept (expframe, kind("defining")),
refine_concept (expframe, value("defined")).

little :-
begin_frame(word_level),
refine_function(expframe, 'adjective'),
refine_concept(expframe, kind("adjectival")),
refine_concept(expframe, value("young, small")).

girl :-
begin_frame(word_level),
refine_function(expframe, 'sustantive'),
refine_concept(expframe, kind("person")),
refine_concept(expframe, value("female, child_or_maiden")),
choose_alt
  ([alt(frame(minus(1),function(equal('article'))),
          invoke(article_incorporation)),
      alt(frame(minus(1),function(equal('adjective'))),
          invoke(adjective_incorporation)),
      else(invoke(no_incorporation))]).

apple :- analogous to the code for girl.

adjective_incorporation :-
incorporate(testframe),
choose_alt
   ([alt(frame(minus(1),function(equal('article'))),
           invoke(article_incorporation)),
      else(invoke(no_incorporation)) ]).

article_incorporation :-
incorporate(testframe),
begin_frame(constituent_level),
refine_concept(expframe, kind("unused")),
refine_concept(expframe, value("unused")).

no_incorporation :-
begin_frame(constituent_level),
refine_concept(expframe, kind("unused")),
refine_concept(expframe, value("unused")).

eats :-      begin_frame(constituent_level),
refine_function(expframe, 'action'),
refine_concept(expframe, kind("ingest")),
refine_concept(expframe, value("ingest_food")),
choose_alt
   ([alt(frame(plus(3),concept(view('eatable'))),
          invoke(eat_something)),
      else(.....................)) ]).

eat_something :-
refine_function(testframe, 'object'),
incorporate(testframe),
choose_alt
    ([alt(frame(minus(1),concept(view('person'))),
            invoke(someone_eats_something)),
       else(.....................)) ]).

someone_eats_something :-
refine_function(testframe, 'agent'),
incorporate(testframe),
begin_frame(sentence_level),
show_solution.
```

4. A Parallel Implementation

In the last section of this paper the implementation (in a logic programming language) of all aspects of PEP discussed so far will receive a closer look. For this implementation Logix has been used, a Flat Concurrent Prolog environment (Silverman et al. 1986).

4.1 General Model Organization

The prototype realization of the PEP model allowing for correct analysis of very simple sentences (such as "The man eats", "A man eats", "Man eats") consists of an expert language (EL) to be used by the linguist when writing his experts, a precompiler that transforms the experts to FCP code and the Logix FCP compiler/emulator, the programming environment. Linguists are offered the EL, which only contains predicates at a high level of abstraction. They may further tune the expert levels discussed earlier and the function attributes they will be using at each level to their own needs. They are only allowed to use the EL predicates according to their own specification of levels and function attributes. The EL is then precompiled to FCP. The main reason for the approach of precompiling is that flattening techniques have to be used on the predicates. These techniques are the domain of computer scientists and the linguist should not be bothered with them. (Precompiling also offers important additional advantages such as syntax checking, checking of potential deadlock, etc.; these features are still under development).

4.2 Data-Structures: Frame Interconnection and Blackboard Information

The lexical-morphological analyzer schedules and invokes the experts corresponding to the elementary lexical units and outputs a blackboard, i.e. a matrix with slots whose columns correspond to those units and whose rows correspond to a level. Each expert has one expert frame associated with it; this expert frame fills one slot of the blackboard. In the beginning of the analysis process all frames and the blackboard contain uninstantiated slots. Experts gradually instantiate the slots. Referring to another expert's expert frame requires walking to it over the blackboard. The walk is defined in a unique way. All slots on the path should be instantiated, otherwise the walk suspends and waits for the instantiation. This is elegantly implemented using the read-only unification of the parallel Prolog versions. Slots that will never be of any use any more, are instantiated to dummy constants in order not to indefinitely block suspended walks.

5. Conclusions and further research

In this paper a further development of the procedural view of natural language analysis (NLU) as proposed by Small's Word Expert Parser has been presented. The Parallel Expert Parser tries to present a truly distributed and parallel interactive model of NLU with clearly defined experts on different levels, hierarchically conceived expert frames and rigidly restricted communication protocols.

Besides further work on the implementation and writing/testing more complex experts, the necessary model of knowledge (concept) representation that has to complete the framework is a major issue for further research. As mentioned above, Hahn (1987) has already worked on this matter, introducing generic, prototypical experts. This is not just a knowledge representation matter, but also one of integrating parallel Prolog with an object-oriented framework (Bourgois, forthcoming).

REFERENCES

ADRIAENS, G. (1986a) - Word Expert Parsing Revised and Applied to Dutch. In Proceedings of the 7th ECAI (Brighton, UK), Volume I, 222-235.

ADRIAENS, G. (1986b) - Process Linguistics: The Theory and Practice of a Cognitive-Scientific Approach to Natural Language Understanding. Phd. thesis, Depts of Linguistics and Computer Science, University of Leuven, Belgium.

ADRIAENS, G. (1987) - A Critical Description of the Coroutine Regime Used in the Word Expert Parser. Computer Science Thesis, Dept of Computer Science, University of Leuven, Belgium.

ADRIAENS, G. & U. HAHN (eds, forthcoming) - Parallel Models of Natural Language Computation. Ablex, New Jersey.

ALTMANN, G. (1988) - Ambiguitiy, Parsing Strategies, and Computational Models. In *Language and Cognitive Processes 3 (2), 73-97.*

ALTMANN, G. & M. STEEDMAN (1988) - Interaction with context during human sentence processing. In *Cognition* 30, 191-238.

BERWICK. R.C. (1983) - Transformational Grammar and Artificial Intelligence: a Contemporary View. In *Cognition and Brain Theory* 6(4), 383-416.

BOURGOIS, M. (forthcoming) - The Parallel Expert Parser comes of age. Explorations in the integration of parallel Prolog and Object-oriented Knowledge Representation for Natural Language Understanding. AI Thesis, Department of Computer Science, University of Leuven, Belgium.

BRISCOE, E.J. (1987) - *Modelling Human Speech Comprehension: A Computational Approach.* Ellis Horwood, Chichester UK.

CODISH, M. & SHAPIRO, E. (1986) - Compiling OR-parallelism into AND-parallelism. Technical Report CS85-18, Department of Applied Mathematics. The Weizmann Institute of Science, Israel.

COTTRELL, G.W. (1985) - A Connectionist Approach to Word Sense Disambiguation. University of Rochester Computer Science Phd (TR-154). Rochester, New York.

COTTRELL, G.W. & S.L. SMALL (1983) - A Connectionist Scheme for Modelling Word Sense Disambiguation. In *Cognition and Brain Theory* 6 (1), 89-120.

DEVOS, M. (1987) - The Parallel Expert Parser. Realization of a Parallel and Distributed System for Natural Language Analysis In Logic Programming Languages. Engineer's Thesis, Department of Computer Science, University of Leuven, Belgium (in Dutch).

FELDMAN, J.A. & D.H. BALLARD (1982) - Connectionist Models and Their Properties. In *Cognitive Science* 6, 205-254.

GRANGER, R.H., K.P. EISELT & J.K. HOLBROOK (1986) - Parsing with Parallelism: A Spreading-Activation Model of Inference Processing During Text Understanding.

HAHN, U. (1986) - A Generalized Word Expert Model of Lexically Distributed Text Parsing. In Proceedings of the 7th ECAI (Brighton, UK), Volume I, 203-211.

HAHN, U. (1987) - Lexikalisch verteiltes Text-Parsing. Eine objekt-orientierte Spezifikation eines Wortexpertensystems auf der Grundlage des Aktorenmodells. University of Konstanz Department of Information Science PhD Thesis.

HILLIS, D. (1986) - The Connection Machine. MIT Press, Cambridge Mass.

HIRAKAWA, H. (1983) - Chart Parsing in Concurrent Prolog. Technical Report of the ICOT Research Center (TR-008). Institute for New Generation Computer Technology, Tokyo.

HIRST, G. (1983) - A Foundation for Semantic Interpretation. In Proceedings of the 21st ACL (Cambridge, Mass), 64-73.

KOWALIK, J.S. (ed) (1988) - Parallel Computation and Computers for Artificial Intelligence. Kluwer, Dordrecht The Netherlands.

MATSOMOTO, Y. (1987) - A Parallel Parsing System for Natural Language Analysis. In *New Generation Computing* 5 (1987), 63-78.

McCLELLAND, J. & RUMELHART D.E. (1986) - Parallel Distributed Processing. MIT Press, Cambridge, Mass.

POLLACK, J. & D. WALTZ (1985) - Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation. In *Cognitive Science* 9, 51-74.

SILVERMAN, W. et al. (1986) - The Logix System User Manual - Version 1.21. Technical Report CS-21 , Department of Computer Science. The Weizmann Institute of Science, Rehovot 76100, Israel.

SHAPIRO, E. (1986) - Concurrent Prolog: A Progress Report. Fundamentals of Artificial Intelligence, W. Bibel & Ph. Jorrand. Lecture Notes in Computer Science, Springer-Verlag, Berlin.

SMALL, S.L. (1980) - Word Expert Parsing: a Theory of Distributed Word-Based Natural Language Understanding. Computer Science Technical Report Series. University of Maryland Phd.

VANLEHN, K. (1984) - A Critique of the Connectionist Hypothesis that Recognition Uses Templates, and not Rules. In Proceedings of the 6th Annual Conference of the Cognitive Science Society (Boulder, Colorado), 74-80.

Chart Parsing for Loosely Coupled Parallel Systems

Henry S. Thompson

Department of Artificial Intelligence
Centre for Cognitive Science

University of Edinburgh

Acknowledgements

The work described in this paper was made possible in part by the use of equipment made available by the Rank Xerox University Grants Programme, BBN Labs Inc., Artificial Intelligence Ltd., Xerox Palo Alto Research Center and Intel Scientific Corp.—to all of them my thanks. None of them are in any way responsible for the opinions expressed here.

0. Introduction

Of the parallel systems currently available, far and away the most common are loosely coupled collections of conventional processors, and this is likely to remain true for some time. By loosely coupled I mean that the processors do not share memory, so that some form of stream or message-passing protocol is required for processor-processor communication. It follows that in most cases the programmer must make explicit appeal to communication primitives in the construction of software which exploits the available parallelism. Even in shared-memory systems, the absence of parallel constructs from available programming languages may mean that appeal to a similar communication model may be necessary, at least in the short term.

Although not ideally suited to loosely coupled systems, the general problem of parsing for speech and natural language is of sufficient importance to merit investigation in the parallel world. This paper reports on explorations of the computation-communication trade-off in parallel parsing, together with the development of an portable parallel parser which will enable the comparison of a variety of parallel systems.

I. Parsing for Loosely Coupled Systems

Given the prevalence of loosely coupled systems, although one might suppose that shared-memory parallelism offers greater scope for the construction of parallel parsing systems, and parallel chart parsers in particular, none-the-less it is a good idea to look at what can be done in the loosely coupled case.

Loosely coupled parallel systems can be expected to do best, that is, show a nearly linear (inverse) relationship between solution time and number of processors, when the problem at hand is (isomorphic to a) tree-search problem with large initial fan-out and compact specifications of sub-problems and results. In such problems, the ratio of communication to computation is low, so the loose coupling does not significantly impede linear speed-up. Large problems can be broken down into as many pieces as there are processors, cheaply distributed to them, and the results cheaply returned.

Parsing of single sentences is not obviously suited to loosely coupled parallel systems. Whether one attacks single-sentence parsing by some form of left-to-right breadth-first parse, or by some form of all-at-once bottom-up breadth-first parse, very high communication costs would seem to be involved. The only hope would seem to be to pursue the latter route nevertheless, and see whether the communication costs can be brought down to an acceptable level. There are a number of different dimensions along which one might try to parallelise the parsing process, but insofar as they involve the distribution of sub-problems, they are highly likely to require the representation of partial solutions. Since this is a primary characteristic of the active chart parsing methodology, my investigations have focussed on parallel implementations of active chart parsers.

II. Parallelism and the Chart

We start with the observation that chart parsing seems a natural technique to base a parallel parser on. Its hallmarks are the reification of partial hypotheses as active edges, and the flexibility it allows in terms of search strategy, and it would seem straight-forward to adapt a chart parser doing pseudo-parallel breadth-first bottom-up parsing into a genuinely parallel parser. Indeed with a shared-memory parallel system, the BBN Butterfly™, I have done just that, and the result exhibits the expected linear speed-up. The approach used was simply to allow multiple processors to remove entries from the queue of hypothesised edges and add them to the chart in parallel, performing the associated parsing tasks and thereby in some cases hypothesising further edges onto the queue. Locks were of course required to prevent race conditions in updating the chart and edge queue, but instrumentation suggested that there was rarely contention for these locks and they had little adverse impact on performance.

Clearly this approach would not be appropriate in the loosely coupled case. One could of course use some system which supports virtual shared memory to implement a shared chart and edge queue. But this would defeat the whole purpose, as the parser would be serialised by the processor responsible for maintaining the shared structures. What I have explored instead is retaining the same granularity of parallelism, namely the edge, but accepting that at least some of the chart itself will have to be distributed among the processors.

III. Distributing the Chart

I have explored the approach of distributing the chart among the processors in several implementations of a chart parser for the Intel Hypercube™, a loosely coupled system, and for a network of Lisp workstations. A memory-independent representation of the chart is used, allowing edges to be easily encoded for transmission between processors. The chart is distributed among the processors on a vertex by vertex basis. Vertices are numbered and assigned to processors in round-robin fashion. Edges 'reside' on the processor which holds their 'hot' vertex, that is, their right-hand vertex if active, left-hand if inactive. From this it can be seen that once a new edge is delivered to its 'home' processor, that processor has all the edges required to execute the fundamental rule with respect to that new edge. Each processor also has a copy of the grammar, so it can perform rule invocation as necessary, and a copy of the dictionary, so that once the input sentence is distributed, pre-terminal edge creation can proceed in parallel.

The following three figures illustrate the distribution of vertices and edges for a simple example sentence and grammar, assuming a three processor system.

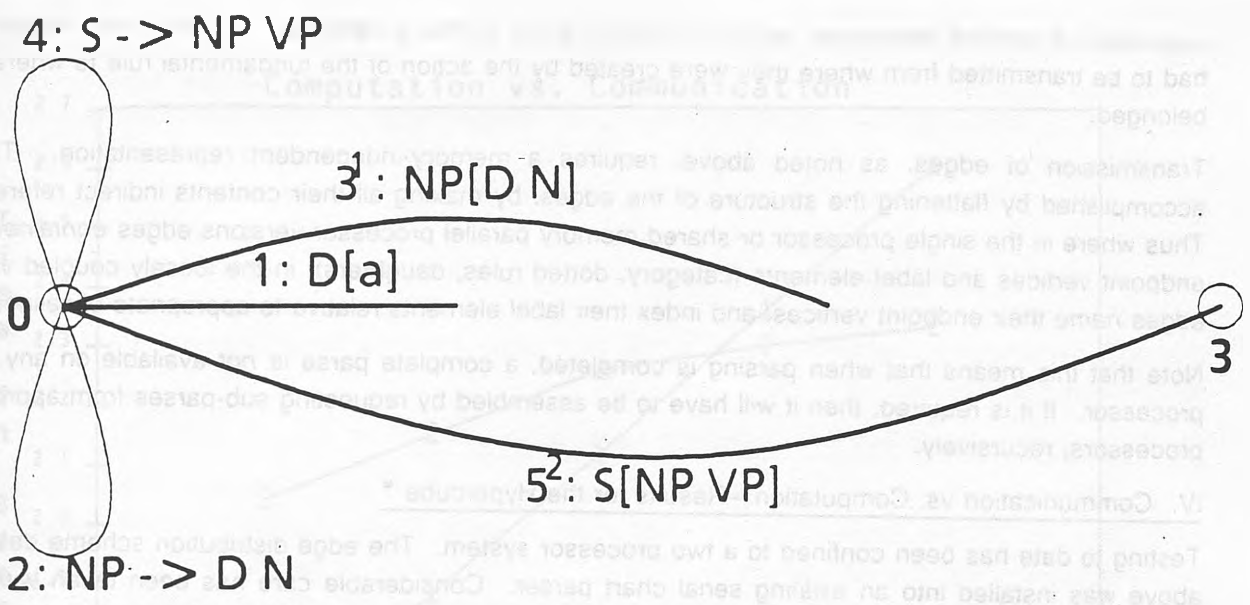


Figure 1a. Chart portion resident on processor 0

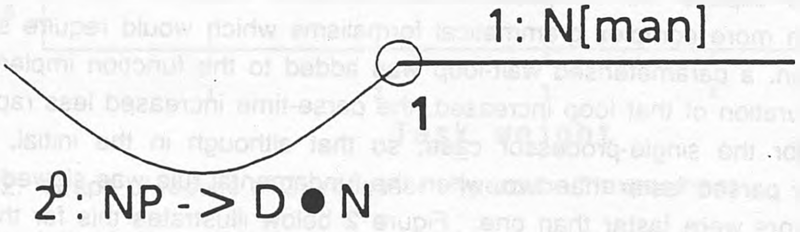


Figure 1b. Chart portion resident on processor 1

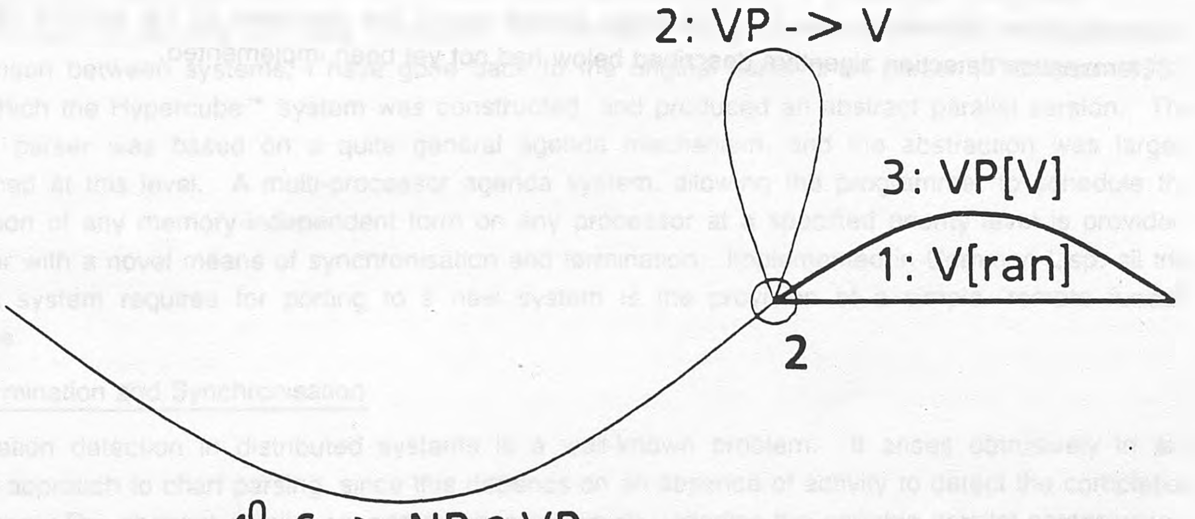


Figure 1c. Chart portion resident on processor 2

Vertices are numbered circles. Edges are thin if active, thick if inactive, and their contents are noted. They are numbered on a per processor basis. Those with superscripts, e.g. 4^0, are ones which originated on another processor, whose number is given by the superscript. Of the eleven edges, four had to be transmitted from where they were created by the action of the fundamental rule to where they belonged.

Transmission of edges, as noted above, requires a memory-independent representation. This is accomplished by flattening the structure of the edges, by making all their contents indirect references. Thus where in the single processor or shared memory parallel processor versions edges *contained* their endpoint vertices and label elements (category, dotted rules, daughters), in the loosely coupled version edges *name* their endpoint vertices, and index their label elements relative to appropriate baselines.

Note that this means that when parsing is completed, a complete parse is *not* available on any single processor. If it is required, then it will have to be assembled by requesting sub-parses from appropriate processors, recursively.

IV. Communication vs. Computation—Results for the Hypercube™

Testing to date has been confined to a two processor system. The edge distribution scheme described above was installed into an existing serial chart parser. Considerable care has been taken within the limits imposed by the host system communications primitives to keep communication bandwidth to a minimum (approx. 100 bits/edge in a single packet). Even with a relatively trivial grammar and lexicon and simple sentences of limited ambiguity, two processors are faster than one under some circumstances. In order to explore the computation/communication trade-off, and to simulate the operation of the system with more complex grammatical formalisms which would require substantially greater per-edge computation, a parameterised wait-loop was added to the function implementing the fundamental rule. As the duration of that loop increased, the parse-time increased less rapidly for the two processor case than for the single-processor case, so that although in the initial, un-slowed, condition, a single processor parsed faster than two, when the fundamental rule was slowed by a factor of around four, two processors were faster than one. Figure 2 below illustrates this for the sentence *The orange saw saw the orange saw with the orange saw* with a standard grammar which allows for PP attachment ambiguity and a lexicon in which *orange* is ambiguous between N and A and *saw* is ambiguous between N and V. The times plotted are to the discovery of the second parse, as the termination detection algorithm described below had not yet been implemented.

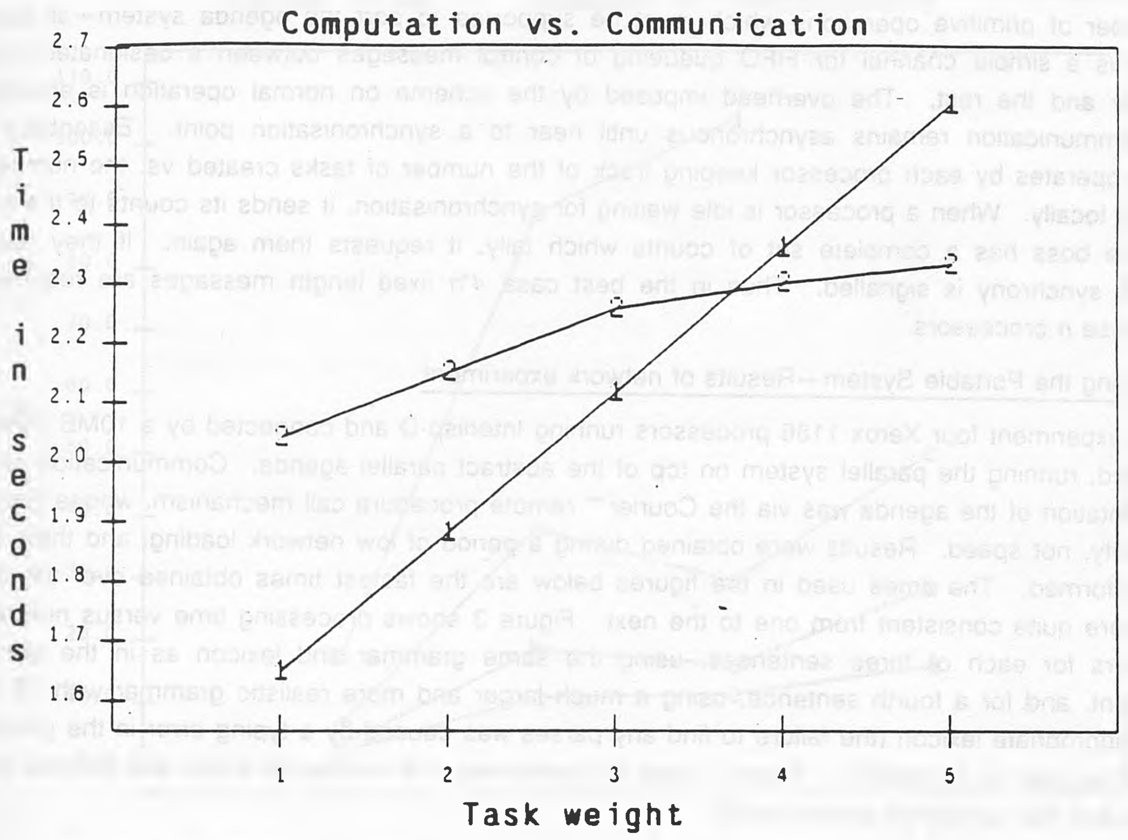


Figure 2. Graph of results of 2 processor Hypercube™ experiment

It is hoped that further experimentation with larger cubes will shortly be possible.

V. Towards Wider Comparability—The Abstract Parallel Agenda

With an eye to allowing an easy extension of this work to other systems, and more principled comparison between systems, I have gone back to the original serial chart parser (Thompson 1983) from which the Hypercube™ system was constructed, and produced an abstract parallel version. The original parser was based on a quite general agenda mechanism, and the abstraction was largely performed at this level. A multi-processor agenda system, allowing the programmer to schedule the evaluation of any memory-independent form on any processor at a specified priority level is provided, together with a novel means of synchronisation and termination. Implemented in Common Lisp, all this agenda system requires for porting to a new system is the provision of a simple 'remote funcall' primitive.

VI. Termination and Synchronisation

Termination detection in distributed systems is a well-known problem. It arises obtrusively in any parallel approach to chart parsing, since this depends on an absence of activity to detect the completion of parsing. The abstract parallel agenda mechanism which underlies the portable parallel parser uses a new (we think—see Thompson, Crowe and Roberts forthcoming for discussion) algorithm for effective synchronisation of task execution (of which termination is a special case). It is thus possible to reconstruct not only the prioritisation function of an agenda (run this in preference to this), but also the ordering function (run this only if that is finished). Unlike some existing termination algorithms, this one is particular appropriate where no constraints can be placed on processor connectivity (any processor may, and usually does, send messages to any other processor). It requires only a modest increase in the number of primitive operations which must be supported to port the agenda system—all that is required is a simple channel for FIFO queueing of control messages between a designated 'boss' processor and the rest. The overhead imposed by the scheme on normal operation is effectively zero—communication remains asynchronous until near to a synchronisation point. Essentially the scheme operates by each processor keeping track of the number of tasks created vs. the number of tasks run locally. When a processor is idle waiting for synchronisation, it sends its counts to the boss. When the boss has a complete set of counts which tally, it requests them again. If they haven't changed, synchrony is signalled. Thus in the best case *4*n* fixed length messages are required to synchronise *n* processors.

VII. Testing the Portable System—Results of network experiment

For this experiment four Xerox 1186 processors running Interlisp-D and connected by a 10MB Ethernet were used, running the parallel system on top of the abstract parallel agenda. Communication for the implementation of the agenda was via the Courier™ remote procedure call mechanism, whose hallmark is reliability, not speed. Results were obtained during a period of low network loading, and three trials were performed. The times used in the figures below are the fastest times obtained over the trials, which were quite consistent from one to the next. Figure 3 shows processing time versus number of processors for each of three sentences, using the same grammar and lexicon as in the previous experiment, and for a fourth sentence, using a much larger and more realistic grammar with 70 rules and an appropriate lexicon (the failure to find any parses was caused by a typing error in the grammar, detected too late for correction). Table 1 gives the sentences, the number of active and inactive edges involved and the number of parses found.

Sentence	active edges	inactive edges	parses
a: The orange saw saw the orange saw.	46	22	1
b: The orange saw saw the orange saw with the orange saw.	88	43	2
c: The orange saw saw the orange saw with the orange saw with the orange saw.	166	82	5
d: The front-end consists of those phases that depend primarily on the source-language.	285	58	0

Table 1. Sentences used in the network experiments

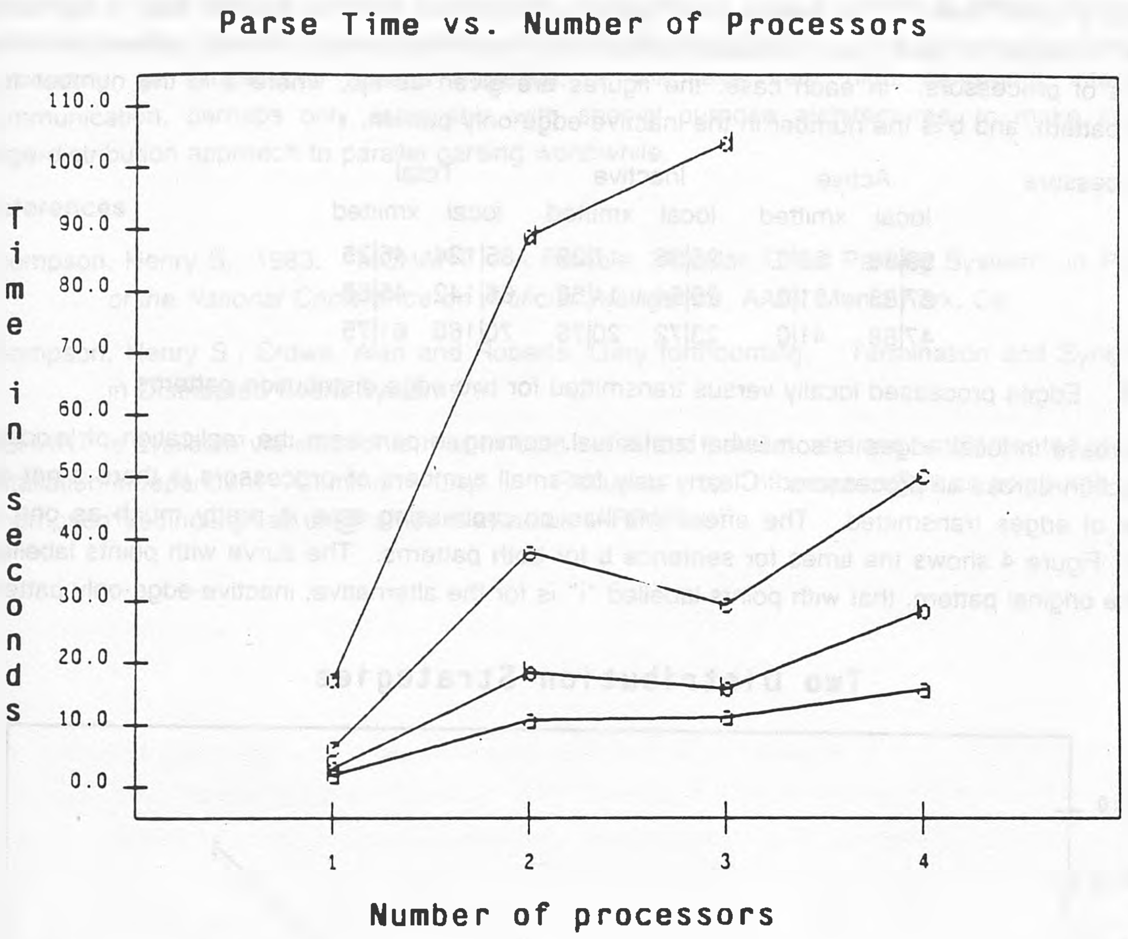


Figure 3. Graph of results of network experiment

Clearly not much encouragement can be taken from this experiment. Although there is some speed-up from two to three processors in some cases, overall the pattern is one where communication costs clearly dominate, so no advantage is gained. With slower processors and/or faster networks, we might hope to see better results, especially given the results in section IV, but the appropriateness of this approach to networked systems must remain in doubt in the absence of better evidence.

VIII. Alternative Patterns of Edge Distribution

One possible alternative decomposition of the task, which might offer some hope of improving the computation/communication trade-off, would be to transmit only inactive edges, but to send them to all processors. Then every processor would have the complete inactive chart, and could run active edges from start to finish without ever sending them anywhere. In order to distribute the computational load, rule invocation would be distributed on a per vertex basis. That is, each processor would only do bottom up rule invocation for those inactive edges which began at a vertex owned by that processor. The plus side of this route is that it sends only inactive edges around, which are simpler to encode, that the final result is available on a single processor, indeed on all processors, without having to be assembled, and that active edge processing is more efficient. The minus side is that the inactive edges have to be sent to all processors. In the simple example given in Figure 1, this actually balances out—four edges in the original implementation, two edges twice in the alternative one. A further experiment with the network system was conducted to explore this approach. The same sentences as before were used, but this time with the new edge distribution pattern. Table 2 below compares sentence b from Table 1, *The orange saw saw the orange saw with the orange saw*, in terms of the number of edges of each type processed locally and transmitted under the two patterns for different numbers of processors. In each case, the figures are given as a|b, where a is the number for the original pattern, and b is the number in the inactive-edge-only pattern.

# of processors	Active		Inactive		Total	
	local	xmitted	local	xmitted	local	xmitted
2	59|88	29|0	26|36	17|25	85|124	46|25
3	57|88	31|0	29|54	14|50	86|142	45|50
4	47|88	41|0	23|72	20|75	70|160	61|75

Table 2. Edges processed locally versus transmitted for two edge distribution patterns

The increase in local edges is somewhat artifactual, coming in part from the replication of lexical edge construction across all processors. Clearly only for small numbers of processors is there a net gain in number of edges transmitted. The effect this has on processing time is pretty much as one would expect. Figure 4 shows the times for sentence b for both patterns. The curve with points labelled "o" is for the original pattern, that with points labelled "i" is for the alternative, inactive-edge-only pattern.

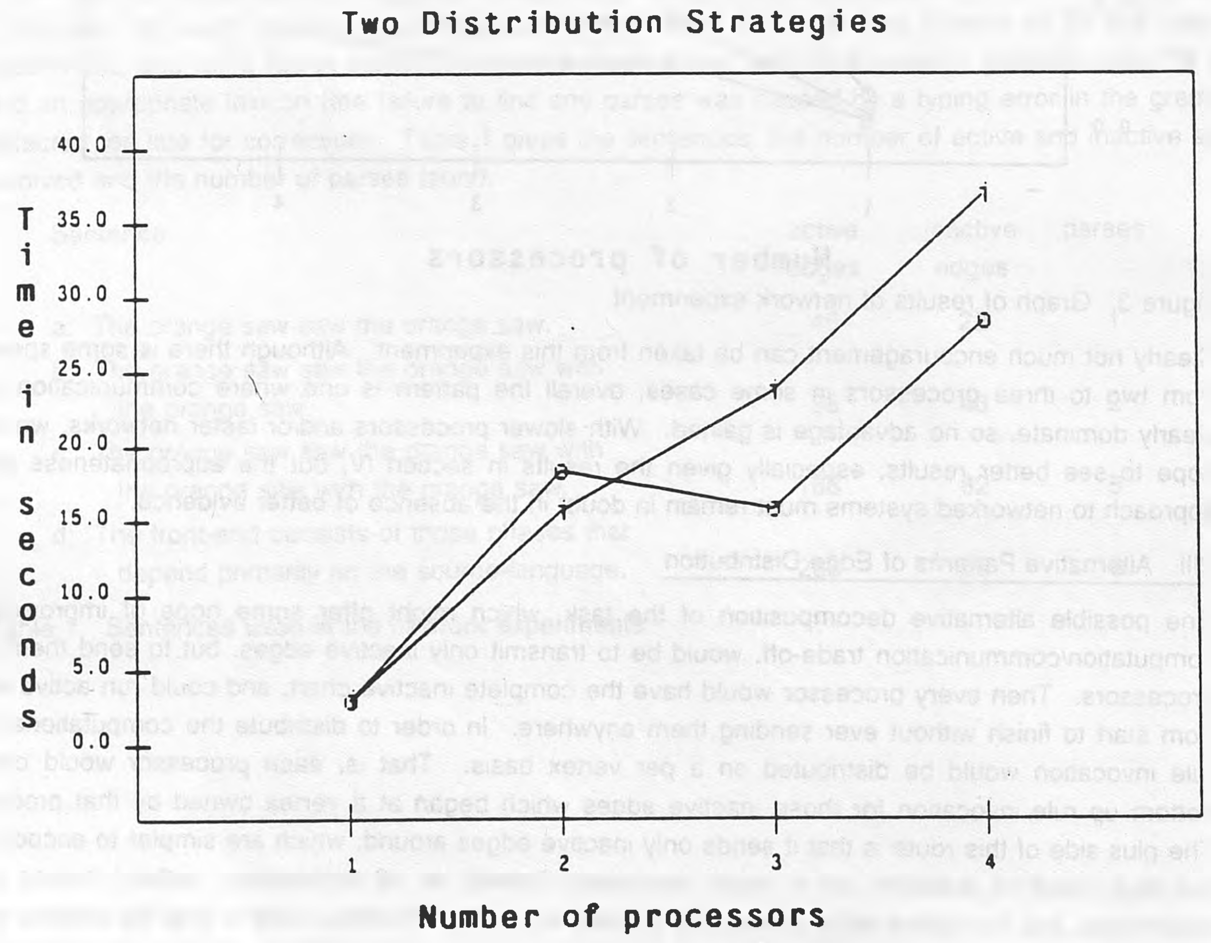


Figure 4. Graph of alternative distributions strategies for parsing sentence b

As expected, only in the two processor case do we see an advantage to the alternative approach. In general it is clear that the principle determinate of processing time is number of edges transmitted—the overhead in the network communication dominates all other factors. The obvious conclusion is that, particularly as processors speeds increase, it will take very high bandwidth inter-processor communication, perhaps only achievable with special purpose architectures, to make at least this edge-distribution approach to parallel parsing worthwhile.

References

Thompson, Henry S. 1983. "MCHART -- A Flexible, Modular, Chart Parsing System", in *Proceedings of the National Conference on Artificial Intelligence*, AAAI, Menlo Park, Ca.

Thompson, Henry S., Crowe, Alan and Roberts, Gary forthcoming. "Termination and Synchronisation in Distributed Event Systems".

MCHART is available via electronic mail in both serial and parallel versions, implemented in a relatively installation-independent Common Lisp. Requests to hthompson@uk.ac.edinburgh (JANET), hthompson%edinburgh.ac.uk@nsfnet-relay.ac.uk (ARPANet).

Parallel Generalized LR Parsing based on Logic Programming

Hozumi TANAKA
Tokyo Institute of Technology

Hiroaki NUMAZAKI
Tokyo Institute of Technology

Abstract

A generalized LR parsing algorithm, which has been developed by Tomita[Tomita 86], can treat a context free grammar. His algorithm makes use of breadth first strategy when a conflict occcurs in a LR parsing table. It is well known that the breadth first strategy is suitable for parall processing. This paper presents an algorithm of a parallel parsing system (PLR) based on a generalized LR parsing. PLR is implemented in GHC[Ueda 85] that is a concurrent logic programming language developed by Japanese 5th generation computer project. The feature of PLR is as follows: Each entry of a LR parsing table is regarded as a process which handles shift and reduce operations. If a process discovers a conflict in a LR parsing table, it activates subprocesses which conduct shift and reduce operations. These subprocesses run in parallel and simulate breadth first strategy. There is no need to make some subprocesses synchronize during parsing. Stack information is sent to each subprocesses from their parent process. A simple experiment for parsing a sentence revealed the fact that PLR runs faster than PAX[Matsumoto 87][Matsumoto 89] that has been known as the best parallel parser.

1 Introduction

As the length of a sentence becomes longer, the number of parsing trees increases and it will take a lot of time to parse a sentence. In order to achieve fast parsing, we should look for a parallel parsing system based on the most efficient and general parsing algorithms. One of parallel parsing systems we have ever known is PAX[Matsumoto 87][Matsumoto 89] that is based on Chart parser. It is well known that LR parser is the most efficient paser, since LR parsing algorithm runs deterministicly for any LR grammar which is a subset of context free grammar. Unfortunately, LR grammar is too weak to parse sentences of natural languages. When we apply LR parsing algorithm to context free grammar, it is an usual case that conflicts appears in a LR pasing table. So we need to generalize the LR parsing algorithm in order to process these conflicts. There are two kinds of strategies to resolve the conflicts, namely a depth first strategy and a breadth first strategy. Nilsson[Nilsson 86] has adopted a depth first strategy and Tomita[Tomita 86] a breadth first strategy which is called a generalized LR parsing. As it is easy for us to simulate the breadth first strategy by parall processing technique. We have developed a parallel generalized LR parsing system (PLR) based on the generalized LR parsing algorithm which makes use of a breadth first strategy.

After we will give a brief introduction of LR parsing algorithm in section 2, we will describe our PLR system details of which will be explained in section 3. PLR is implemented in a concurrent logic programming language called GHC[Ueda 85] that is developed by Japanese 5th generation computer project. One of the most significant feature of PLR is to regard each entry of a LR parsing table as a process which handles shift and reduce operations. If the process discovers a conflict in a LR parsing table, it creates and activates subprocesses in order to process shift and reduce operations in the conflict. These subprocesses run in parallel and simulate breadth first strategy. There is no need to make subprocesses synchronize during parsing. Stack information is sent to each subprocesses from their parent process. In order to understand PLR algorithm we will show a trace of actual parsing in subsection 3.6.

In section 4, we will explain some results of the experiment which parses sentences using PLR and PAX. The experiment revealed the fact that PLR runs faster than PAX that is one of the best parallel parsers.

2 Generalized LR Parsing algorithm

The generalized LR parser is guided by a LR parsing table which is generated from grammar rules given. Fig.2 shows an ambiguous English grammar. Fig.2 shows a LR parsing table generated from the English grammar. The LR parsing table is devided into two parts, an action table and a goto table.

The lefthand side of the table is called 'action table', the entry of which is determined by a pair of generalized LR parser's state (the row of the table) and a lookahead preterminal(the column of the table) of an input sentence. There are two kinds of operations, a shift and a reduce operations. Some entries of the LR table contains more than two operations which mean that there is a conflict in the entry, and a parser should conduct more than two operations at once.

The symbol 'sh N' in some entries means that generalized LR parser has to push a lookahead preterminal on the LR stack and go to 'state N'. The symbol 're N' means that generalized LR parser has to reduce several topmost elements on the stack using a rule numbered 'N'. The symbol 'acc' means that generalized LR parser ends with success of parsing. If an entry doesn't contain any operation, generalized LR parser recognizes an error.

The righthand side of the table is called a 'goto table' which decides a state that the parser should enter after every reduce operation. The LR table shown in fig.2 has 4 conflicts at the state 14 (row number 14) and state 16 for the column of 'p' and 'relp'. Each of four entries, which have a conflict, contains two operations, a shift and a reduce operation. Such a conflict is called a 'shift-reduce conflict'. When a parser encounters a conflict, it cannot determine which operation should be carried out first. In PLR explained in the next section, conflicts will be resolved using parallel processing technique and we do not mind the order of the operations in a conflict.

3 Implementation of PLR

PLR is implemented in GHC that is a concurrent logic programming language developed by Japanese 5th generation computer project. In our system, each entry in a LR parsing table are regarded as a process which will handle shift and reduce operations. If the process discovers a conflict in a LR parsing table, it activates subprocesses in order to process shift and reduce

(1)	S	→	NP, VP.
(2)	S	→	S, PP.
(3)	NP	→	NP, RELC.
(4)	NP	→	NP, PP.
(5)	NP	→	det, noun.
(6)	NP	→	noun.
(7)	NP	→	pron.
(8)	VP	→	v, NP.
(9)	RELC	→	relp, VP.
(10)	PP	→	p, NP.

fig.1: Ambiguous English grammar

	det	noun	pron	v	p	relp	$	NP	PP	VP	RELC	S
0	sh1	sh2	sh3					5				4
1		sh6										
2				re6	re6	re6	re6					
3				re7	re7	re7	re7					
4					sh7		acc		8			
5				sh10	sh7	sh9			12	11	13	
6				re5	re5	re5	re5					
7	sh1	sh2	sh3					14				
8					re2		re2					
9				sh10						15		
10	sh1	sh2	sh3					16				
11					re1		re1					
12				re4	re4	re4	re4					
13				re3	re3	re3	re3					
14				re10	sh7/re10	sh9/re10	re10		12		13	
15				re9	re9	re9	re9					
16				re8	sh7/re8	sh9/re8	re8		12		13	

fig.2: LR parsing table obtained from fig.1 grammar

```
(1) a:- true| b,c.
(2) b:- true| true.
(3) c:- true| true.
```

fig.3: typical stat nent of GHC

operations in the conflict. These subprocesses run in parallel and simulate breadth first strategy for the generalized LR parsing. There is no need to make subprocesses synchronize during parsing. Stack information is sent to each subprocesses from their parent process.

3.1 Brief Introduction of GHC

Before explaining the details of PLR algorithm, we will give a brief introduction of GHC. Typical GHC statements are given in fig.3. Roughly speaking, the vertical bar in a GHC statement of fig.3 works as a cut symbol of Prolog. When a goal 'a' is executed. a process corresponding to the statement (1) is activated and the body becomes a new goal in which 'b' and 'c' are excuted simultaneously, since GHC adopts AND-parallel strategy. In other word, subprocesses 'b' and 'c' are created by a parent process 'a' and they run in paralell. Although GHC has a few of synchronization mechanisms, it will not be necessary for you to understand them.

3.2 Description of PLR Algorithm

At first, PLR creates a list of preterminals of an input sentence which will be parsed. PLR parser begins activating an action process which corresponds to the LR table entry determined by the state '0' and the first preterminal in the preterminal list. The action process activates the other processes according to the comands specified in the LR talbe entry. Activated processes recieves stack information from the parent process and also perform some comands specified in the corresponding LR table entry. The process activation will continue until some processes find out an 'acc' or an 'error' entry. If we have a conflict during parsing, more than two subprocesses will be activated at once and run in parallel. There are three kinds of processes which are activated in PLR parser.

- action process:
  An action process carries out shift and/or reduce operations. In case of a shift operation, the action process pushes a lookahead preterminal on a stack and activates a new process which corresponds to new state given by the shift operation. When an action process encounters a reduce operation, a reduce process will be activated and recieve stack information from the parent action process. If an action process finds a conflict, more than two subprocesses will be activated each of which perform either a shift or a reduce operation. These subprocesses run in parallel. If an action process enconters an 'acc' operation, the action process will extract the result of the parsing and end with success. On the contrary, if all of the above conditions are not satisfied, an action process will end with failure.

- reduce process:
  Using a grammar rule specified by a reduce operation, the reduce process makes a reduction of an appropriate portion of stack, and the reduce process activates a goto process in order to enter a new state.

- goto process:
  Using stack information given by a reduce process, a goto process activates an action process to enter a new state.

In the following subsections, we will give the GHC definition of PLR processes obtained by the LR table shown in fig 2.

3.3 Definition of Action Process

Followings are examples of definitions of an action process.

1. Suppose an action process 'i0' that corresponds to the entry in fig.2 whose row and column are 0 and 'noun' respectively. As the entry contains 'sh 1', the process has to activate a subprocess which carries out a shift operation. The definition of the process 'i0' is shown below.

   ```
   i0(noun, Stack, [noun,NextCat|List], Info) :- true |
           i1(NextCat, [[1,noun]|Stack], [NextCat|List], Info).
   ```

   In the above process definition, the predicate 'i0' is a process name, and its first argument is a lookahead preterminal 'noun'. The second argument is 'Stack' on which information about state, grammatical categories and the other information are pushed. The third is a list of preterminals of an input sentence. The fourth outputs 'Info', the results of parsing. The subprocess 'i1' is activated and carries out a 'sh 1' operation. The subprocess 'i1' recieves a new stack which consists of 'Stack', state '1' and a preterminal 'noun'. Note that in the third argument of the process 'i1', preterminal 'noun' is eliminated from the list of preterminals, since preterminal 'noun' should be shifted.

2. Consider an entry of state '2' and a lookahead preterminal 'v' in fig.2. The definition of action process 'i2' is given below :

   ```
   i2(v, Stack, List, Info) :- true |
                 re6(v, Stack, List, Info).
   ```

   In the body of an action process 'i2', a subprocess 're6' is activated in order to conduct a reduce operation. The subprocess 're6' recieves the same stack information and a preterminal list as those of the parent process 'i2'. The detail of the reduce process will be explained later.

3. Consider an entry of state '14' and a lookahead preterminal 'p' in fig.2. We will find out a shift-reduce conflict, 'sh 7/re 10'. The definition of an action process 'i14' is as follows.

   ```
   i14(p, Stack, [p,NextCat|List], Info) :- true |
                 i7(NextCat, [[7,p]|Stack], [NextCat|List], Info1),
                 re10(p, Stack, [p,NextCat|List], Info2),
                 merge(Info1, Info2, Info).
   ```

In the body of the process 'i14', both subprocesses 'i7' and 're10' carry out a shift and a reduce operation simultaneously. The 'merge' process is a built-in process which merges the output produced by the subprocesses 'i7' and 're10'.

4. Consider the entry of state '4' and a lookahead preterminal '$' in fig.2. We will find out 'acc' in the entry which indicates a success of parsing. The definition of the action process 'i4' is as follows.

```
i4($, [[_,Result]|_], _, Info) :- true |
                        Info=[Result].
```

In the body of the action process 'i4','[Result]' is sent to the fourth argument 'Info', and finally the action process 'i4' terminates with success.

5. If no operation is specfied in an entry, an error handling process has to be activated. We have to define an error handling process in some states if necessary. The following is a definition of an error process in state '0' which should be placed at the end of definitions of the process 'i0'.

```
otherwise.
i0(_, _, _, Info) :- true |
                Info=[].
```

The statement 'otherwise' is a built-in statement which declares that GHC statements below 'otherwise' should be executed after all GHC stetements before 'ohterwise' fails.

3.4 Definition of Reduce Process

The following definition of a reduce process 're10' is an example of reduce actions correspondds to the grammar rule numbered 10 in fig.1((10) PP→p,NP).

```
re10(NextCat, OldStack, List, Info) :-
            OldStack=[[_,T1],[_,T2],[State,T3]|Tail]|
            pp(State, NextCat, [pp,T2,T1], [[State,T3]|Tail], List, Info).
```

In the second argument of 're10', the topmost two elements of 'OldStack' are popped and sent to a goto process 'pp' in which the third argument '[pp,T2,T1]' constructs a syntactic tree whose root is 'pp' in accordance with the grammar rule 10 in fig.1. The name of the goto process 'pp' is the name of the lefthand side nonterminal symbol in the grammar rule 10. The first argument 'State' is a new state number extracted from 'OldStack'. Note that the reduce process 're10' passes a next incomming preterminal 'NextCat' to the 'pp' process, since a reduce process does not consume any incomming preterminals.

3.5 Definition of Goto Process

After a reduce operation is carried out, a goto process is activated in order to enter a new state in which a new action process will be activated. At that time, the goto process uses both an incomming nonterminal symbol and a state number on the top of the stack.

We will give a sample definition of goto processes.

```
    s(0, NextCat, Tree, Stack, List, Info) :- true |
                i4(NextCat, [[4,Tree]|Stack], List, Info).
```

The process 's' defined above is activated after 's' is constructed by a reduce process in state '0'. As the entry of row '0' and column 's' in the LR table of fig.2 includes '4', the goto process 's' activates an action process 'i4' pushing state '4' and tree information onto the stack.

3.6 An Example of PLR Parsing

Given a LR table of fig.2, a translater generates the following definitions of parsing processes.

```
i0(det,Stack,[_,NextCat|List],Info):- true|
        i1(NextCat,[[1,det]|Stack],[NextCat|List],Info).

i0(noun,Stack,[_,NextCat|List],Info):- true|
        i2(NextCat,[[2,n]|Stack],[NextCat|List],Info).

i0(pron,Stack,[_,NextCat|List],Info):- true|
        i3(NextCat,[[3,pron]|Stack],[NextCat|List],Info).

otherwise.
i0(_,_,_,Info):- true|Info=[].

i1(noun,Stack,[_,NextCat|List],Info):- true|
        i6(NextCat,[[6,noun]|Stack],[NextCat|List],Info).

        . . . . . . . . . . . . . . . . . . . . . . . .
```

Following is an example of PLR parsing.

```
input sentence : i open the door with a key .

Parsing begins with activating the following action process 'i0'.
 i0(pron,[[0,[]]],[pron,v,det,noun,p,det,noun,$],Info)
  ^        ^Stack   ^List of Preterminal
  Lookahead

Activates the action process 'i3' for 'shift 3'.
 i3(v,[[3,pron],[0,[]]],[v,det,noun,p,det,noun,$],Info)

Activates the reduce process 're7' for 'reduce 7'.
 re7(v,[[3,pron],[0,[]]],[v,det,noun,p,det,noun,$],Info)
       [[3,pron],[0,[]]]=[[_,T1],[State,T2]|Tail]
Activates the goto process 'np'.
 np(0,v,[np,pron],[[0,[]]],[v,det,noun,p,det,noun,$],Info)
```

```
    ^State ^Tree    ^Stack

Activates the action process 'i5' for 'goto 5'.
 i5(v,[[5,[np,pron]],[0,[]]],[v,det,noun,p,det,noun,$],Info)

Activates the action process 'i10' for 'shift 10'.
 i10(det,[[10,v],[5,[np,pron]],[0,[]]],[det,noun,p,det,noun,$],Info)

          . . . . . . . . . . . . . . . . . . . .

 i16(p,[[16,[np,det,noun]],[10,v],[5,[np,pron]],[0,[]]],[p,det,noun,$],Info)
A conflict 'shift 7/reduce 8' occures.
Activates 'i7' and 're8' processes simultaneously.

 i7(det,[[7,p]|[[16,[np,det,noun]],[10,v],[5,[np,pron]],[0,[]]]],[det|[n,$]],Info)
 re8(p,[[16,[np,det,noun]],[10,v],[5,[np,pron]],[0,[]]],[p,det|[noun,$]],Info)

          . . . . . . . . . . . . . . . . . . . .

Both processes end with success and produce the following results in 'Info'.
 i4($,[[4,[s,[np,pron],[vp,v,[np,[np...],[pp,p,[np...]]]]]],[0,[]]],[$],Info)
    Info=[s,[np,pron],[vp,v,[np,[np,det,noun],[pp,p,[np,det,noun]]]]]

 i4($,[[4,[s,[s,[np,pron],[vp,v,[np,[np...]]]],[pp,p,[np...]]]],[0,[]]],[$],Info)
    Info=[s,[s,[np,pron],[vp,v,[np,[np,det,noun]]]],[pp,p,[np,det,noun]]]
```

4 The Results of A Experiment

We conducted an experiment to parse many English sentences with many PP attachments such as :

NP,v,NP
NP,v,NP,PP
NP,v,NP,PP,PP
NP,v,NP,PP,PP,PP
. . . .

In the experiment, PLR and PAX are used to parse sentences. The number enclosed by parenthesis in fig.4 indicates the number of parsing trees. PLR runs 1.4 times faster than PAX that was known as the best parallel parser in the past. In order to get all parsing trees of a sentence with 9 PP attachments, PLR takes about 65 sec. on Sun-3/260 workstation. It means that PLR produces a parsing tree only every 4 msec.

The reader should note that the PLR which we explained in this paper does not use a graph structured stack. For comparison, the results of parsing which makes uses of the graph structured stack is shown by a solid line. The PLR parser with a graph structured stack runs 10 times slower than the one without a graph structured stack. The reason is that the former

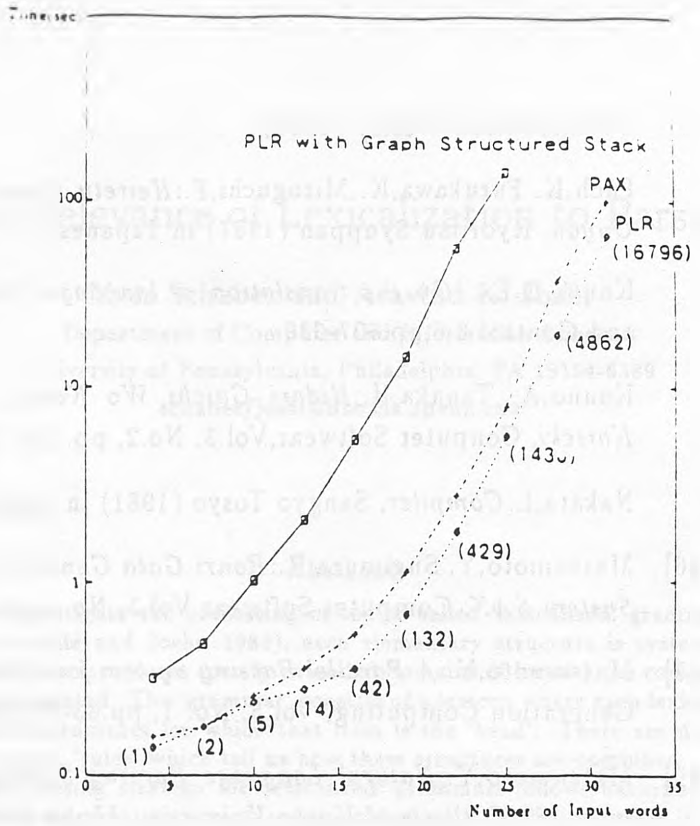


fig.4: The result of Parsing time

causes many processes to wait for synchronization. We are now considering the reason why PLR parser without the graph structured stack runs so fast. One of the reasons is that PLR parser without the graph does not cause many processes to suspend for synchronizations.

5 Conclusion

We described an exaple of the implementation of the PLR algorithm in GHC in which each entry of the LR table is regarded as a process which handles shift and reduce operations. When a conflict occurs in an entry of the LR table, the corresponding parsing process activates two or more subprocesses which run in parallel and simulate breadth first strategy of the generalized LR parsing. Each subprocess is given the stack information from the parent process and runs further to execute a shift and a reduce operation.

The experiment has revealed that PLR runs faster than PAX that has been known as the best parallel parser. PLR runs so fast that it will be a promising parser for processing many complex natural language sentences.

However, PLR has many problems to be solved, for example, handling of gapping and idiom, and integration of syntactic and semantic processing which are urgent problems to be solved in the near future.

References

[Aho 72] Aho,A.V.and Ulman,J.D.: *The Theory of Parsing,Translation,and Compiling,* Prentice-Hall,Englewood Cliffs,New Jersey (1972)

[Aho 85] Aho,A.V.,Senthi,R.and Ulman,J.D.: *Compilers Principles,Techniques,and*

Tools,Addison-Wesley (1985)

[Fuchi 87] Fuch,K. Furukawa,K. Mizoguchi,F.:*Heiretu Ronri Gata Gengo GHC To Sono Ouyou*, Kyoritsu Syuppan (1987) in Japanese

[Knuth 65] Knuth,D.E.: *On the translation of languages from left to right*,Information and Control 8:6,pp.607-639

[Konno 86] Konno,A. Tanaka,H.:*Hidari Gaichi Wo Kouryo Shita Bottom Up Koubun Kaiseki*, Conputer Softwear,Vol.3, No.2, pp.115-125 (1986) in Japanese

[Nakata 81] Nakata,I.:*Compiler*, Sangyo Tosyo (1981) in Japanese

[Matsumoto 86] Matsumoto,Y. Sugimura,R.:*Ronri Gata Gengo Ni Motodsuku Koubun Kaiseki System SAX*, Computer Softwear,Vol.3, No.4, pp.4-11 (1986) in Japanese

[Matsumoto 87] Matsumoto,Y.:*A Parallel Parsing System for Natural Language Analysis*, New Generation Computing, Vol.5, No. 1, pp.63-78 (1987)

[Matsumoto 89] Matsumoto,Y.:*Natural Language Parsing Systems based on Logic Programming*, Ph.D thesis of Kyoto University, (June 1989)

[Mellish 85] Mellish,C.S.:*Computer Interpretation of Natural Language Descriptions*, Ellis Horwood Limited (1985)

[Nilsson 86] Nilsson,U.: *AID:An Alternative Implementation of DCGs*, New Generation Computing, 4, pp.383-399 (1986)

[Okumura 89] Okumura,M.:*Sizengengo Kaiseki Ni Okeru Imiteki Aimaisei Wo Zoushinteki Ni Kaisyou Suru Keisan Model*, Natural Language Analysis Working Group,Information Processing Society of Japan,NL71-1 (1989) in Japanese

[Pereira 80] Pereira,F.and Warren,D.: Definite Clause Grammar for Language Analysis-A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell*, Vol.13, No.3, pp.231-278 (1980)

[Tokunaga 88] Tokunaga,T. Iwayama,M. Kamiwaki,T. Tanaka,H.:*Natural Language Analysis System LangLAB*, Transactions of Information Processing Society of Japan,Vol.29, No.7, pp.703-711 (1988) in Japanese

[Tomita 86] Tomita,M.:*Efficient Parsing for Natural Language*, Kluwer Academic Publishers (1986)

[Tomita 87] Tomita,M.: *An Efficien Augmented-Context-Free Parsing Algorithm*, Computational Linguistics, Vol.13, Numbers 1-2, pp.31-46 (1987)

[Ueda 85] Ueda,K.:*Guarded Horn Clauses*, Proc. The Logic Programming Conference, Lecture Notes in Computer Science, 221 (1985)

[Uehara 83] Uehara,K. Toyoda,J.: *Sakiyomi To Yosokukinou Wo Motsu Jutugo Ronri Gata Koubun Kaiseki Program : PAMPS*, Transactions of Information Processing Society of Japan, Vol.24, No.4, pp.496-504 (1983) in Japanese

The Relevance of Lexicalization to Parsing*

Yves Schabes and Aravind K. Joshi
Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104-6389
schabes/joshi@linc.cis.upenn.edu

Abstract

In this paper, we investigate the processing of the so-called 'lexicalized' grammar. In 'lexicalized' grammars (Schabes, Abeillé and Joshi, 1988), each elementary structure is systematically associated with a lexical 'head'. These structures specify extended domains of locality (as compared to CFGs) over which constraints can be stated. The 'grammar' consists of a lexicon where each lexical item is associated with a finite number of structures for which that item is the 'head'. There are no separate grammar rules. There are, of course, 'rules' which tell us how these structures are combined.

A general two-pass parsing strategy for 'lexicalized' grammars follows naturally. In the first stage, the parser selects a set of elementary structures associated with the lexical items in the input sentence, and in the second stage the sentence is parsed with respect to this set. We evaluate this strategy with respect to two characteristics. First, the amount of filtering on the entire grammar is evaluated: once the first pass is performed, the parser uses only a subset of the grammar. Second, we evaluates the use of non-local information: the structures selected during the first pass encode the morphological value (and therefore the position in the string) of their 'head'; this enables the parser to use non-local information to guide its search.

We take Lexicalized Tree Adjoining Grammars as an instance of lexicalized grammar. We illustrate the organization of the grammar. Then we show how a general Earley-type TAG parser (Schabes and Joshi, 1988) can take advantage of lexicalization. Empirical data show that the filtering of the grammar and the non-local information provided by the two-pass strategy improve the performance of the parser.

We explain how constraints over the elementary structures expressed by unification equations can be parsed by a simple extension of the Earley-type TAG parser. Lexicalization guarantees termination of the algorithm without special devices such as restrictors.

1 Lexicalized Grammars

Most current linguistic theories give lexical accounts of several phenomena that used to be considered purely syntactic. The information put in the lexicon is thereby increased in both amount and complexity: see, for example, lexical rules in LFG (Kaplan and Bresnan, 1983), GPSG (Gazdar, Klein, Pullum and Sag, 1985), HPSG (Pollard and Sag, 1987), Combinatory Categorial Grammars (Steedman 1985, 1988), Karttunen's version of Categorial Grammar (Karttunen 1986, 1988), some versions of GB theory (Chomsky 1981), and Lexicon-Grammars (Gross 1984).

We say that a grammar is 'lexicalized' if it consists of:[1]

- a finite set of structures associated with each lexical item, which is intended to be the 'head' of these structures; the structures define the domain of locality over which constraints are specified; constraints are local with respect to their lexical 'head';
- an operation or operations for composing the structures.

Notice that Categorial Grammars (as used for example by Ades and Steedman, 1982 and Steedman, 1985 and 1988) are 'lexicalized' according to our definition since each basic category has a lexical item associated with it.

*This work is partially supported by ARO grant DAA29-84-9-007, DARPA grant N0014-85-K0018, NSF grants MCS-82-191169 and DCR-84-10413. We have benefited from our discussions with Anne Abeillé, Lauri Karttunen, Mitch Marcus and Stuart Shieber. We would also like to thank Ellen Hays.

[1] By 'lexicalization' we mean that in each structure there is a lexical item that is realized. We do not mean simply adding feature structures (such as head) and unification equations to the rules of the formalism.

A general two-step parsing strategy for 'lexicalized' grammars follows naturally. In the first stage, the parser selects a set of elementary structures associated with the lexical items in the input sentence, and in the second stage the sentence is parsed with respect to this set. The strategy is independent of the nature of the elementary structures in the underlying grammar. In principle, any parsing algorithm can be used in the second stage.

The first step selects a relevant subset of the entire grammar, since only the structures associated with the words in the input string are selected for the parser. In the worst case, this filtering wou: , select the entire grammar. The number of structures filtered during this pass depends on the nature of the input string and on characteristics of the grammar such as the number of structures, the number of lexical entries, the degree of lexical ambiguity, and the languages it defines.

Since the structures selected during the first step encode the morphological value of their 'head' (and therefore its position in the input string), the first step also enables the parser to use non-local information to guide its search. The encoding of the value of the 'head' of each structure constrains the way the structures can be combined. It seems that this information is particularly useful for parsing algorithms that have some top-down behavior.

This parsing strategy is general and any standard parsing technique can be used in the second step. Perhaps the advantages of the first step could be captured by some other technique. However this strategy is extremely simple and is consistent with the linguistic motivations for lexicalization.

2 Lexicalized TAGs

Not every grammar is in a 'lexicalized' form.[2] In the process of lexicalizing a grammar, we require that the 'lexicalized' grammar produce not only the same language as the original grammar, but also the same structures (or tree set).

For example, a CFG, in general, will not be in a 'lexicalized' form. The domain of locality of CFGs can be easily extended by using a tree rewriting grammar (Schabes, Abeillé and Joshi, 1988) that uses only substitution as a combining operation. This tree rewriting grammar consists of a set of trees that are not restricted to be of depth one (as in CFGs). Substitution can take place only on non-terminal nodes of the frontier of each tree. Substitution replaces a node marked for substitution by a tree rooted by the same label as the node (see Figure 1; the substitution node is marked by a down arrow $\downarrow$).

However, in the general case, CFGs cannot be 'lexicalized', if only substitution is used. Furthermore, in general, there is not enough freedom to choose the 'head' of each structure. This is important because we want the choice of the 'head' for a given structure to be determined on purely linguistic grounds.

If adjunction is used as an additional operation to combine these structures, CFGs can be lexicalized. Adjunction builds a new tree from an auxiliary tree β and a tree α . It inserts an auxiliary tree in another tree (see Figure 1). Adjunction is more powerful than substitution. It can weakly simulate substitution, but it also generates languages that could not be generated with substitution.[3]

Substitution and adjunction enable us to lexicalize CFGs. The 'heads' can be freely chosen (Schabes, Abeillé and Joshi, 1988). The resulting system now falls in the class of mildly context-sensitive languages (Joshi, 1985). Elementary structures of extended domain of locality combined with substitution and adjunction yield Lexicalized TAGs.

TAGs were first introduced by Joshi, Levy and Takahashi (1975) and Joshi (1985). For more details on the original definition of TAGs, we refer the reader to Joshi (1985), Kroch and Joshi (1985), or Vijay-Shanker (1987). It is known that Tree Adjoining Languages (TALs) are mildly context sensitive. TALs properly contain context-free languages.

[2]Notice the similarity of the definition of 'lexicalized' grammar with the offline parsibility constraint (Kaplan and Bresnan 1983). As consequences of our definition, each structure has at least one lexical item (its 'head') attached to it and all sentences are finitely ambiguous.

[3]It is also possible to encode a context-free grammar with auxiliary trees using adjunction only. However, although the languages correspond, the set of trees do not correspond.

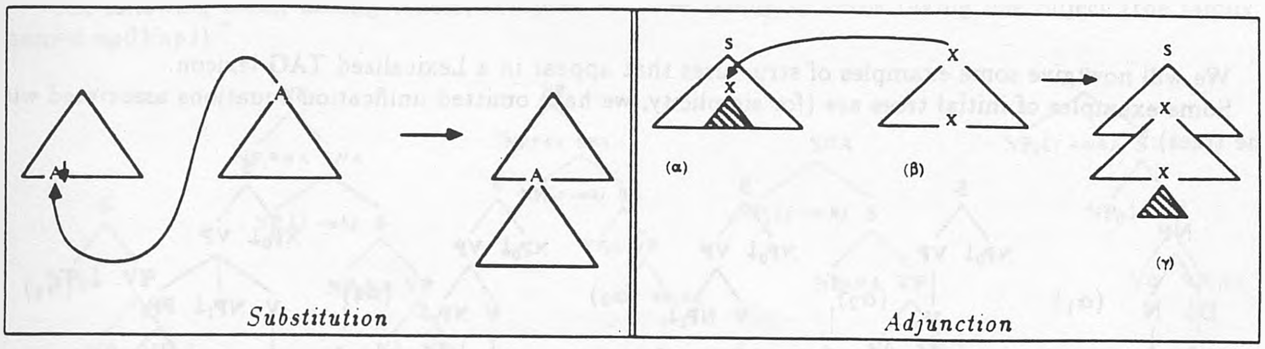


Figure 1: *Combining operations*

TAGs with substitution and adjunction are naturally lexicalized.[4] A Lexicalized Tree Adjoining Grammar is a tree-based system that consists of two finite sets of trees: a set of initial trees, I and a set of auxiliary trees A (see Figure 2). The trees in $I \cup A$ are called **elementary trees**. Each elementary tree is constrained to have at least one terminal symbol which acts as its 'head'.

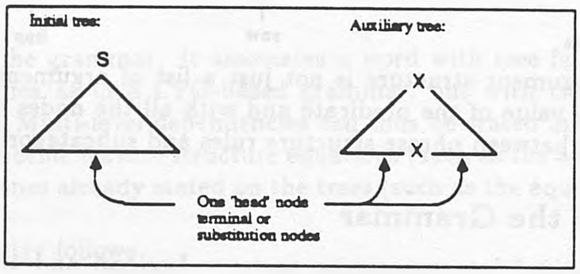


Figure 2: *Schematic initial and auxiliary trees*

The **tree set** of a TAG G, $\mathcal{T}(G)$ is defined to be the set of all derived trees starting from S-type initial trees in I. The **string language** generated by a TAG, $\mathcal{L}(G)$, is defined to be the set of all terminal strings of the trees in $\mathcal{T}(G)$.

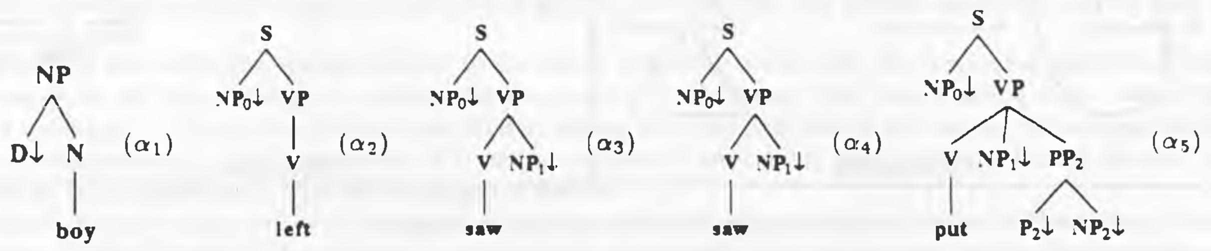
By lexicalizing TAGs, we have associated lexical information to the 'production' system encoded by the TAG trees. We have therefore kept the computational advantages of 'production-like' formalisms (such as CFGs, TAGs) while allowing the possibility of linking them to lexical information. Formal properties of TAGs hold for Lexicalized TAGs.

As first shown by Kroch and Joshi (1985), the properties of TAGs permit us to encapsulate diverse syntactic phenomena in a very natural way. TAG's extended domain of locality and its factoring recursion from local dependencies lead, among other things, to localizing the so-called unbounded dependencies. Abeillé (1988a) uses the distinction between substitution and adjunction to capture the different extraction properties between sentential subjects and complements. Abeillé (1988c) makes use of the extended domain of locality and lexicalization to account for NP island constraint violations in light verb constructions; in such cases, extraction out of NP is to be expected, without the use of reanalysis. The relevance of Lexicalized TAGs to idioms has been suggested by Abeillé and Schabes (1989).

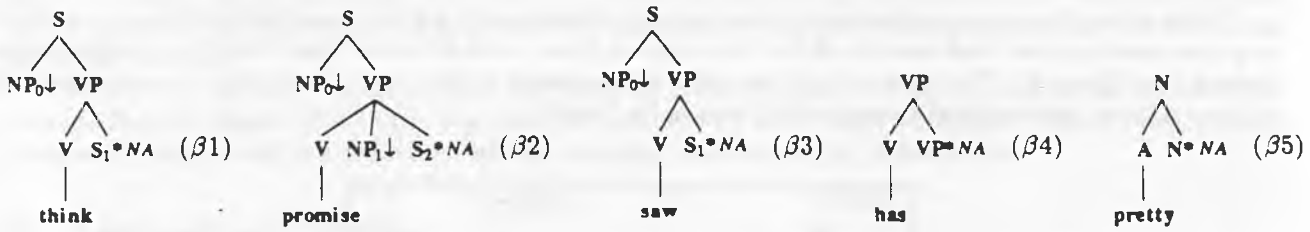
[4] In some earlier work of Joshi (1969, 1973), the use of the two operations 'adjoining' and 'replacement' (a restricted case of substitution) was investigated both mathematically and linguistically. However, these investigations dealt with string rewriting systems and not tree rewriting systems.

We will now give some examples of structures that appear in a Lexicalized TAG lexicon.

Some examples of initial trees are (for simplicity, we have omitted unification equations associated with the trees):[5]



Examples of auxiliary trees (they correspond to predicates taking sentential complements or modifiers):



In this approach, the argument structure is not just a list of arguments. It is the syntactic structure constructed with the lexical value of the predicate and with all the nodes of its arguments that eliminates the redundancy often noted between phrase structure rules and subcategorization frames.[6]

2.1 Organization of the Grammar

A Lexicalized TAG is organized into two major parts: a **lexicon** and **tree families**, which are sets of trees. Although it is not necessary to separate trees from their realization in the lexicon, we chose to do so in order to capture some generalities about the structures. TAG's factoring recursion from dependencies, the extended domain of locality of TAGs, and lexicalization of elementary trees make Lexicalized TAG an interesting framework for grammar writing. Abeillé (1988b) discusses the writing of a Lexicalized TAG for French. Bishop, Cote and Abeillé (1989) similarly discuss the writing of a Lexicalized TAG grammar for English.

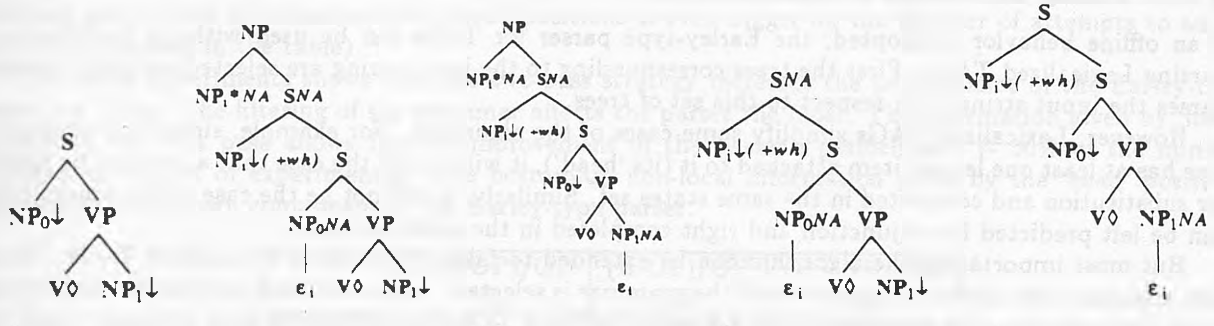
2.1.1 Tree Families

A **tree family** is essentially a set of sentential trees sharing the same argument structure abstracted from the lexical instantiation of the 'head' (verb, predicative noun or adjective). Because of the extended domain of locality of Lexicalized TAG, the argument structure is not stated by a special mechanism but is implicitly stated in the topology of the trees in a tree family. Each tree in a family can be thought of as all possible syntactic 'transformations' of a given argument structure. Information (in the form of feature structures) that is valid independent of the value of the 'head' is stated on the tree of the tree family. For example, the agreement between the subject and the main verb or auxiliary verb is stated on each tree of the tree family. Currently, the trees in a family are explicitly enumerated.

[5]The trees are simplified and the feature structures on the trees are not displayed. ↓ is the mark for substitution nodes. * is the mark for the foot node of an auxiliary tree and NA stands for null adjunction constraint. This is the only adjunction constraint not indirectly stated by feature structures. We put indices on some non-terminals to express syntactic roles (0 for subject, 1 for first object, etc.). The index shown on the empty string (ϵ) and the corresponding filler in the same tree is for the purpose of indicating the filler-gap dependency.

[6]Optional arguments are stated in the structure.

The following trees, among others, compose the tree family of verbs taking one object (the family is named *np0Vnp1*):[7]



(*αnp0Vnp1*) (*βR0np0Vnp1*) (*βR1np0Vnp1*) (*αW0np0Vnp1*) (*αW1np0Vnp1*)

αnp0Vnp1 is an initial tree corresponding to the declarative sentence, *βR0np0Vnp1* is an auxiliary tree corresponding to a relative clause where the subject has been relativized, *βR1np0Vnp1* corresponds to the relative clause where the object has been relativized, *αW0np0Vnp1* is an initial tree corresponding to a wh-question on the subject, *αW1np0Vnp1* corresponds to a wh-question on the object.

2.1.2 The Lexicon

The **lexicon** is the heart of the grammar. It associates a word with tree families or trees. Words are not associated with basic categories as in a CFG-based grammar, but with tree-structures corresponding to minimal linguistic structures. Multi-level dependencies can thus be stated in the lexicon.

It also states some word-specific feature structure equations (such as the agreement value of a given verb) that have to be added to the ones already stated on the trees (such as the equality of the value of the subject and verb agreements).

An example of a lexical entry follows:

```
loves, V : np0Vnp1 {VP.b:<mode>=ind,
                    VP.t:<agr pers>= 3,
                    VP.t:<agr num>= singular,
                    VP.t:<tense>=present}.
```

It should be emphasized that in our approach the category of a word is not a non-terminal symbol but a multi-level structure corresponding to minimal linguistic structures: sentences (for predicative verbs, nouns and adjectives) or phrases (NP for nouns, AP for adjectives, PP for prepositions yielding adverbial phrases).

2.2 Parsing Lexicalized TAGs

An Earley-type parser for TAGs has been developed by Schabes and Joshi (1988). It is a general TAG parser. It handles adjunction and substitution. It can take advantage of lexicalization. It uses the structures selected after the first pass to parse the sentence. The parser is able to use the non-local information given by the first step to filter out prediction and completion states. It is extended to deal with feature structures for TAGs as defined by Vijay-Shanker and Joshi (1988). The extended algorithm we propose always terminates when used on Lexicalized TAGs without special devices such as restrictors. Unification equations are associated with both extended linguistic structures and lexical information given by the 'head'. This representation allows a more natural and more direct statement of unification equations.

[7]The trees are simplified. ◊ is the mark for the node under which the 'head' word of the tree is attached.

2.2.1 Taking Advantage of Lexicalization

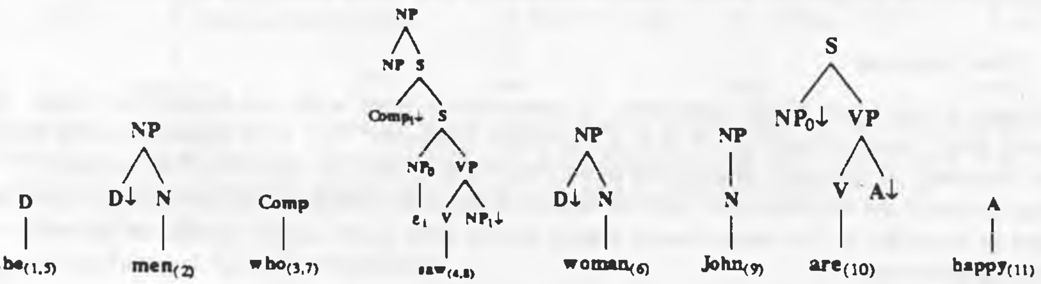
If an offline behavior is adopted, the Earley-type parser for TAGs can be used with no modification for parsing Lexicalized TAGs. First the trees corresponding to the input string are selected and then the parser parses the input string with respect to this set of trees.

However, Lexicalized TAGs simplify some cases of the algorithm. For example, since by definition each tree has at least one lexical item attached to it (its 'head'), it will not be the case that a tree can be predicted for substitution and completed in th- same states set. Similarly, it will not be the case that an auxiliary tree can be left predicted for adjunction and right completed in the same states set.

But most importantly the algorithm can be extended to take advantage of Lexicalized TAGs. Once the first pass has been performed, a subset of the grammar is selected. Each structure encodes the morphological value (and therefore the positions in the string) of its 'head'. Identical structures with different 'head' values are merged together (by identical structures we mean identical trees and identical information, such as feature structures, stated on those trees).[8] This enables us to use the 'head' position information while processing efficiently the structures. For example, given the sentence

`The` $_1$ `men` $_2$ `who` $_3$ `saw` $_4$ `the` $_5$ `woman` $_6$ `who` $_7$ `saw` $_8$ `John` $_9$ `are` $_{10}$ `happy` $_{11}$

the following trees (among others) are selected after the first pass:[9]



The trees for **men** and for **woman** are distinguished since they carry different agreement feature structures (not shown in the figure).

Notice that there is only one tree for the relative clauses introduced by **saw** but that its 'head' position can be 4 or 8. Similarly for **who** and **the**.

The 'head' positions of each structure impose constraints on the way that the structures can be combined (the 'head' positions must appear in increasing order in the combined structure). This helps the parser to filter out predictions or completions for adjunction or substitution. For example, the tree corresponding to **men** will not be predicted for substitution in any of the trees corresponding to **saw** since the 'head' positions would not be in the right order.

We have been evaluating the influence of the filtering of the grammar and the 'head' position information on the behavior of the Earley-type parser. We have conducted experiments on a feature structure-based Lexicalized English TAG whose lexicon defines 200 entries associated with 130 different elementary trees.[10] Twenty five sentences of length ranging from 3 to 14 words were used to evaluate the parsing strategy. For each experiment, the number of trees given to the parser and the number of states were recorded.

In the first experiment (referred to as *one pass*, *OP*), no first pass was performed. The entire grammar (i.e., the 130 trees) was used to parse each sentence. In the second experiment (referred to as *two passes no 'head'*, *NH*), the two-pass strategy was used but the 'head' positions were not used in the parser. And in the third experiment (referred to as *two passes with 'head'*, *H*), the two-pass strategy was used and the information given by the 'head' positions was used by the parser.

The average behavior of the parser for each experiment is given in Figure 3. The first pass filtered on average 85% (always at least 75%) of the trees. The filtering of the grammar by itself decreased by 86% the

[8] Unlike our previous suggestions (Schabes, Abeillé and Joshi, 1988), we do not distinguish each structure by its 'head' position since it increases unnecessarily the number of states of the Earley parser. By factoring recursion, the Earley parser enables us to process only once parts of a tree that are associated with several lexical items selecting the same tree. However, if termination is required for a pure top-down parser, it is necessary to distinguish each structure by its 'head' position.

[9] The example is simplified to illustrate our point.

[10] The trees are differentiated by their topology and their feature structures but not by their 'head' value.

number of states ($(NH - OP)/OP$). The additional use of the information given by the 'head' positions further decreased by 50% ($(H - NH)/NH$) the number of states. The decrease given by the filtering of the grammar and by the information of the head positions is even bigger on the number of attempts to add a state (not reported in the table).[11]

This set of experiments shows that the two-pass strategy increases the performance of the Earley-type parser for TAGs. The filtering of the grammar affects the parser the most. The information given by 'head' p.sition in the first pass allows further improvement of the parser's performance (- 50% of the number of states on the set of experiments). The bottom-up non-local information given by the 'head' positions improves the top-down component of the Earley-type parser.

	(NH-OP)/OP (%)	(H-OP)/OP (%)	(H - NH)/NH (%)
# trees	-85	-85	0
# states	-86	-93	-50

Figure 3: *Empirical evaluation of the two-pass strategy*

We performed our evaluation on a relatively small grammar and we did not evaluate the variations across grammars. The lexical degree of ambiguity of each word, the number of structures in the grammar, the number of lexical entries, and the length (and nature) of the input sentences are parameters to be considered. Although it might appear easy to conjecture the influence of these parameters, the actual experiments are difficult to perform since statistical data on these parameters are hard to obtain. We hope to perform some limited experiments along those lines.

2.3 Parsing Feature-Based TAGs

As defined by Vijay-Shanker (1987) and Vijay-Shanker and Joshi (1988), to each adjunction node in an elementary tree two feature structures are attached: a top and a bottom feature structure.[12] When the derivation is completed, the top and bottom features of all nodes are unified simultaneously. If the top and bottom features of a node do not unify, then a tree must be adjoined at that node. This definition can be easily extended to substitution nodes. To each substitution node we attach one feature structure which acts as a top feature. The updating of feature structures in the cases of adjunction and substitution is shown in Figure 4.

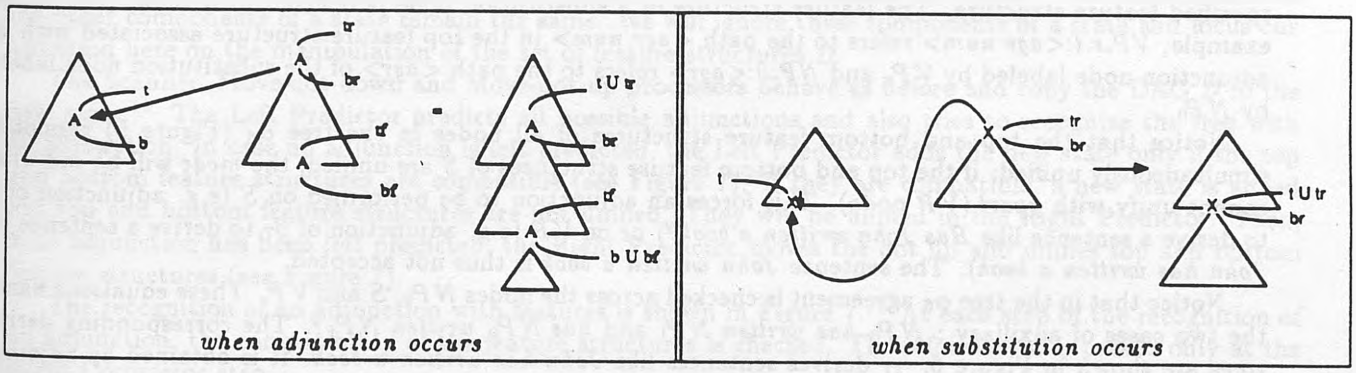
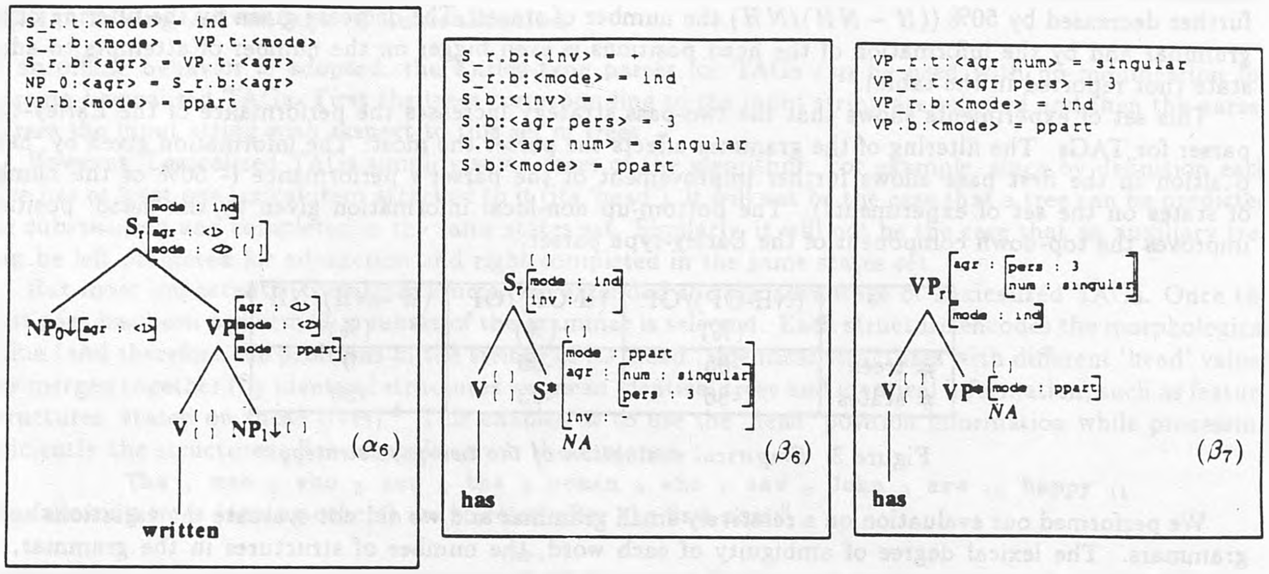


Figure 4: *Updating of feature structures*

[11] A state is effectively added to a states set if it does not exist in the set already.

[12] The top feature structure corresponds to a view to the top of the tree from the node. The bottom feature corresponds to the view to the bottom.



```
S_r.t:<mode> = ind
S_r.b:<mode> = VP.t:<mode>
S_r.b:<agr> = VP.t:<agr>
NP_0:<agr> = S_r.b:<agr>
VP.b:<mode> = ppart
```

```
S_r.b:<inv> = +
S_r.b:<mode> = ind
S.b:<inv> = -
S.b:<agr pers> = 3
S.b:<agr num> = singular
S.b:<mode> = ppart
```

```
VP_r.t:<agr num> = singular
VP_r.t:<agr pers> = 3
VP_r.b:<mode> = ind
VP.b:<mode> = ppart
```

Figure 5: *Examples of unification equations*

2.3.1 Unification Equations

As in PATR-II (Shieber, 1984, 1986), we express with unification equations dependencies between DAGs[13] in an elementary tree. The extended domain of locality of TAGs allows us to state unification equations between features of nodes that may not necessarily be at the same level.

The system consists of a TAG and a set of unification equations on the DAGs associated with nodes in elementary trees.

An example of the use of unification equations in TAGs is given in Figure 5.[14]

Notice that coindexing may occur between feature structures associated with different nodes in the tree. Top or bottom features of a node are referred to by a node name (e.g. S_r)[15] followed by *.t* (for top) or *.b* (for bottom). The semicolon states that the following path specified in angle brackets is relative to the specified feature structure. The feature structure of a substitution node is referred to without *.t* or *.b*. For example, *VP_r.t:<agr num>* refers to the path *<agr num>* in the top feature structure associated with the adjunction node labeled by VP_r and *NP_0:<agr>* refers to the path *<agr>* of the substitution node labeled by NP_0.

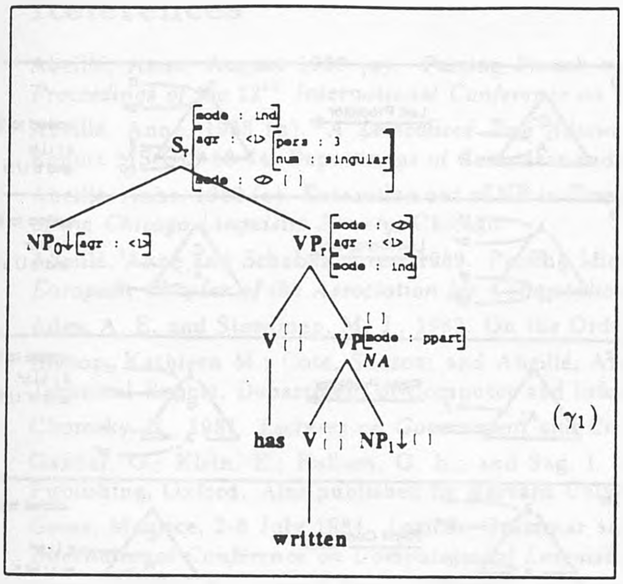
Notice that the top and bottom feature structures of all nodes in the tree α_6 (Figure 5) cannot be simultaneously unified: if the top and bottom feature structures of S are unified, the *mode* will be *ind* which cannot unify with *ppart* (VP node). This forces an adjunction to be performed on S (e.g. adjunction of β_6 to derive a sentence like *Has John written a book?*) or on VP (e.g. adjunction of β_7 to derive a sentence like *John has written a book*). The sentence *John written a book* is thus not accepted.

Notice that in the tree α_6 agreement is checked across the nodes NP_0, S and VP. These equations handle the two cases of auxiliary : NP_0 *has written* NP_1 and *has* NP_0 *written* NP_1*?*. The corresponding derived trees are shown in Figure 6. γ_1 derives sentences like *John has written a book*. It is obtained by adjoining β_7 on the VP node in α_6. γ_2 derives sentences like *Has John written a book?*. It is obtained by adjoining β_6 on the S node in α_6. The obligatory adjunction imposed by the mode feature structure has disappeared in the derived trees γ_1 and γ_2. However, to be completed, γ_1 and γ_2 need NP-trees to be substituted in the nodes labeled by NP (e.g. *John* and *a book*).

[13]Directed Acyclic Graphs which represent the feature structures.

[14]In these examples we have merged the information stated on the trees and in the lexicon. We write unification equations above the tree to which they apply. We have also printed to the right of each node the matrix representation of the top and bottom feature structures.

[15]We implicitly require that each node have a unique name in an elementary tree. If necessary, subscripts differentiate nodes of the same category.



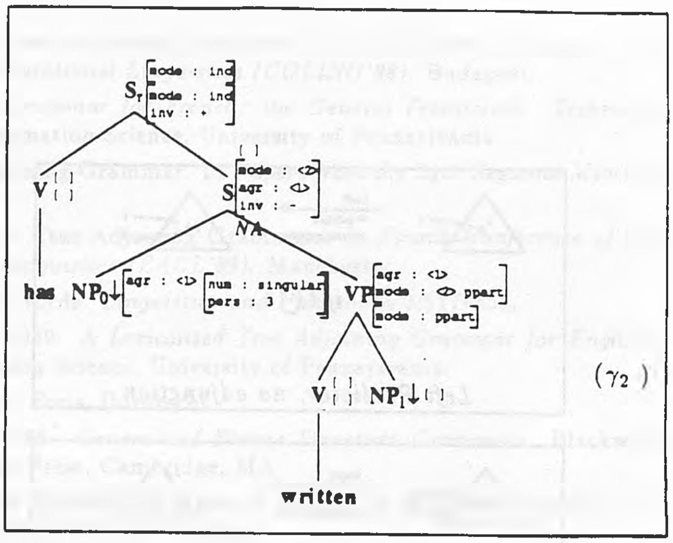


Figure 6: NP_0 *has written* NP_1 and *Has* NP_0 *written* NP_1*?*

2.3.2 Extension to the Earley-type Parser

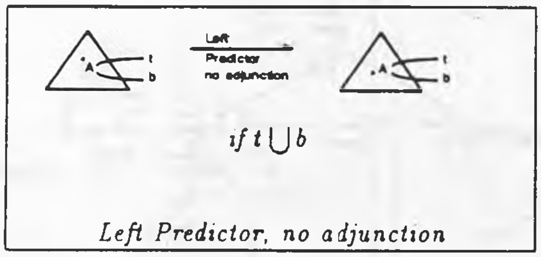
The Earley-type algorithm for TAGs (Schabes and Joshi, 1988) can be extended to parse Lexicalized TAG with unification equations on elementary trees. The extension is similar to the one proposed by Shieber (1985) in order to parse the PATR-II formalism but it does not require the use of restrictors. For the recognition of a substituted tree, we choose to check that unification constraints are compatible at the prediction step and we pass information only at the completion step. For the recognition of an adjunction, we choose to check only that unification constraints are compatible at the Left Predictor, Left Completor and Right Predictor steps and we pass information only at the Right Completor step.

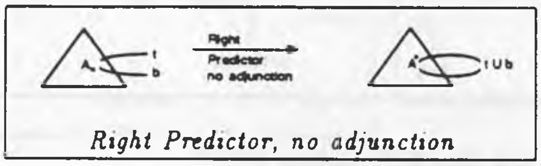
What follows is an informal explanation of the extension to the Earley-type parser. A new component D is added to the states manipulated by the Earley-type parser. D specifies the feature structures associated with each node of the tree represented by the state. It is a set of feature structures. The manipulation of the other components of a state remain the same. We will ignore these components of a state and focus our attention here on the manipulation of the set of feature structures D.

The Scanner, Move-dot-down and Move-dot-up processors behave as before and copy the DAG D to the new state.[16] The Left Predictor predicts all possible adjunctions and also tries to recognize the tree with no adjunction. In case no adjunction is left predicted, the Left Predictor adds the new state only if the top and bottom feature structures are compatible (see Figure 7). If they are compatible, a new state is added but top and bottom feature structures are not unified. They will be unified in the Right Predictor. Then, if no adjunction has been left predicted, the Right Predictor moves the dot up and unifies top and bottom feature structures (see Figure 7).

The recognition of an adjunction with features is shown in Figure 7.[17] At each step of the recognition of an adjunction, the compatibility of the feature structures is checked. The information is passed only at the Right Completor step.

[16] Identical states have identical components, identical feature structures D.
[17] A substituted tree is recognized in a similar way and is not explained here.





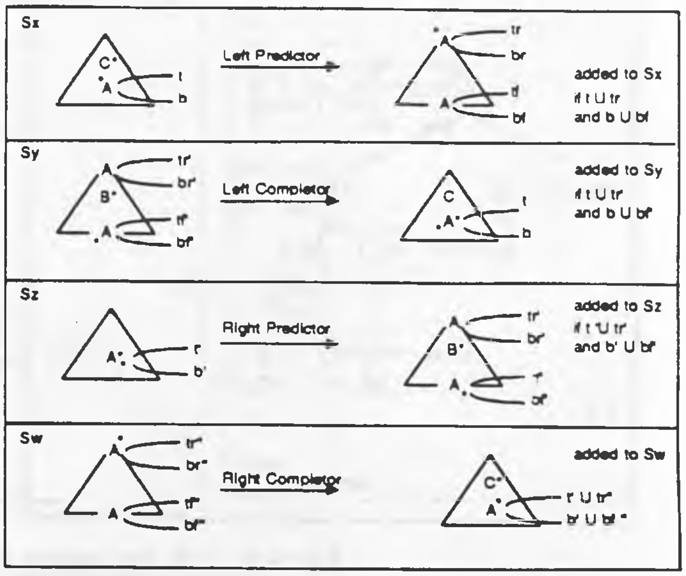


Figure 7: *No Adjunction* *Recognition of an adjunction*

For non-lexicalized TAGs, this approach does not guarantee termination of the algorithm (for similar reasons as for CFG-based unification grammar, Shieber, 1985). However for Lexicalized TAGs, even when recursion occurs, the termination of the algorithm is guaranteed since the recognition of a tree entails the recognition of at least one input token (its 'head') and since information is passed only when a tree is completely recognized. If information were passed before the Right Completor step (in case of adjunction), restrictors (as defined by Shieber, 1985) can be used to guarantee termination. However we believe that in practice (for the Lexicalized TAGs for French and English) passing information at an earlier step than the Right Completor step does not improve the performance.

3 Conclusion

In 'lexicalized' grammars, each elementary structure is systematically associated with a lexical 'head'. These structures specify extended domains of locality (as compared to the domain of locality in CFGs) over which constraints can be stated. The 'grammar' consists of a lexicon in which each lexical item is associated with a finite number of structures for which that item is the 'head'.

Lexicalized grammars suggest a natural two-step parsing strategy. The first step selects the set of structures corresponding to each word in the sentence. The second step tries to combine the selected structures.

We take Lexicalized TAGs as an instance of lexicalized grammar. We illustrate the organization of the grammar. Then we show how the Earley-type parser can take advantage of the two-step parsing strategy. Experimental data show that its performance is thereby drastically improved. The first pass not only filters the grammar used by the parser to produce a relevant subset but also enables the parser to use non-local bottom-up information to guide its search. Finally, we explain how constraints over these structures expressed by unification equations can be parsed by a simple extension of this algorithm. Lexicalization guarantees termination of the algorithm without a special mechanism such as the use of restrictors.

The organization of lexicalized grammars, the simplicity and effectiveness of the two-pass strategy (some other technique would perhaps achieve similar results) seem attractive from a linguistic point of view and for processing. We are currently exploring the possibility of extending this approach to Categorial Grammars.

References

Abeillé, Anne, August 1988 (a). Parsing French with Tree Adjoining Grammar: some Linguistic Accounts. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88)*. Budapest.

Abeillé, Anne, 1988 (b). *A Lexicalized Tree Adjoining Grammar for French: the General Framework.* Technical Report MS-CIS-88-64, Department of Computer and Information Science, University of Pennsylvania.

Abeillé, Anne, 1988 (c). Extraction out of NP in Tree Adjoining Grammar. In *Papers from the 24th Regional Meeting of the Chicago Linguistic Society.* Chicago.

Abeillé, Anne and Schabes, Yves, 1989. Parsing Idioms in Tree Adjoining Grammars. In *Fourth Conference of the European Chapter of the Association for Computational Linguistics (EACL'89).* Manchester.

Ades, A. E. and Steedman, M. J., 1982. On the Order of Words. *Linguistics and Philosophy* 3:517–558.

Bishop, Kathleen M.; Cote, Sharon; and Abeillé, Anne, 1989. *A Lexicalized Tree Adjoining Grammar for English.* Technical Report, Department of Computer and Information Science, University of Pennsylvania.

Chomsky, N., 1981. *Lectures on Government and Binding.* Foris, Dordrecht.

Gazdar, G.; Klein, E.; Pullum, G. K.; and Sag, I. A., 1985. *Generalized Phrase Structure Grammars.* Blackwell Publishing, Oxford. Also published by Harvard University Press, Cambridge, MA.

Gross, Maurice, 2-6 July 1984. Lexicon-Grammar and the Syntactic Analysis of French. In *Proceedings of the 10th International Conference on Computational Linguistics (COLING'84).* Stanford.

Joshi, Aravind K., August 1969. Properties of Formal Grammars with Mixed Type of Rules and their Linguistic Relevance. In *Proceedings of the International Conference on Computational Linguistics.* Sanga Saby.

Joshi, Aravind K., 1973. A Class of Transformational Grammars. In M. Gross, M. Halle and Schutzenberger, M.P. (editors), *The Formal Analysis of Natural Languages.* Mouton, La Hague.

Joshi, Aravind K., 1985. How Much Context-Sensitivity is Necessary for Characterizing Structural Descriptions—Tree Adjoining Grammars. In Dowty, D.; Karttunen, L.; and Zwicky, A. (editors), *Natural Language Processing—Theoretical, Computational and Psychological Perspectives.* Cambridge University Press, New York. Originally presented in a Workshop on Natural Language Parsing at Ohio State University, Columbus, Ohio, May 1983.

Joshi, A. K.; Levy, L. S.; and Takahashi, M., 1975. Tree Adjunct Grammars. *J. Comput. Syst. Sci.* 10(1).

Kaplan, R. and Bresnan [illegible], 1983. Lexical-functional Grammar: A Formal System for Grammatical Representation. In Bresnan, J. (editor), *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge MA.

Karttunen, Lauri, 1986. *Radical Lexicalism.* Technical Report CSLI-86-68, CSLI, Stanford University. To also appear in *New Approaches to Phrase Structures,* University of Chicago Press, Baltin, M. and Kroch A., Chicago, 1988.

Kroch, A. and Joshi, A. K., 1985. *Linguistic Relevance of Tree Adjoining Grammars.* Technical Report MS-CIS-85-18, Department of Computer and Information Science, University of Pennsylvania.

Pollard, Carl and Sag, Ivan A., 1987. *Information-Based Syntax and Semantics. Vol 1: Fundamentals.* CSLI.

Schabes, Yves and Joshi, Aravind K., June 1988. An Earley-Type Parsing Algorithm for Tree Adjoining Grammars. In 26th *Meeting of the Association for Computational Linguistics (ACL'88).* Buffalo.

Schabes, Yves; Abeillé, Anne; and Joshi, Aravind K., August 1988. Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88).* Budapest.

Shieber, Stuart M., July 1984. The Design of a Computer Language for Linguistic Information. In 22nd *Meeting of the Association for Computational Linguistics (ACL'84).* Stanford.

Shieber, Stuart M., July 1985. Using Restriction to Extend Parsing Algorithms for Complex-feature-based Formalisms. In 23rd *Meeting of the Association for Computational Linguistics (ACL'85).* Chicago.

Shieber, Stuart M., 1986. *An Introduction to Unification-Based Approaches to Grammar.* Center for the Study of Language and Information, Stanford, CA.

Steedman, M. J., 1985. Dependency and Coordination in the Grammar of Dutch and English. *Language* 61:523–568.

Steedman, M., 1987. Combinatory Grammars and Parasitic Gaps. *Natural Language and Linguistic Theory* 5:403–439.

Vijay-Shanker, K., 1987. *A Study of Tree Adjoining Grammars.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania.

Vijay-Shanker, K. and Joshi, A.K., August 1988. Feature Structure Based Tree Adjoining Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics (COLING'88).* Budapest.

A Framework for the Development of Natural Language Grammars

Massimo MARINO
Department of Linguistics
University of Pisa
Via S.Maria 36 I-56100 Pisa - ITALY
Electronic Mail: MASSIMOM@ICNUCEVM.BITNET

Abstract

This paper describes a parsing system used in a framework for the development of Natural Language grammars. It is an interactive environment suitable for writing robust NL applications generally. Its heart is the SAIL parsing algorithm that uses a Phrase-Structure Grammar with extensive augmentations. Furthermore, some particular parsing tools are embedded in the system, and provide a powerful environment for developing grammars, even of large coverage.[1]

1. Introduction

Every parsing system should embed a set of tools or mechanisms which should provide an aid in treating a minimum set of linguistic phenomena. Designing SAIL we have mainly taken into account the generality of the parsing system in order to give a wide freedom to the grammar designer, so as to investigate many possible solutions in grammar design in order to adopt the best of them. SAIL (System for the Analysis and Interpretation of Language) is the parsing algorithm of the SAIL Interfacing System (SIS) (/Marino 1988a/, /Marino 1988b/, /Marino 1989/), and just because of its features of generality the design has been driven by some general aspects which derive from various theoretical as well as computational accounts.

1. Whatever representation is adopted for the structure of the parsed sentences, it is agreed that complex sets of syntactic and/or semantic features must describe the linguistic units. Therefore, it is necessary to provide feature handling mechanisms. This point has suggested to us a way of providing a very rich language for handling feature structures (FS in the following). FSs are represented as trees where each arc is labelled by an attribute, and nodes can be pointers to the following alternative paths or a pointer to a leaf node where the value for the path spanned so far is found. They can store many kinds of information thanks to their efficient processing provided by a core set of functions.

2. Some linguistic phenomena encountered in parsing NL, such as long-distance dependency or the ability of treating some context-sensitive cases, led us to see the SAIL grammar rules as processes executed by a processor, a role covered by the parser. The rules of a grammar have associated some information related to their status of processes which are scheduled in a priority queue, according to some their priority of execution (/Knuth 1973/, /Aho et al. 1983/). This also allows, for instance, that the execution of some rule can be requested to perform context-sensitive recognition, or some rules can exchange between each other some information under the form of messages to perform the treatment of long-distance dependency.

3. The parser is structured as a bottom-up (shift reduce) all-paths algorithm, and a formalism for the grammar rules was defined to allow syntactic processing in parallel with semantic processing. The grammar of SAIL is a Phrase-Structure Grammar (PSG) with extensive augmentations, so that we also take advantage from the compositionality principle naturally

[1]This work has been carried out within the framework of the ESPRIT Project P527 CFID (Communication Failure in Dialogue: Techniques for Detection and Repair).

embedded in bottom-up parsers. As mentioned above, the parser is seen as a processor, thus one of its main tasks is to schedule the processes/rules to run in a priority queue. This queue is not completely under control of the parser since the grammar rules and the dictionary can also issue some specific operations or requests about the management of the scheduling task.

4. The need of a flexible front-end for the user is of primary importance to provide a powerful and complete development environment. The user interface built over SAIL, the SIS, is the framework where a user can interact with the underlying parsing system in developing grammars. This interface provides a set of commands, defined by means of a semantic grammar, that are caught and processed by SAIL and can handle many possible requests of the user.

In the following section we give a brief description of the grammar and dictionary format and how a grammar is defined in SAIL. Section 3 gives an overview of the SAIL parsing system, parser organization, and data structures it uses. Section 4 describes the parsing tools available in the system and their purposes. Finally, section 5 shows just one example of a grammar fragment where some parsing tools described in the previous sections are used.

2. The SAIL Grammar

The Grammar Format

The formalism we adopt to express grammar rules, called Complex Grammar Unit (CGU), defines a syntactic and a semantic side called syntactic rule and semantic rule, respectively. The syntactic rule contains the production, the tests, the actions and the recovery actions. The semantic rule contains the semantic counterpart of the syntactic tests and actions. The presence of the syntactic/semantic recovery actions is a very powerful mean to undertake alternative actions whether the rule fails either matching the right-hand side of the production or checking the syntactic/semantic tests. In this way the rules need not to be crudely rejected when they fail but, for instance, they can activate other rules that could be applied successfully.

A rule in SAIL is written defining all the previous CGU's items. In addition, it is also necessary to provide the status of the rule/process, so that it can be properly taken into account by the parser. The status says whether a rule can be scheduled for application or not by the parser. It can be **active** or **inactive**. Active rules always are scheduled by the parser, whereas inactive rules are not (inactive rules can be seen as sleeping rules). The status plays a central role in the organization of a grammar. As an example, if a rule detects some right or wrong conditions in the parsing structure it can either set active or activate an inactive rule.

Summarizing, a grammar rule is composed of three main items: 1) the status: **active** or **inactive**); 2) the production in context-free (CF) format, in the following denoted by $A \leftarrow w_1 \dots w_n$, $n \geq 1$, where the left-arrow means that the left-hand side is reduced from the right-hand side according to the bottom-up strategy of parsing; 3) the augmentations.

The production is augmented with an additional item, called the son-flag list. This list says for every category in the right-hand side whether the corresponding node matched in the parsing structure must be considered as a son of the left-hand side or not. If a son-flag is set to + for a right-hand side category the corresponding matched node is a son of the left-hand side node, otherwise it is not a son node if the flag is -. We have two types of production depending on its structure: CF and context-sensitive (CS) productions. CF productions, represented by $A \leftarrow w_1 \dots w_n$, are defined like:

$$\begin{array}{l} (A \quad (w_1 \ \dots \ w_n) \\ \qquad (+ \quad \dots \ +)) \end{array}$$

where all nodes matched by the right-hand side must be sons of the left-hand side node. CS productions represented by: $c_1 \dots c_p \, A \, c_{p+1} \dots c_q \leftarrow c_1 \dots c_p \, w_1 \dots w_n \, c_{p+1} \dots c_q$, $1 \leq p \leq q$, $n \geq 1$, are defined like:

$$(A \quad (c_1 \dots c_p w_1 \dots w_n c_{p+1} \dots c_q) \\ (- \dots - + \dots + - \dots -))$$

where only the nodes with a plus flag inside a context of minus-flagged nodes are sons of the left-hand side node.[2]

The augmentations cover the syntactic and semantic tests and actions of the CGU model. They are the body of a rule and are pieces of Lisp code executed by the parser during the application of the rule. Status, production and augmentations is the information provided by the grammar writer for every rule of a grammar. A rule is a named instance of a complex data structure defined according to the following **defrule** format:

```
(defrule
      :gname                    gname
      :rname                    rname
      :production               <production> [<son-flag-list>]
    [ status                    <status>
      :syn-tests                <code>
      :sem-tests                <code>
      :syn-actions              <code>
      :sem-actions              <code>
      :syn-recovery-actions     <code>
      :sem-recovery-actions     <code>] )
```

gname is the grammar name where the rule *rname* is defined. These two names must be provided in every rule definition since in the SIS we can have more than one grammar available which must be referred to by a name. A grammar usually is defined by a **defgramm** declaration of the form:

(**defgramm** *gname* [*root*])

where *root* is the root category of *gname*. This declaration sets up all data structures for the grammar being defined and must be issued before any rule definition.

The Dictionary Format

Any dictionary of a grammar contains a set of forms that are associated with a set of syntactic and semantic information. A **form** is whatever sequence of words $w_1\ w_2 \dots w_n$. When $n=1$ we have a **single form**, otherwise a **multiple form** ($n>1$). For any form, be it single or multiple, the first word w_1 is called the **key form**. The key form is the mean for storing and retrieving all information of the whole form in the data structures built by the **defgramm** declaration. Any form has associated three kinds of information, forming an **interpretation**: syntactic category; semantic value; a set of features. A form can have more than one interpretation. In this case, a set of interpretations must be defined supplying as the first item the key form; afterwards, for every sequence of words following the key form, the set of interpretations. An entry of the dictionary is defined according to the

[2]This definition leaves free the user of defining rules with discontinuous constituents in the syntactic representation. Currently the parser does not embed any strategy for a full treatment of these cases since the classical definition of adjacency is implemented. This structure was initially motivated in order to define CS rules by only one rule, and not by two (see Section 4.). Furthermore, such a structure allows a faster search in the parsing structure, performed by the matcher of the production, when, for instance, far constituents must be identified for long-distance tasks. Anyway, stated the important role that can be covered by the representation of discontinuous constituents (see /Bunt at al. 1987/), extension of the parser about this topic can be one of our future tasks.

following format:

```
                (defentry keyform gname
                   (defform form
                      (set-int :category   <category>
                               [:semval    <semval>
                               :features   <features>] )+ )+ )
```

where *keyform* must be a string of just one word, e.g., "dog", "train", etc.; the *form* must be either the null string "" for the single form *keyform*, or a string of one or more words. Every form definition of this kind is said to be in **defentry** format. <*category*> is the syntactic category and <*semval*> is the semantic value. The features must be provided in the following format:

```
<features>      ::= ( ((<attributes>) (<value>))+ )
<attributes>    ::= a sequence of feature attributes
<value>         ::= a value for the feature attributes
```

As an example:

```
( ((GENDER) (MASC))
  ((NUMBER) (SING))
  ((KIND-OF ARG1) (THING)) )
```

Here are some examples of dictionary entries. The most trivial of them is:

```
(defentry "train" my_grammar
   (defform ""
      (set-int   :category   Noun)))
```

where the single form **train** is defined by one interpretation of category Noun. An example of a single form with two interpretations is the following:

```
(defentry "tree" my_grammar
   (defform ""
      (set-int   :category   Noun
                 :features   ( ((KIND-OF OBJ) (PLANT)) ))
      (set-int   :category   Noun
                 :features   ( ((KIND-OF OBJ) (DATA-STRUCTURE)) ))))
```

where **tree** is defined as a plant and as a data structure. An example of multiple form is:

```
(defentry "in" my_grammar
   (defform ""
      (set-int   :category   Prep))
   (defform "the"
      (set-int   :category   CompPrep)))
```

where **in** is defined as a preposition and **in the** as a compound preposition.

The Feature Structures

In the current system we have adopted a data structure that can be at the same time efficient to be processed, homogeneous and reusable in various places of the system. This is why the same data structures are processed at different times in different places of the system. For instance, the lexical information looked-up from the dictionary is stored at parsing time in the terminal nodes of the parsing structure the parser builds. Thus, it is obvious to give the same format to the data in the dictionary and in the nodes of the parsing structure. Feature structures, in their classical definition as sets of attribute-value pairs, are associated with each interpretation of any form in the dictionary and of any node in the parsing structure. FSs are treated as trees, and it is possible to manage structures from the bottom of the parsing structure by means of a specific package of functions, called Feature Structure Handler (FSH), allowing the main operations on FSs as creation, modification, deletion. Currently, this package contains 12 main operations that can be applied on FSs. Over this set of low level operations on FSs we have developed a set of graph functions accessible by the user, which act on the FSs associated with the nodes of the parsing structure.

Rules with Non-Operative Productions (NOP Rules)

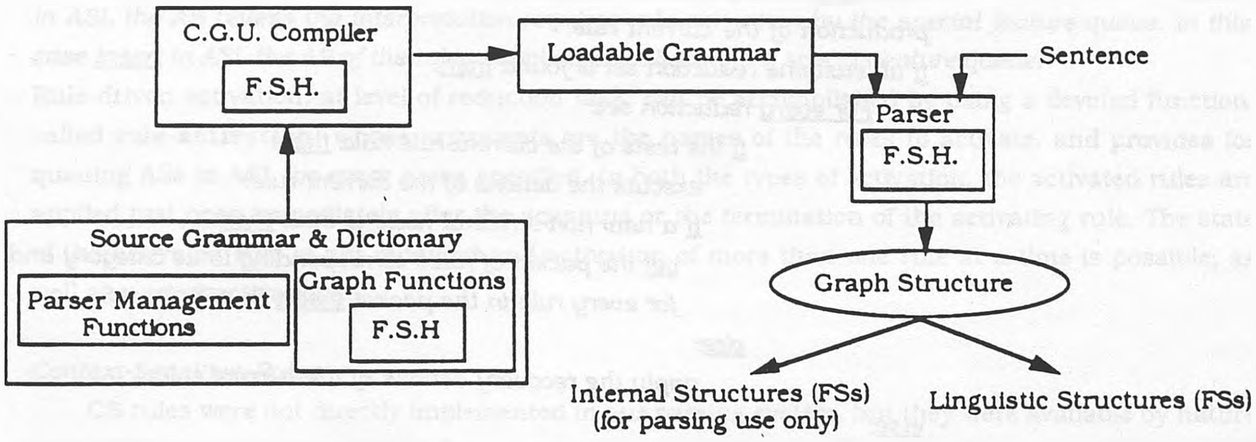
When non-operative productions are defined in some rule they do not build a new node, but can perform various actions, such as activating other rules, or altering semantic structures. There are three types of non-operative productions depending on the NOP category used in the left-hand side:

$$\{ \text{<NOP>} \mid \text{<NOP-ASE>} \mid \text{<NOP-SE>} \} \leftarrow w_1 \dots w_n$$

If <NOP> is used then only the syntactic rule is applied and the semantic rule is never considered. Only the semantic rule can be applied and the syntactic one is ignored by using the category <NOP-SE>. Finally, both the rules are applied by using the category <NOP-ASE>. As we shall see in Section 4. this kind of production can be useful in CS recognition, providing an alternative way for defining CS rules. Moreover, NOP rules are also useful when it is necessary to control the activation of *real* rules, with the objective of limiting the indeterminism of the parser.

3. Overview of the SAIL Parsing System

In this section we describe briefly the parser, the data structures it handles, and how it works. Starting from the FSH core package, we have adopted this data structure wherever possible inside the parsing system as the figure below shows. The parser builds a parsing structure under the form of a graph, where each node contains two kinds of information: an internal structure of data used by the parsing algorithm only, and the linguistic (syntactic and semantic) information set by the grammar rules. Both these structures are represented in a unique FS managed by the parser and the running grammar by using the underlying FSH functions. Any source grammar must have a set of rules and a set of dictionary forms written in the formats described previously. Grammar rules can make use of two sets of functions: the graph functions, which use the FSH package to update the linguistic structures of the graph, and the parser management functions to handle the various parsing tools and mechanisms (see Section 4.).


C.G.U. Compiler
F.S.H.
Loadable Grammar
Sentence
Parser
F.S.H.
Source Grammar & Dictionary
Graph Functions
Parser Management Functions
F.S.H
Graph Structure
Internal Structures (FSs) (for parsing use only)
Linguistic Structures (FSs)

The parser is a CF-based one, originally derived from the ICA (Immediate Constituent Analysis) algorithm described in /Grishman 1976/. It is a bottom-up shift-reduce action-based algorithm, performing left-to-right scanning and reduction in an immediate constituent analysis. The data structure it works on is a graph where all possible parse-trees are connected. The graph is composed of nodes that can be terminal or non-terminal. Terminal nodes are built in correspondence to a scanned form, whereas non-terminals are built whenever a rule (other than a NOP rule) is applied. The parsing system was designed to view the grammar rules as processes to be executed, and the parser as the processor. At any moment, the parser, following a priority schema, handles a queue of processes awaiting execution. In fact we can have different types of rules with different priorities of execution. So it is possible that a rule, when applied, sends a request for execution of another rule inserting the called rule in the appropriate position in the queue. After a scanning or a reduction, the parser gets a set of active rules which are the applicable rules at that moment. When the parser takes such a set - called a packet - for every rule in the packet[3] it builds a process descriptor and inserts it in the queue. We call such a process descriptor an application specification (AS), while the queue is called the application specification list (ASL). ASs are composed of all the necessary information useful to execute the process on the proper context. ASs in a given ASL are ordered depending upon the rule involved in an AS. In general, if standard active rules have to be executed, ASL is handled with a LIFO policy. The parser performs all possible reductions building more than one node if necessary, extracting one AS at a time before analyzing the next one. After an AS is extracted from ASL the parser searches a match for the right-hand side on the graph. The matching, if successful, returns one or more sets of nodes, called reduction sets. For every reduction set, the application of the rule is tried. In this way we can connect together all possible parses for a sentence in a unique structure. The complete algorithm of the parser is therefore:

```
Until the end of the sentence is reached:
    Scan a form:
        build a new terminal node for the scanned form;
        For every interpretation of the node:
            get the packet of rules corresponding to its category and for every rule in
            the packet insert in ASL the AS;
    For every AS in ASL:
        get the first AS from the top of ASL;
        get the rule specified in the AS, it is the current rule, and access the node
        specified in the AS, it is the current node;
        starting from the current node perform the match on the graph using the
        production of the current rule;
        if at least one reduction set is found then:
            For every reduction set:
                if the tests of the current rule hold then:
                    execute the actions of the current rule;
                    if a new non-terminal node is built then:
                        get the packet of rules corresponding to its category and
                        for every rule in the packet insert in ASL the AS;
                else:
                    apply the recovery actions of the current rule;
        else:
            apply the recovery actions of the current rule;
```

[3]A packet is a set of active rules. Any grammar is partitioned as a set of packets such that, for every category cat of the grammar, the packet P(cat) is the set of those rules that have cat as the right-most category in their right-hand side. This partitioning is useful for getting the rules applicable at a given moment and it is used by the matcher of the productions.

4. Parsing Tools

Rule Disabling/Enabling Operations

As stated previously, rules can assume two different states, active or inactive. The rule's state is determined at the moment of rule definition. In addition, it is possible to change the state during the parse by using two specific functions. In the application of a rule, others may be changed from active to inactive, performing a disabling operation, or changed from inactive to active, performing an enabling operation. It is possible to change the state of one or more rules at a time and the rules can also perform self-enabling and self-disabling operations. Changes of state effected during a parsing are not permanent. At the end of each parsing the rules are reconfigured as indicated in their original definition.

Dictionary-Driven and Rule-Driven Activation

The mechanism of activation of rules can be used in our parsing system in order to improve the determinism of the parser. We remark that the parsing algorithm is basically a bottom-up parallel **non-deterministic** parser, so that partitioning a grammar as a set of active and inactive rules, and driving their application by an activation mechanism, we can achieve a great control on the parser directly from the grammar, without embedding specific control strategies within the parsing algorithm.
Activation of rules can be effected during the two main phases of the parser activity: scanning and reduction. Dictionary-driven activation can be performed when the parser scans a form defined with an interpretation like the following:

```
(set-int
          :category    <category>
          :semval      <semval>
          :features    (((queue) (rule-name+))))
```

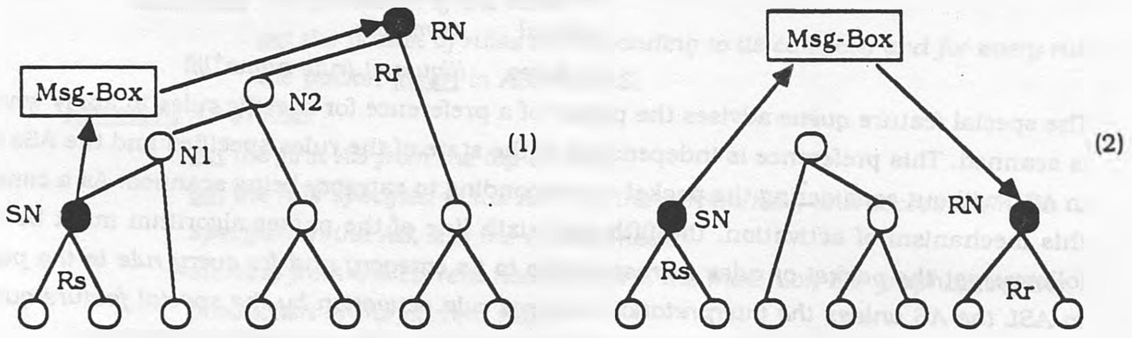
The special feature *queue* advises the parser of a preference for specific rules to apply when the form is scanned. This preference is independent of the state of the rules specified and the ASs are queued in ASL without considering the packet corresponding to category being scanned. As a consequence of this mechanism of activation, the fifth and sixth line of the parser algorithm must be changed as follows: *<u>get</u> the packet of rules corresponding to its category and for every rule in the packet <u>insert</u> in ASL the AS unless the interpretation requires rule activation by the special feature queue. In this case <u>insert</u> in ASL the AS of the rules supplied as values of the special feature queue.*
Rule-driven activation, at level of reduction task, can be accomplished by using a devoted function, called **rule-activation**, whose arguments are the names of the rules to activate, and provides for queuing ASs in ASL for every name specified. In both the types of activation, the activated rules are applied just once immediately after the scanning or the termination of the activating rule. The state of the activated rule is not modified and activation of more than one rule at a time is possible, as well as nested activations.

Context-Sensitive Rules

CS rules were not directly implemented in our parsing system, but they were available by nature (in addition to the way currently defined in Section 2.) thanks to the rule-activation mechanism and NOP rules. The complete application for a CS production $\alpha A\beta \leftarrow \alpha\gamma\beta$ is made in two steps. The first one concerns a context determination, the context being represented by the right-hand side of the CS production, $\alpha\gamma\beta$. The second one is just an application of the CF production $A \leftarrow \gamma$, if and only if the first step has determined the context where the CF production is applicable. This can be easily accomplished by defining a NOP rule for the context determination as first step. Afterwards, this NOP rule must activate the CF rule as second step, building the node A in the proper context.

Message Passing

The message passing mechanism is a parsing tool that makes possible asynchronous operations on linguistic data. This way of processing implies the co-operation between two rules which interact with each other exchanging some information by means of a sending and a receiving task performed at the two independent times of rule application. The sending task is performed by the sending rule at a time T_1, sending a message for another rule. This latter rule must perform the receiving task to receive the message at its execution time T_2, ($T_2>T_1$). Since the relevant linguistic data the parser works on are stored as FSs, the messages are FSs. We have implemented two approaches of message passing. The first one makes use of a global FS where any rule can store global features. Any rule during a parse can access this global FS and whatever feature value. This type of FS is the global counterpart of the FS stored in every node of the graph structure: the FS of a node is local and can only be accessed by the nodes linked to its node by a direct connection link. Therefore, there being no right of privacy on features in the global FS, this particular structure must be accessed with care by the rules since it can be a place of conflicts among them.
The second approach provides a structure that preserves the right of privacy of the messages. Also in this case the messages are FSs, and are stored in a sort of mailbox, called message-box. Any rule can refer to the message-box to store a message, specifying the destination rule. On the other side, any rule can refer to the message-box to get messages, and only the messages addressed to it will be available. Let us consider the two cases shown in the following partial parse-trees.

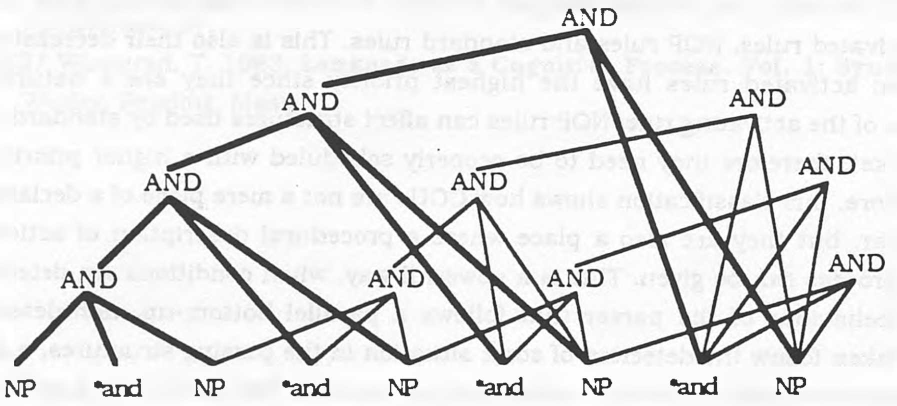


We suppose some information, created or raised in the node SN from the terminal side by the rule Rs, must be used in the node RN built by the rule Rr. (1) shows that the message-box could be used bypassing the nodes N1,N2. This is useful when (some) data from SN are not relevant for processing in N1 and N2, gaining the advantage that no memory space is wasted using the nodes N1,N2 for raising the data from SN to RN. On the other side, (2) shows a case where no path exists between SN and RN. Therefore, the only connection between the nodes can be a common structure accessed by them. The use of the message-box is very easy since all the work is done by two functions. The function **sendmsg** makes a copy of a subset of the FSs of the nodes it can access (i.e., the nodes corresponding to the left- and right-hand side of the production) and stores it in the message-box. The function **receivemsg** gets a message under the form of FS and stores it in the node corresponding to the left-hand side of the production.
All the functions: **sendmsg**, **receivemsg**, and those for handling the global FS are implemented using the FSH package.

5. An Example: SAILing X and X and ... X

The example shows a fragment of a grammar whose aim is to drive the parser according to a specific strategy of recognition achieving as result an optimized parsing structure, i.e., the minimum number of nodes strictly necessary is built.

The recognition of indefinitely long clauses of the form X **and** X **and** ... X could be achieved by using the productions: AND ← NP •and NP, AND ← AND •and NP, where, for instance, X can be an NP and •and is the category of **and**. These productions produce a parsing structure of the kind shown below. Being k the number of conjunctions, the number of the nodes N(k) built by these two productions is given by: N(k) = TN(k) + NTN(k), TN(k) = 2k + 1, NTN(k) = (1/2)k(k + 1).



TN(k) determines the number of the terminal nodes, and NTN(k) the number of the non-terminal nodes. For the graph above N(4) = 19, since TN(4) = 9 and NTN(4) = 10. This kind of parsing structure is not optimized, besides N(k) is a quadratic function of k. In the figure above we have drawn in boldface lines the parsing structure with the minimum number of nodes we want. For this optimized structure NTN(k) is a linear function of k: NTN(k) = k. Therefore, the formula for the optimized case $N_o(k)$ is: $N_o(k) = 3k + 1$.

Our grammar fragment is based on a watch-rule, called Check-and-rule, that checks whether the parser has already built a node of category AND followed by •and NP. This rule has the production: <NOP> ← AND •and NP, and if its right-hand side has no match it means that the first node AND has to be built. Check-and-rule has the following definition.

```
(defrule
          :gname                    my_grammar
          :rname                    Check-and-rule
          :production               (<NOP> (AND •and NP))
          :status                   active
          :syn-actions              (rule-activation '(Make-and-rule NP))
          :syn-recovery-actions     (rule-activation '(Make-first-and-rule NP)))
```

The syn-actions are applied if the right-hand side has a match and the rule Make-and-rule is activated to build a non-terminal node AND. The syn-recovery-actions are applied when the parser has to build for the first time a node AND, and the rule Make-first-and-rule is activated. These two activated rules must be inactive since the watch-rule has the work of activating them.

```
(defrule
          :gname          my_grammar
          :rname          Make-first-and-rule
          :production     (AND (NP •and NP))
          :status         inactive)
```

```
(defrule
        :gname        my_grammar
        :rname        Make-and-rule
        :production   (AND (AND *and NP))
        :status       inactive)
```

6. Final Remarks

Some remarks about the parsing system and the parsing tools described so far are in order. A first point concerns the priority assigned to rules. It is clear that we can have three main kinds of rules: activated rules, NOP rules and standard rules. This is also their decreasing priority order of execution: activated rules have the highest priority since they are a natural completion and extension of the activating rule; NOP rules can affect structures used by standard (non-NOP) rules in their packet, therefore they need to be properly scheduled with a higher priority than the others. Furthermore, this classification shows how CGUs are not a mere place of a declarative description of a grammar, but they are also a place where a procedural description of actions concerning the parsing process can be given. This is a powerful way, when conditions are detected, of altering the natural behaviour of the parser that follows a parallel bottom-up, non-deterministic strategy. Actions taken follow the detection of some situation in the parsing structures, e.g., the activation of a rule instead of another when a misspelling is found in the input, and a parsing process can be driven by a grammar where only the necessary rules for context detection are set active and those devoted to build structures inactive. This way of setting control of the parser places this parsing system in the category of situation-action parsers (/Winograd 1983/).
Unfortunately this paper cannot be a place for a wide description of examples of grammars using the parsing tools of SAIL. Some running examples, as well as that described above, can be found in /Marino 1988a/. Moreover, some ill-formed input cases have been faced, e.g., lexical/syntactic ill-formedness, constraint violation, constituent shuffling, missing constituents, in /Ferrari 1989/. A wide report of the work developed in the framework of the European ESPRIT Project P527 CFID using the SAIL Interfacing System is in /Deliverable 9/ where, among other things, the description of an English grammar and semantics is shown (/Mac Aogain et al. 1989/).

The author is thankful to Giacomo Ferrari who made possible this work.

References

/Aho et al. 1983/ Aho, A., V., Hopcroft, J., E. and Ullman, J., D. 1983. **Data Structures and Algorithms.** Addison-Wesley, Reading, Mass.

/Bunt at al. 1987/ Bunt, H., Thesingh, J. and van der Sloot, K. 1987. Discontinuous Constituents in Trees, Rules, and Parsing. **Proceedings of the 3rd Conference of the European Chapter of the ACL.** Copenhagen, Denmark, pp. 203-210.

/Deliverable 9/ **Deliverable 9: Implementation of Dialogue System.** 1989. Ref. CFID.D9.2. ESPRIT Project 527 (CFID).

/Ferrari 1989/ Ferrari, G. 1989. **The Treatment of Ill-Formed Input within the Frame of SAIL.** Working Paper. ESPRIT Project 527 (CFID).

/Grishman 1976/ Grishman, R. 1976. A Survey of Syntactic Analysis Procedures for Natural Language. **American Journal of Computational Linguistics.** Microfiche 47, pp. 2-96.

/Knuth 1973/ Knuth, D., E. 1973. **The Art of Computer Programming. Vol.III: Sorting and Searching.** Addison-Wesley, Reading, Mass.

/Mac Aogain et al. 1989/ Mac Aogain, E. and Harper, J. 1989. Semantics and Grammar. In

/Deliverable 9/.

/Marino 1988a/ Marino, M. 1988. The SAIL Interfacing System: A Framework for the Development of Natural Language Grammars and Applications. **Technical Report DL-NLP-88-1**. Department of Linguistics. University of Pisa.

/Marino 1988b/ Marino, M. 1988. A Process-Activation Based Parsing Algorithm for the Development of Natural Language Grammars. **Proceedings of 12th International Conference on Computational Linguistics**. Budapest. Hungary, pp. 390-395.

/Marino 1989/ Marino, M. 1989. SAIL: A Prototype Environment for Writing NL Applications. In /Deliverable 9/.

/Winograd 1983/ Winograd, T. 1983. **Language as a Cognitive Process. Vol. 1: Syntax.** Addison-Wesley, Reading, Mass.

An Efficient Method for Parsing Erroneous Input

Stuart Malone and Sue Felshin
Athena Language Learning Project
Massachusetts Institute of Technology

The Athena Language Learning Project receives major funding from the Annenberg/Corporation for Public Broadcasting Project.

Abstract

In a natural language processing system designed for language learners, it is necessary to accept both well-formed and ill-formed input. This paper describes a method of maintaining parsing efficiency for well-formed sentences while still accepting a wide range of ill-formed input.

1. Introduction

The Athena Language Learning Project is developing advanced educational software for foreign language learners. One of the tools we are developing is a natural language parser for use by first through fourth semester students of various languages. This parser must be able to recover from and correct a wide range of morphological, syntactic, and semantic errors, and yet still run in real time. We have designed a system where all of these errors can be handled by the parser uniformly and efficiently.

2. Description of the Parser

Our parser is a nondeterministic LALR(1) parser, written in Common Lisp, similar to that of Tomita [1] but differing in several significant ways.

- First, we associate a **reduction function** with each rule in the grammar. Whenever a reduce action is performed by the parser it calls the corresponding reduction function, which constructs the new node of the parse tree from the nodes on the right side of the production. As it does this, it may perform various tests on its input and either mark errors on the new node or, rarely, return NIL to fail. This is similar to the system of relaxation of predicates used by Weischedel and Black [2] and others; we mark errors where they allowed predicates to be relaxed, and return NIL from reductions where their predicates failed.

- Second, in order to properly handle linguistic phenomena like movement and binding, we needed to make the parser context sensitive. We did this by associating context sensitive information about a parse with each parse stack. This information is passed in to each reduction function, which examines and modifies the information as appropriate in order to build the new node.
- Third, we did not want the parser to return *every possible* parse of the student's input, given the relaxed rules of our grammar. A strict grammar for a natural language already has to consider many possible parses of the input—allowing erroneous input increases the problem by an order of magnitude or more. Computing all these parses would be a waste of time, and would make the system unusably slow. Instead, we only want the parser to return the "most likely" parses.
- Fourth, we decided that it was essential for our parser to perform semantic analysis at the same time as syntactic analysis in order to reduce ambiguity. Even though syntactic errors are common for language learners, semantic errors are more unusual. If parsing can be guided by semantic constraints as well as syntactic ones, then we can expect to come up with the better interpretations of the student's input with less work.

Adding context sensitive information to each parse stack had the significant disadvantage that it became impractical to use some of Tomita's more sophisticated techniques such as graph-structured stacks and local ambiguity packing.[1] However, abandoning these techniques allowed us to take advantage of a different one: a best-first searching strategy. Creating graph-structured stacks requires the use of breadth-first search in order to keep all of the parse stacks synchronized on the input. Without graph-structured stacks, it becomes possible to advance different parses of a sentence at different rates, forging ahead with parses that look promising, and postponing work on less likely ones.

3. Marking Errors

In our parser, every word and every node in the parse tree contains an error-count which is initially zero. Whenever an error is detected, our code increases the error-count of the word or node and attempts to generate a plausible corrected node.

Four kinds of errors are detected by the lexical analysis pass of the parser, and are marked on individual words before they are parsed.

errors in the lexicon	Some errors are so common that we have anticipated them by entering them directly into the lexicon. For instance, use of the wrong gender ending on a noun in Spanish, e.g., "abriga" for "abrigo" (*"overcoat"*). Lexical lookup returns a word marked with an error.
spelling errors	When regular lexical lookup fails, we run a spelling checker to search for known words with similar spellings. Each misspelled word is marked with an error.

[1]This was because, in order for two parse stacks to be merged, the context sensitive information associated with each stack had to be compatible. This situation was so rare that the bookkeeping involved wasted more time than was saved.

blocked word errors	Irregular forms of words are stored in the lexicon as subentries of their regular forms. After lexical lookup, a second pass checks the irregular subentries of the returned word to make sure none of them should have been used instead. If it determines that one should have, it marks the word with an error before returning it. For example, the Spanish word "tenió" in place of "tuvo", or the English word "haved" in place of "had".
surface filter errors	The surface filter looks at the stream of words returned by lexical lookup and performs arbitrary surface operations, such as splitting "compound" words into their components, combining single meanings given by more than one word, and insuring that words are properly contracted. As it does this, it marks any errors it finds on the appropriate words. In English, for instance, a surface filter checks for correct "a/an" alternation as in "a dog" vs. "an apple".

It is important to understand that the lexical analysis pass may return several different interpretations for a single word, some of which may have errors while others may not. For instance, in English the word "seed" could either be the correct singular form of the noun "seed" or the incorrect (blocked) past tense of the verb "to see".

Three other kinds of errors are detected during parsing and are marked on nodes by the reduction functions that create those nodes.

structural errors	The grammar productions anticipate certain structural errors, similar to the way that the lexicon anticipates certain lexical ones. For instance, Spanish detects improper use of preposition-like words, e.g., "encima la mesa" instead of "encima de la mesa" (*"on (top of) the table"*).
agreement errors	The reduction functions mark errors as appropriate for any context dependent and/or independent syntactic requirements which are violated by the current constituents. In English, the noun phrase "a books" would be marked with an agreement error and assumed to be plural.
semantic errors	The reduction functions also access the case frame interpreter, which builds semantic structure and marks any necessary errors. Even though the semantic structures are separate from the syntactic nodes of the parse tree, semantic errors are marked on the nodes of the parse tree so that they will be visible to the parser.[2]

4. How Parsing Proceeds

Now that we have explained how we mark errors on the words and nodes of a parse tree, we can explain how these errors are used by the parser to direct parsing. At this point it is helpful to introduce a term for the information that is stored about a partially-completed parse. We call this infomation the parse state, or **pstate** for short. A pstate contains the following information:

[2]The semantic structures built by the case frame interpreter introduce a new level of ambiguity—each represents any number of possible semantic interpretations of the constituent. The error-count of a case frame is the error-count of its best interpretation.

- The traditional LALR parse stack of alternating nodes and state numbers.
- The current word of the input, which is the cu nt look-ahead token for the LALR parser and will be the next word shifted onto the parse stack.
- An error-count, which is used to determine which pstate is "best" in the best-first search for a successful parse.
- Context-sensitive infomation which varies from language to language.

The most interesting part of a pstate for this discussion is the error-count, which is used to direct the best-first parsing. The error-count of a pstate is the sum of the error-counts of the nodes in its parse stack, plus the error-count of its current word.

The parser keeps a sorted list of pstates. Pstates with the same error-count are ordered arbitrarily. Each step through the parser pops the first (best) pstate off of this list and looks up the next actions for the pstate in the LALR tables. For each action, the parser does the following:

- If the action is a shift action, it shifts the current word onto the parse stack. A new pstate is created for each possible following word, and the following words are made the current words of the new pstates. Each new current word's error-count is added to the error-count for its pstate.
- If the action is a reduce action, then the arguments to the reduction, which are the right side constituents, are popped off the parse stack and passed to the reduction. If the reduction constructs a result node, a new pstate is created, the node is pushed onto its parse stack, and the node's error count is added to the pstate's error-count.

Each pstate created during the above procedure is inserted in its proper position in the list of pstates, and the procedure is repeated with the new best pstate. This continues until either the list of pstates becomes empty, in which case parsing has failed, or enough pstates parse to completion that the remaining (worse) pstates are simply thrown away. To determine when to throw away pstates, we maintain a range which we call the **style threshold**. Whenever the error-count of a pstate becomes larger than the error-count of the best successful parse plus the style threshold, that pstate is removed from the list of pstates and thrown away. However, no pstates are thrown away until there is a successful parse.

4.1. An Example

As an example of how this system works, we'll describe the parsing of the sentence "Dije donde llovió," which is incorrect Spanish for *"(I) said where (it) rained."* "Donde" with no accent is a subordinating conjunction, as in "I'll go *where* you go." "Dónde", with an accent, is a pro-PP introducing a complement clause, as in "I said *where* it rained." Unsurprisingly, students of Spanish use the wrong form quite often. Lexical analysis of "donde" in our system returns two words, the subordinating conjunction and the pro-PP, the latter marked with an error-count of 500 for the lack of an accent.

The parser starts with a single pstate, call it A, where the current word is "dije", the first word in the input sentence. The grammar first pushes various empty nodes onto the parse stack, including an empty COMP and a pro subject, and eventually shifts "dije". Since the next word, "donde", is ambiguous, the parser must now split this pstate into two new pstates, B and C. Pstate B receives the subordinating conjunction as its current word, and has its error-count increased by 0. Pstate C receives the pro-PP, and has its error-count increased by 500. Processing of pstate C is then postponed because it's not the best available pstate.

Parsing continues with pstate B. The subordinate clause "donde llovió" is completed and attached to the S node dominating "dije". Now the sentence is ready to be finished off. But finishing it off precludes the possibility of more arguments being parsed, and the verb "dije", which requires a direct object or complement clause, has received neither. The case frame interpreter marks an error of 600 for a missing argument, and when this is added to pstate B, it is no longer the best pstate. Thus pstate B is now postponed in favor of pstate C.

Parsing of pstate C now resumes at the point it was left off, and by parsing "donde llovió" as a complement clause, continues to a successful completion. If the style threshold is less than 100, parsing will stop, and pstate C, with an error-count of 500, will be returned. Otherwise, parsing of pstate B will resume until it is successfully completed with an error-count of 600, and both pstates B and C will be returned.

5. Anticipated Errors

Because the error-count of a pstate determines whether or not the pstate should be actively pursued, postponed, or thrown away, it is vitally important for error-counts to be accumulated as soon as possible.

Take, for example, the sentence "You drink too much beer." This sentence has two interpretations: the obvious one, and an erroneous one where a case-blocking modifier has been placed between the verb and the direct object (this reading should be "You drink beer too much"). The error in the second interpretation is in the *placement* of the ADVP "too much" within the VP—there is nothing wrong with the ADVP itself. Conceptually, therefore, the error should be marked on the VP. But before this VP can be built, the erroneous modifier and the direct object must be parsed; this will waste a lot of work before this reading's pstate is postponed in favor of the first reading, and eventually discarded unfinished due to the style threshold. Alternatively, we can build a modifier node around the ADVP node and mark the error there, saving the time it takes to parse the direct object. Or best of all, before starting to build the ADVP in the first place, we can build an empty node and mark the error on it. Thus this pstate will be postponed as soon as the empty error node is created—before either the ADVP *or* the direct object have been parsed.

We call these errors, which are marked on empty nodes *before* the erroneous input, **anticipated errors**. In many ways, anticipated errors are the most important category of error, because they have the greatest influence on the speed of parsing. Anticipated errors allow us to postpone or discard a pstate *before* we have wasted a great deal of time on it.

We handle many structural errors by writing explicit productions to parse ill-formed input, and these errors can always be anticipated. For instance, in our system we can write a rule such as:

```
    VP => VBAR MOD,BAD-MOD? OBJ
```

This rule says: "To parse a VP, parse a VBAR, optionally followed by a MOD marked with the BAD-MOD error, followed by an OBJ." This is automatically expanded by our LALR table generator into:

```
VP => VBAR MOD,BAD-MOD? OBJ

MOD,BAD-MOD? =>
MOD,BAD-MOD? => MOD,BAD-MOD

MOD,BAD-MOD => BAD-MOD MOD

BAD-MOD =>                          (create an empty error node)
```

These rules will mark the partially completed parse with an error as soon as the parser decides that there is a MOD after the VBAR, *before* the MOD, OBJ, or VP has been created. Since parsing is best first, this partial parse will be postponed until it is the best parse available—which may never happen. However, if the parse *does* become the best available, the work done to construct the VBAR will not have been wasted. Processing of this partial parse will continue right where it left off.

Some structural errors are too complex to anticipate through the use of the LALR table generator's error facililty. For example, in Spanish, infinitive sbar complement clauses must be introduced with one of two different complementizers, "a" or "de"; or with no complementizer at all, depending on the higher verb. English speakers, who are accustomed to always using the particle "to", frequently choose the wrong complementizer in Spanish. Since all three structures (either complementizer or none at all) are potentially correct in Spanish, there is no place in any production to mark the error.

We can still anticipate the error, however, by having the reduction function called when the complementizer is reduced look at the higher verb. We can find the higher verb using the context-sensitive information that is kept with each pstate. We call the case frame interpreter to determine whether the higher verb allows an SBAR complement clause, and if so, whether the verb allows the given complementizer. We push a small error if the wrong complementizer was used and a very large error if no clause is allowed at all.

6. Conclusions

We have written grammars for Spanish, English, French, German, Russian, and Classical Greek. The most comprehensive of these is for Spanish, where the grammar contains over five hundred context-free productions.

The following data demonstrates the time saved by using best-first parsing and by trimming unlikely pstates through use of the style threshold. Time is measured by the total number of reductions performed during parsing the input, counting both successful and trimmed pstates. We generally set the style threshold at 30 to allow for slight variations in interpretations of the input. Setting the style threshold to a very large number (such as 10,000) approximates what would happen if an equivalent grammar were run using Tomita's parser.

Dije donde llovió.

"(I) said where (it) rained."

Style threshold	# of parses	# of reductions
0	1	680
30	1	680
100	2	1,010
10,000	19	3,549

Hace buen tiempo en Bogotá.

"(It) makes good weather in Bogota."

Style threshold	# of parses	# of reductions
30	1	831
10,000	22	7499

Acknowledgements

We wish to thank Professor Robert C. Berwick and Dr. Janet H. Murray of MIT for supporting our research and convincing us to write this paper.

References

[1] Masaru Tomita.
Efficient Parsing for Natural Language.
Kluwer Academic Publishers, Boston, 1986.

[2] Ralph M. Weischedel and John E. Black.
Responding Intelligently to Unparsable Inputs.
American Journal of Computation Linguistics 6(2):97–109, April–June, 1980.

Analysis Techniques for Korean Sentences based on Lexical Functional Grammar

Deok Ho Yoon, Yung Taek Kim
Department of Computer Engineering
Seoul National University
Seoul, Korea

ABSTRACT

The Unification-based Grammars seem to be adequate for the analysis of agglutinative languages such as Korean, etc. In this paper, the merits of Lexical Functional Grammar is analyzed and the structure of Korean Syntactic Analyzer is described. Verbal complex category is used for the analysis of several linguistic phenomena and a new attribute of UNKNOWN is defined for the analysis of grammatical relations.

1. Introduction

In these days, various kinds of Unification-based Grammars are developed and widely researched[1,2]. Lexical Functional Grammar(LFG)[3,4] is one of them and seems to meet well for the grammatical characteristics of Korean.

We have developed a Korean natural language parser, KOSA(KOrean Syntactic Analyzer) which is based on the LFG. It is the analysis part of the KEMTS(Korean-English Machine Translation System) which is our current machine translation system.

In this chapter the grammatical characteristics of Korean and the merits of LFG formalism are presented.

1-1. The Grammatical Characteristics of Korean

Korean which is classified into the Ural-Altaic languages and belongs to the agglutinative languages is greatly different in the linguistic structures from the Indo-European languages such as English.

Korean adopts a short-clause as the unit of the spacing words. One short-clause is constructed by the concatenation of one or more morphemes of individual lexical categories. The concatenation is restricted by word conjoin conditions.

The most common patterns of short-clauses are 'verb(suffix)$^{+}$' and 'noun(postnoun)$^{+}$'. In such patterns, morphemes belonging to verb or noun bring the major informations. But because Korean is an agglutinative language, such morphemes have no conjugation and cannot have auxiliary informations freely. In Korean, such auxiliary informations are expressed by suffixes or postnouns which follow verb or noun, and their informations have an important role on the analysis of Korean[10].

Suffixes represent grammatical informations such as modality, tense, mood, voice, and etc. In Korean, agreement rules about gender, number or person are not developed well, but various idiomatic expressions of complex patterns are widely used.

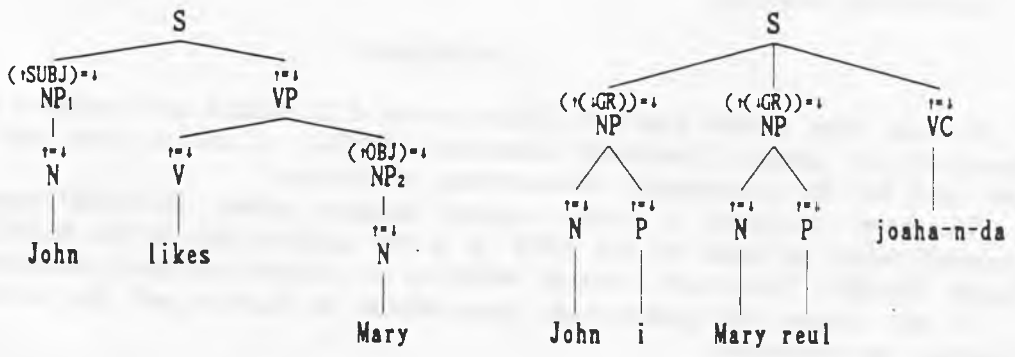
The major function of the postnoun is to show the grammatical relation(GR) between an NP and a verb. Unlike the Indo-European languages in which the GR information is directly obtained from the structure of the sentence, in Korean postnoun tells the GR. So there is no need to distinguish NP and PP, and the order of NPs does not affect on the meaning. This brings on the relatively free word order of Korean.

When postnoun with other kind of information is used, the postnoun with the GR information is omitted frequently. To analyze such cases, inferences using various knowledges and heuristics are required.

1-2. The Merits of LFG for Korean Analysis

LFG has several merits for the analysis of Korean sentences. Some of them comes from the fact that Korean is not a well structured language.

The first merit is the fact that the primitives of LFG are the grammatical relations (GRs) such as SUBJ, OBJ, etc., but not the phrases such as NP, VP, etc. In English, the GRs of NPs can be detected from the order in the phrase tree. For example, we can see that NP_1 is the SUBJ of S and NP_2 is the OBJ of S from the c-structure for English in Fig.1-a, but this is not permitted for Korean as shown in Fig.1-b, because of the free word order of NPs. LFG offers a convinient way to analyze the implicit GRs, and more extended analysis methods will be proposed in chapter 4.



(a) Fig-1. GR of NPs in two C-structures (b)

The second merit is the fact that postnouns and suffixes in Korean can be easily and efficiently analyzed with lexical rules.

Also LFG provides convinience of invoking the inference mechanisms with grammatical devices and constraint conditions for various purposes such as the determination of UNKNOWN attributes.

In the design of KOSA, we tried to maximize such merits of LFG. Following chapters will describe the structure of KOSA and the techniques that we adopt.

2. The Structure of KOSA

Korean Syntactic Analyzer, KOSA is a Korean parser based on LFG. It analyzes a Korean sentence and extracts the grammatical informations in the form of an f-structure. The output of KOSA can be used in various applications. KOSA has developed as the analysis module of a Korean-English Machine Translation System, KEMTS and the output of KOSA is used as the intermediate structures for translation.

KOSA consists of three modules: LexAnal, CstrAnal and FstrAnal. Fig-2 shows the block diagram of KOSA. Each section describes the structure of each module.

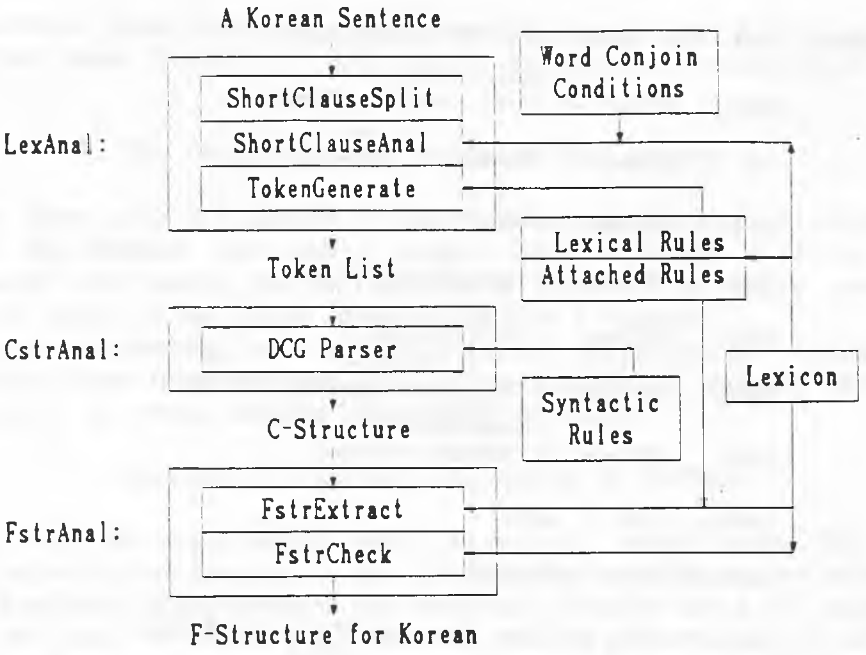


Fig-2. Block Diagram of KOSA

2-1. The Structure of LexAnal Module

LexAnal module analyzes a Korean sentence into the token strings and consists of three phases: ShortClauseSplit, ShortClauseAnal and TokenGenerate.

The ShortClauseSplit phase splits a Korean sentence into a number of short-clauses using blanks and punctuation symbols as the delimeters. This phase can be constructed easily as a simple finite state automata.

Each short-clause is analyzed into morphemes in the ShortClauseAnal phase. As shown in section 1-1, the concatenations of morphemes are restricted by the word conjoin conditions which check the lexical categories, the phonology and the semantics. Although the word conjoin conditions seem to be complicated, they are just simply some local rules which deal only adjacent morpheme pairs. So this phase can be implemented as an automata, too.

TokenGenerate phase generates the token strings from the morphemes. In this phase, some morpheme patterns are combined into one complex token. Among some kinds of complex tokens, verbal complex(VC) tokens are the most important. Typically a verb and its following suffixes are combined into one VC token. But there also exist more complex VC token types, and they are discusses in chapter 3. By generating complex tokens, many local linguistic phenomena can be excluded from the CstrAnal/FstrAnal modules. Because these modules analyze the global relationship among the sentence constituents, the approach of combining morphems can greatly enhance the efficiency. This phase is implemented as the recursive pattern rewriting rules.

2-2. The Structure of CstrAnal Module

The syntactic rules of the CstrAnal module are shown in Fig-3, and these rules are enough to analyze most Korean sentences. Complex tokens are dealt like the simple tokens according to their lexical categories. Each syntactic rule has functional schemata showing the method of unification. By adding these functional schemata to each branch of the phrase trees, the c-structures are constructed.

(S1) $\mathrm{S[Type]} \rightarrow (\overset{(\uparrow(\downarrow GR))=\downarrow}{\mathrm{NP}} \quad \overset{\downarrow\in(\uparrow ADJ)}{\mathrm{AVP}})^{*} \quad \overset{\uparrow=\downarrow}{\mathrm{V[Type]}}$

(S2) $\mathrm{S[Type]} \rightarrow \overset{\downarrow\in\uparrow}{\mathrm{S[connective]}} \quad \overset{\downarrow\in\uparrow}{\mathrm{S[Type]}}$

(NP1) $\mathrm{NP[Type]} \rightarrow \overset{\uparrow=\downarrow}{\mathrm{N}} \quad \overset{\uparrow=\downarrow}{\mathrm{P[Type]}}$

(NP2) $\mathrm{NP[Type]} \rightarrow \overset{\uparrow=\downarrow}{\mathrm{S[nominative]}} \quad \overset{\uparrow=\downarrow}{\mathrm{P[Type]}}$

(NP3) $\mathrm{NP[Type]} \rightarrow \overset{\downarrow\in(\uparrow ADJ)}{\mathrm{ADJ}} \quad \overset{\uparrow=\downarrow}{\mathrm{NP[Type]}}$

(NP4) $\mathrm{NP[Type]} \rightarrow \overset{(\uparrow(\downarrow GR))=\downarrow}{\mathrm{NP[possesive/conjunctive]}} \quad \overset{\uparrow=\downarrow}{\mathrm{NP[Type]}}$

(NP5) $\mathrm{NP[Type]} \rightarrow \overset{\overset{\downarrow\in(\uparrow XADJ)}{(\downarrow UNKNOWN)=\uparrow}}{\mathrm{S[modify]}} \quad \overset{\uparrow=\downarrow}{\mathrm{NP[Type]}}$

(AVP1) $\mathrm{AVP} \rightarrow \overset{\uparrow=\downarrow}{\mathrm{ADV}}$

(AVP2) $\mathrm{AVP} \rightarrow \overset{\uparrow=\downarrow}{\mathrm{S[adverb]}}$

Fig-3. The Syntactic Rules of KOSA

(S1) shows the structure of a simple sentence and (S2) shows the coordinative sentences. (NP1) and (NP2) show the basic structures of NPs and (NP3)-(NP5) show the constituents which can modify the NPs. With above rules, postnouns are combined with nouns(or nominal clauses) at the lowest level of the c-structure, but this has no problem because the postnouns supply only the auxiliary informations.

The unhierarchical syntactic rule (S1) makes the forms of c-structures flat and brings on much ambiguity especially on the position of NPs. So above rules examine context-sensitive constraints to decrease the ambiguity. The applications of rules are restricted by the context-sensitive informations in the bracket. But this approach is not enough to prohibit the ambiguity of NP's position. To resolve such ambiguity, the possibility for the unification of f-structures should be examined.

This module is implemented with the DCG(Definite Clause Grammar) parser[5] on PROLOG.

2-3. The Structure of FstrAnal Module

The FstrAnal module consists of two phases: FstrExtract and FstrCheck.

Because CstrAnal module results much ambiguity, FstrAnal module should cover the task of filtering out illegal c-structures as well as the task of analyzing the f-structures. Two phases of this module, will function as a two-level filter and generate the result f-structures from correct c-structures only.

FstrExtract phase extracts the f-structures of the input sentence from the c-structures by the bottom-up unification algorithm[3,6]. The complexity of the unification algorithm in KOSA is not heavy, and is the level of general unification algorithm for LFG formalism. Even though the grammatical characteristics of Korean are not reflected well by the unification algorithm, they are reflected through the lexicon informations and the functional schemata shown in section 2. Attached rules are used to extract the functional schemata for the verbal complex tokens in this phase. Chapter 3 will describe the functions of the attached rules.

FstrCheck phase examines the extracted f-structures whether they are grammatical or not. Grammatical devices and constraint conditions of LFG are utilized for KOSA, but some constraint conditions are modified and extended in order to solve Korean linguistic phenomena. Some heuristics to the determine the unknown GR values of NPs are used in this phase. Section 4-2 will describe the modifications/extensions and the heuristics.

3. The Introduction and Usage of VC category

In English, there is the VP category which consists of all sentence constituents except the subject of the sentence. But such a category can't be found in Korean because of the free word order among the NP constituents including the subject constituents. So Korean verb seems to be directly governed by the S category.

Verbs are ususally combined with suffixes or another morphemes into complex tokens in TokenGenerate phase. In this chapter, various usages of the VC category which means the lexical category of verbal complex tokens will be shown.

3-1. Analysis of Auxiliary Informations in Suffixes

In Koean, there are many suffixes with complex and various usages. But most of them does not affect on the meaning of the verb supplying only the auxiliry informations. So when the FstrExtract phase extracts the functional schemata for a VC token which consists of a verb and its following suffixes, the auxiliary informations of suffixes are appended to the functional schemata of the verb.

For example, Korean word *'meok-eot-da'* means 'ate'. *'meok'* is a verb which means 'eat', *'eot'* is a past-tense suffix, and *'da'* is a ending suffix for descriptive sentences. The FstrExtract phase appends these informations from lexicon like below.

```
vc([v(meok),f(eot,tense),f(da,final)])  :
     (↑PRED)  =  'EAT<(↑SUBJ)(↑OBJ)>'
     (↑TENSE)  =  PAST
     (↑MODE)  =  DESC
```

3-2. Analysis of Idiomatic Expressions

Koean has many idiomatic expressions on the predicate part. If idiomatic expressions are analyzed in CstrAnal/FstrAnal modules, the c-structures and the functional schemata can become much more complicated. So KOSA combines each idiomatic expression into one VC token in TokenGenerate phase, and obtains their functional schemata from the attached rules in FstrExtract phase. This approach greatly diminishes the overhead of CstrAnal and FstrAnal modules.

For example, a Korean idiomatic predicate *'meok-eul soo eop-da'* consists of three short-clauses and five morphemes. It means 'cannot eat', and can be thought as 'eat' with auxiliary information of negative possibility. So KOSA, combines this expression into one VC token and the attached rule adds the functional schemata, (↑POSSIVILITY)= '-' to those from lexicon. Below is the result token and functional schemata.

```
vc([v(meok),f(eul,modify),n(soo),v(eop),f(da,final)])  :
     (↑PRED)  =  'EAT<(↑SUBJ)(↑OBJ)>'
     (↑MODE)  =  DESC
     (↑POSSIBILITY)  =  '-'
```

3-3. Analysis of Duplicated Constituents Expressions

Some Korean sentences have duplicated subjects or duplicated objects. This phenomenon is called as duplicated constituents problem, and KOSA analyzes the typical case of this problem using VC category.

For example, in Korean *'Cheolsoo-ga ki-ga keu-da'* means 'Cheolsoo is tall'. Because postnoun *'ga'* is a subject marker, there exist two subjects *'Cheolsoo-ga'* and *'ki-ga'*. As *'ki'* means 'height' and *'keu'* means 'big', *'kei-ga keu'* means 'be tall'. In Korean, the verb, *'ki-keu'* which means 'be tall' is also used. Like this, many Korean adjective verbs are often expressed in the form of a subject and following simple adjective verb. So KOSA combines *'ki-ga keu-da'* into one VC token, and the attached rule interprets it just like *'ki-keu-da'*. Similar method is applied to verbs which require duplicated objects.

3-4. Analysis of Passive/Causative Expressions

In Korean, passive/causative expressions are all represented using suffixes. For example, *'meok-hi-da'* means 'be eaten', and *'hi'* is a suffix showing passiveness. Similarly *'meok-i-da'* means 'let ... eat', and *'i'* is a suffix showing causativeness.

KOSA combines such an expression into one VC token, and obtains the functional schemata for this token using the methods proposed by Kaplan[7,8].

For *'meok-hi-da'* and *'meok-i-da'*, the attached rule for passiveness/causativeness transforms the functional schemata of *'meok-da'* like below.

```
vc([v(meok),f(da,final)]) :                  => vc([v(meok),f(hi,pass),f(da,final)]) :
    (↑PRED) = 'EAT<(↑SUBJ)(↑OBJ)>'                (↑PRED) = 'EAT<(↑OBL_AGT)(↑SUBJ)>'
    (↑MODE) = DESC                                (↑MODE) = DESC

vc([v(meok),f(da,final)]) :                  => vc([v(meok),f(i,cause),f(da,final)]) :
    (↑PRED) = 'EAT<(↑SUBJ)(↑OBJ)>'                (↑PRED) = 'LET<(↑SUBL)(↑OBJ2)(↑XCOMP)>(↑OBJ)'
    (↑MODE) = DESC                                (↑XCOMP PRED) = 'EAT<(↑SUBJ)(↑OBJ)>'
                                                  (↑XCOMP SUBJ) = (↑OBJ2)
                                                  (↑XCOMP OBJ) = (↑OBJ)
                                                  (↑MODE) = DESC
```

4. Determination Techniques of Grammatical Relations

The GR of Korean NPs are mainly determined by the postnouns. The GR value of P is transmitted by '↑=↓' to the NP, and indirectly used by '(↑(↓GR))=↓'[9].

But sometimes the GRs of NPs cannot be determined by the postnouns for two reasons. One reason is the omission of the postnoun showing the GR value. Another reason comes from the relation between the relative clauses and the antecedents.(Relative clause precede its antecedent, in Korean.) Here the antecedent has a role as an NP in the relative clause. But the postnoun of the antecedent shows only the GR for main clause, and the GR for relative clause is unknown.

Even in such cases, we should find the hidden GRs for correct analysis. This chapter describes the determination techniques of such unknown GRs.

4-1. Introduction of UNKNOWN Attributes

Because the heuristics to determine the unknown GR value should refer to the global relations among the VC and another NPs, the f-structure of the sentence should be able to be extracted before the heuristics are invoked. So we have introduced the UNKNOWN attribute to represent the temporary GR values. It is inserted and used during the FstrExtract phase, and changed to the correct GR value by the heuristics in FstrCheck phase.

The UNKNOWN is inserted by two methods. When the postnoun showing the GR value is omitted, the 'null' postnoun whose lexicon information has the functional schemata, '(↑GR)=UNKNOWN' is inserted in TokenGenerate phase. By the functional schemata, UNKNOWN becomes the attribute representing the NP whose GR is unknown. For the relative clause, syntactic rule (NP5) in section 2-2 inserts the UNKNOWN attribute whose value is the f-structure for the antecedent to the relative clause.

5-1. Analysis Result of LexAnal Module

- after ShortClauseSplit phase: five short-clauses are generated

  [*'woori-ga'*, *'ta-n'*, *'bihaenggi-neun'*, *'Seoul-e'*, *'dochakha-et-da'*]

- after ShortClauseAnal phase: eleven morphemes are generated

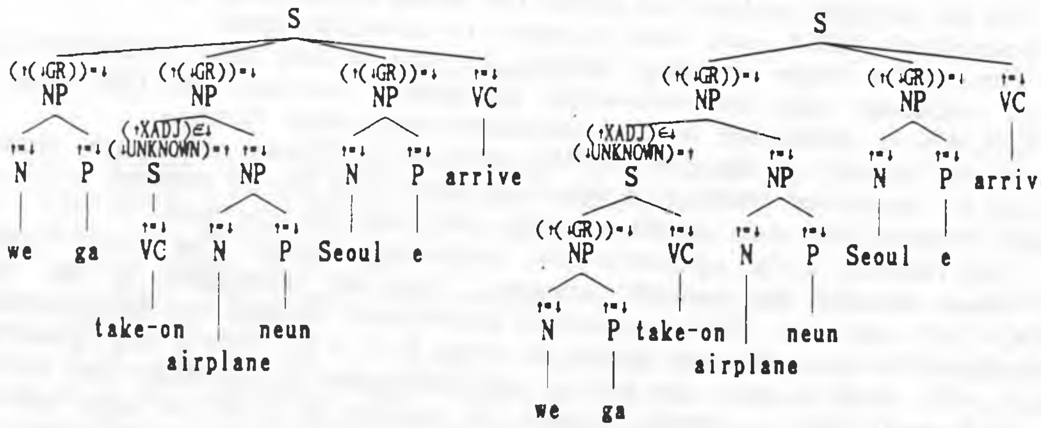
  [noun(we), post(*ga*,subj-mark), verb(take-on), suffix(*n*,modify),
  noun(airplane), post(*neun*,topic), noun(Seoul), post(*e*,obl_{loc}-mark),
  verb(arrive), suffix(*et*,tense), suffix(*da*,final)]

- after TokenGenerate phase: eight tokens are generated

  [noun(we), post(*ga*,subj-mark), vc([verb(take-on), suffix(*n*,modify)]),
  noun(airplane), post(*neun*,topic), noun(Seoul), post(*e*,obl_{loc}-mark),
  vc([verb(arrive),suffix(*et*,tense),suffix(*da*,final)])]

5-2. Analysis Result of CstrAnal Module

- after CstrAnal module: two alternative c-structures are generated as below



5-3. Analysis Result of FstrAnal Module

- functional schemata of morphemes obtained from lexicon

noun(we):	(↑PRED) = 'PRO'	noun(airplane):	(↑PRED) = 'AIRPLANE'
	(↑NUM) = PLURAL	noun(Seoul):	(↑PRED) = 'SEOUL'
	(↑PERS) = 3		
verb(take-on):	(↑PRED) = 'TAKE-ON<(↑SUBJ)(↑OBJ)>'		
verb(arrive):	(↑PRED) = 'ARRIVE<(↑SUBJ)(↑OBL_{LOC})>'		
post(*ga*):	(↑GR) = SUBJ	suffix(*n*):	(↑MODE) = MODIFY
post(*neun*):	(↑TOPIC) = '+'	suffix(*et*):	(↑TENSE) = PAST
post(*e*):	(↑GR) = OBL_{LOC}	suffix(*da*):	(↑MODE) = DESC

- functional schemata of complex tokens obtained by lexical rules

vc([verb(take-on), suffix(*n*,modify)]):
(↑PRED) = 'TAKE-ON<(↑SUBJ)(↑OBJ)>'
(↑MODE) = MODIFY

vc([verb(arrive),suffix(*et*,tense),suffix(*da*,final)]):
(↑PRED) = 'ARRIVE<(↑SUBJ)(↑OBL_{LOC})>'

4-2. Extension of Constraints for UNKNOWN

There are several grammatical devices and constraint conditions in LFG, but some of them are used in modified or extended forms for the effective use of UNKNOWNs.

Because Korean sentences can have multiple NPs with unknown GR values, f-structure with multiple UNKNOWN attributes should be permitted and the consistency constraint should be relaxed. KOSA has solved this problem without any change of the unification algorithm by attaching index numbers to the UNKNOWN attributes as UNKNOWN_1, UNKNOWN_2,... when they are inserted.

The completeness/coherence constraint sould be extended for sentences with multiple UNKNOWNs. This extension is similar to that stated in [8], but the number of UNKNOWN attributes can be more than one here. So the extended completeness/ coherence constraint is as following: The number of UNKNOWN attributes should be less than or equal to the number of unsaturated grammatical functions of the PRED value for the intermediate f-structures, and should be equal for the final f-structures.

4-3. Heuristics for GR-Determination of the UNKNOWNs

For the complete analysis, the hidden GR values of the UNKNOWN attributes should be determined. KOSA uses three heuristics to determine them.

First is the simple mapping method. If there is only one UNKNOWN attribute in an f-structure and one unsaturated grammatical function, the GR value of the UNKNOWN is determined as the unsaturated grammatical function.

If the number of the UNKNOWN attributes is N(more than one), there should be also N unsaturated grammatical functions. Then they can be matched in N! different ways. To select the most proper mapping, two heuristics are used.

One heuristic is the agreement-point comparison method. The lexicon informations for nouns contain the semantic markers. They are transmitted to the values of UNKNOWN attributes. Each unsaturated grammatical function has the agreement-point information for each semantic feature on range [-1.0..1.0]. This is also given from the lexicon. For each mapping, the sum of agreement-points is calculated and the mapping of the highest score is selected. Because the number N is not so large, this heuristic does not bring a heavy overhead on examination.

The other heuristic is used when the agreement-points of several mappings are tied at the highest. Although NPs have almost free order in Korean, we can find the common word orders among them. The orders are not indispensable, but usual sentences follow them. So we can use these common word orders to determine the GR values of the UNKNOWNs. To find the order between the NPs without referring to the c-structure, we can utilize the index number attached to the UNKNOWNs.

5. Sentence Analysis Example of KOSA

In this chapter, the analysis steps of KOSA will be illustrated for following example. The first line of the example is the real Korean input, the second line is the input sentence written in Roman alphabet, the third line shows the meanings of morphemes belonging to the noun or verb category, and the last line shows the meaning of input sentence. For easy understanding, we replaced the Korean characters with Italic-style and morphemes belonging to the noun or verb category with English word.

우리가 탄 비행기는 서울에 도착했다.
woori-ga ta-n bihaenggi-neun Seoul-e dochakha-et-da.
we take-on airplane Seoul arrive
The airplane which we took on arrived at Seoul.

6. Conclusion

We have introduced the structure of KOSA, a natural language parser for Korean, and discussed some related issues. In the design of KOSA, the overall concept of LFG formalism is adopted, and LFG is confirmed to meet well for the grammatical characteristics of Korean. But some additional concepts for analysis are developed for KOSA further. Among them, the usages of verbal complex category and some heuristics concerned with the UNKNOWN attributes are formulated and discussed. In English, there are similar grammatical functions to the UNKNOWN such as TOPIC. But Korean NPs are far more flexible and free from the restriction of grammatical structures. And sentences with multiple UNKNOWNs are also common. So the heuristics that meet well for Korean are necessary, and the heuristics shown here can also be used to recover the omitted NPs.

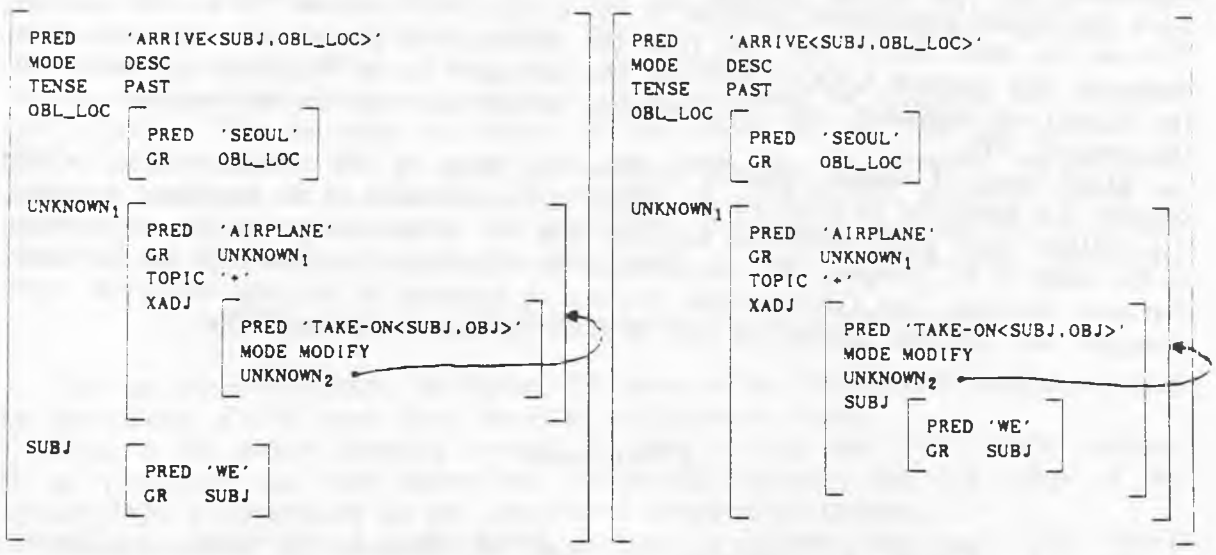
Main issues of current research includes the usage of NP tokens, each of which consists of a noun and its following postnouns, and replacement of the functional schemata '$(\uparrow(\downarrow GR))=\downarrow$' with a GR-determine function. The NP token concept has the same origin as the usage of VC category, and can provides the reduction of overhead for the CstrAnal/ FstrAnal modules. The GR-determine function is expected to be very useful for more complete and efficient analysis of the relations between verbs and NPs.

[References]

1. Sag, I.A., Kaplan, R., Karttunen, L., Kay, M., Pollard, C., Sieber, S., Zaenen, A., "Unification and Grammatical Theory", CSLI, 1987.
2. Sieber, S.M., An Introduction to Unification-Based Approaches to Grammar, pp.5-7, CSLI Lecture Notes No.4, 1986.
3. Bresnan, J.(Ed.), The Mental Representation of Grammatical Relations, Cambridge Mass.: MIT Press, 1982.
4. Kaplan, R.M. and Bresnan, J., "Lexical Functional Grammar: a formal system for grammatical representation", 1982, pp.173-281, in Bresnan, J.(Ed.), The Mental Representation of Grammatical Relations, Cambridge Mass.: MIT Press, 1982.
5. Bresnan, J., "The Passive and Lexical Theory", 1982, pp.3-86, in Bresnan, J.(Ed.), The Mental Representation of Grammatical Relations, Cambridge Mass.: MIT Press, 1982.
6. Wescoat, M.T., "Practical Instructions for Working with the Formalism of Lexical Functional Grammar", pp.1-37, in Bresnan, J.(Ed.) Lexical Functional Grammar, Stanford University, 1987.
7. Pereira, F.C.N. and Waren, D.H.D., "Definite Clause Grammars for Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks", 1980, pp.231-278, Artificial Intelligence 13.
8. Peter Sells, Lectures on Contemporary Syntactic Theory, pp.135-191, CSLI Lecture Notes No.3, 1985.
9. Ishikawa, A., Complex Predicates and Lexical Operations in Japanese, pp.64-83, University Microfilm International, 1985.
10. 남기심, 고영근, 표준국어문법론, pp.93-104, 탑출판사, 1985.

(↑TENSE) = PAST
(↑MODE) = DESC

- after FstrExtract phase: two alternative f-structures are generated as below

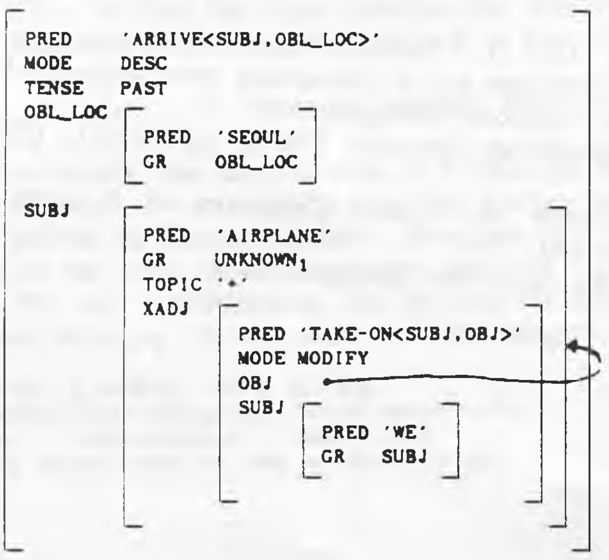


```
PRED      'ARRIVE<SUBJ,OBL_LOC>'              PRED      'ARRIVE<SUBJ,OBL_LOC>'
MODE      DESC                                MODE      DESC
TENSE     PAST                                TENSE     PAST
OBL_LOC   [PRED  'SEOUL'  ]                   OBL_LOC   [PRED  'SEOUL'  ]
          [GR    OBL_LOC  ]                             [GR    OBL_LOC  ]
UNKNOWN1  [PRED  'AIRPLANE'                   UNKNOWN1  [PRED  'AIRPLANE'
          [GR    UNKNOWN1                               [GR    UNKNOWN1
          [TOPIC '+'                                    [TOPIC '+'
          [XADJ  [PRED 'TAKE-ON<SUBJ,OBJ>']             [XADJ  [PRED 'TAKE-ON<SUBJ,OBJ>'
          [      [MODE MODIFY           ]               [      [MODE MODIFY
          [      [UNKNOWN2              ]               [      [UNKNOWN2
SUBJ      [PRED 'WE']                                   [      [SUBJ  [PRED 'WE']
          [GR   SUBJ]                                   [      [      [GR   SUBJ]
```

- after FstrCheck phase: final f-structure

left alternative: rejected as illegal
<SUBJ,OBL_LOC> : {OBL_LOC,UNKNOWN$_1$,SUBJ}
=> coherency constraint violation
<SUBJ,OBJ> : {UNKNOWN$_2$}
=> completeness constraint violation

right alternative: selected
<SUBJ,OBL_LOC> : {OBL_LOC,UNKNOWN$_1$}
=> UNKNOWN$_1$ turns out to be SUBJ
<SUBJ,OBJ> : {UNKNOWN$_2$,SUBJ}
=> UNKNOWN$_2$ turns out to be OBJ



```
PRED      'ARRIVE<SUBJ,OBL_LOC>'
MODE      DESC
TENSE     PAST
OBL_LOC   [PRED  'SEOUL'  ]
          [GR    OBL_LOC  ]
SUBJ      [PRED  'AIRPLANE'
          [GR    UNKNOWN1
          [TOPIC '+'
          [XADJ  [PRED 'TAKE-ON<SUBJ,OBJ>'
          [      [MODE MODIFY
          [      [OBJ
          [      [SUBJ  [PRED 'WE']
          [      [      [GR   SUBJ]
```

In section 4, several problems of the first method are described. And in the successive section, a modified implementation is showed. We explain three modifications. The first modification is to uses the information of any proper attributes on the node. This information is manually described in augmented rewriting-rules. The information consists of the names of relations and the calculation of arguments for the relations. The second modification is to raises the priority of the structure which appears the cooccurrences judged solely as correct all through the period of acquisition. The third is to collect cooccurrence data on two phases.

In section 6, we show the analysis performance of the modified version on our experiment. The results show that modified version shows better performances than the previous version, when relatively small number of acquired data is utilized. Furthermore we show another experiment which measures the appearance rate of acquired cooccurrences data in each parsed text with the measurement of an analysis performance in each text. By this measurement, we can confirm that texts having high appearance ratio are analyzed more accurately than texts having low appearance ratio.

2. Features of the utilized parser

In our method, cooccurrence data are collected with a parser. Here, we utilize a parser of a English-to-Japanese machine translation system named KATE. The analysis technique for a English sentence is based on augmented context free grammar like LINGOL. Cook-Kasami-Younger algorithm and Early algorithm are implemented with some fast parsing techniques[2] in this parser. Other features of the parser are :

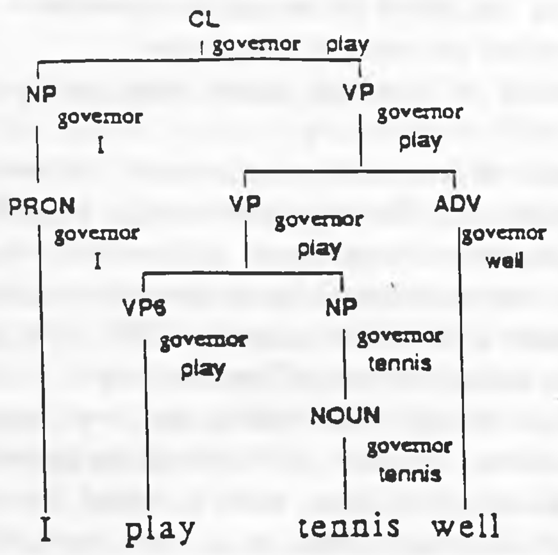
(1) Each node of syntactic trees generated by the parser has attributes information which is the meaning representation of the sub-tree governed by the node.

(2) On each node, a governor (the word which represents the phrase) is given as one of attributes.

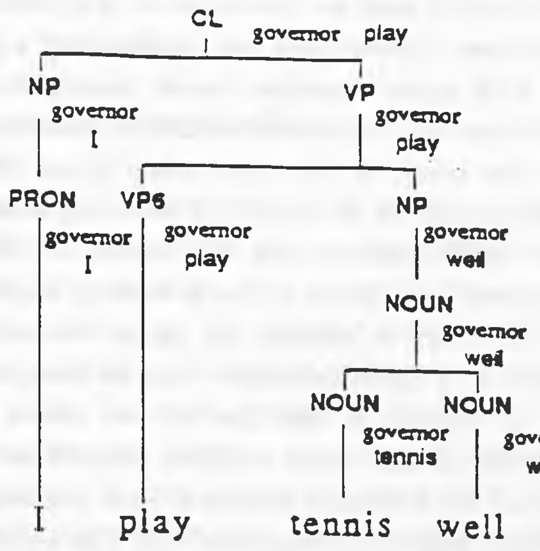
(3) We can register partial patterns of possible syntactic trees, and when a rule generates such a pattern on parsing stage, then the application of the rule is inhibited. These inhibiting patterns are used for the suppression of ambiguous trees.

Examples of generated trees and governors are showed bellow. [Fig 1, 2]

[Figure 1 An example of an analysis tree]



[Figure 2 An example of an analysis tree]



3. Acquisition and usage of relationships between governors in a simple version

Details of the first method are explained here. We call the program for this method a simple version. This version is more easily implemented than the modified version described in section 5, but lacks the accuracy in collecting cooccurrence data, We show this method for explanation purposes.

3.1 Discrimination procedure of a cooccurrence and maintenance of stored cooccurrence-data

Learning Cooccurrences by using a parser

Kazunori Matsumoto Hiroshi Sakaki Shingo Kuroiwa
KDD Kamifukuoka R&D Labs.
Saitama, Japan

ABSTRACT

This paper describes two methods for the acquisition and utilization of lexical cooccurrence relationships. Under these method, cooccurrence relationships are obtained from two kinds of inputs : example sentences and the corresponding correct syntactic structure. The first of the two methods treats a set of governors each element of which is bound to a element of sister nodes set in a syntactic structure under consideration, as a cooccurrence relationship. In the second method, a cooccurrence relationship name and affiliated attribute names are manually given in the description of augmented rewriting rules. Both methods discriminate correctness of cooccurrence by the use of the correct syntactic structure mentioned above.

Experiment is made for both methods to find if thus obtained cooccurrence relationship is useful for the correct analysis.

1. Introduction

Much attention should be paid for the role of minutely described grammar and real world knowledge in order to improve natural language analysis performance. In this respect, the authors have tried to acquire and use coocurrence data for the improvement of analysis performance. By combining a parser and an acquisition mechanism, we implemented a learning program of lexical cooccurrence data. The program has two kinds of inputs, example sentences and the corresponding correct structures. The related study of learning grammar from sentences and their semantic structure is conducted in LAS[1] (Language Learning System) by Anderson. He is of the opinion that most of grammars are derived from semantic structures. We advocate the use of syntactic structures, because information such as cooccurrence is a reflection of the real world and is easily derived from syntactic structure. Furthermore we implemented a parser to utilize the acquired lexical data.

This paper describes above mentioned two methods for acquiring lexical cooccurrences and also describes the experiment results of the methods.

The result of the experiments shows (1) a reduction of the number of alternative analysis trees (2) the increase on probability of selecting a correct analysis tree. The experiments might be influenced by the used sentences and the nature of the used grammar. However we believe that our methods proposed here is one of the promising ways to reflect real world knowledge to sentence analysis.

At first, we explain the parser we use. This parser is based on augmented CFG. And the parser produces a forest (multiple analysis trees), and selects a single structure from the forest.

In section 3, the first of the above methods is showed. The method has two features : (1) Comparing generated analysis structures with the correct structure which should be generated by a parser for a treated sentence, each sequence of the governors on sister nodes is judged into two cases, correct case or wrong one. (2) The sequence which is always judged as a wrong case through the period of acquisition, is utilized for reducing analysis trees generated by the parser.

Experiments are made to measure effects of the second feature above. The result shows : when the set of example sentences are equivalent to the set of analyzed sentences, very few ambiguous analysis trees are generated. Almost all the selections of generated trees, then, are successful. However when the set of examples are not equivalent to the set of analyzed sentences, only one third of ambiguous trees are eliminated and probability of selection decreases a little in comparison with a original (no action) case.

And we measure the transition of following three values as the analysis performance of the parser, with the amount of increasing inputted pairs as a parameter.

(a) average of the number of generated trees per a sentence

(b) probability of generating a correct analysis tree

(c) probability of selecting a correct analysis tree

We made two experiments to measure above values.

One is measured in the condition that the set of sentences for the acquisition program is equivalent to the set of sentences analyzed by the parser. Actually we can't make the set of inputted pairs equivalent to the set of model sentences in a practical occasion. Because of the monotonous increase of acquired cooccurrence data in each category, however, we consider the result of this experiment gives a prospective view of the effect of the filtering. We observe following results. [Fig.3]

(a) With the increase of inputted pairs, average of trees decreases almost monotonously. Finally, the average becomes approximate 1.0 starting from 2.5 at the beginning. (A in Fig.3 shows the reduction of trees)

(b) Probability of generating a correct tree severely goes down, when amount of inputted pairs are few. And finally the probability, of course, becomes equal to the probability initial.

(c) Probability of selecting a correct tree also goes down, when inputted pairs are few, and after number of inputted pairs exceeds one third of the number of final inputted pairs, the probability becomes better than that of the beginning. (C in Fig.3 shows the improvement)

The second experiment is made in the condition that the set of 2,400 sentences inputted for the acquisition program is not equivalent to the set of 800 sentences parsed. Following observation is made.

(a) With the increase of inputted pairs, the average of trees decrease with a somewhat nonmonotonic. (A in Fig.4 shows the reduction of number of trees)

(b) As in the case of the previous experiment, probability of generating a correct tree goes down severely at the beginning but does not resume the initial state. (B in Fig.4)

(c) Probability of selecting a correct tree also goes down at the beginning and, what is worse, the probability finally becomes lower than that of the initial state, in spite of the assisting effect of reducing ambiguities. (C in Fig.4)

4. Problems in the simple version

This section explains eight problems of the previous simple implementation.

Problem [1] : Meaningless and purposeless data acquired.

Because the previous version discriminates and classifies all the sequences of governors appearing in all the rewriting rules, the learning program acquires purposeless cooccurrence data from the governors which represents no cooccurrence relationships. For example, in the case of the rule TEXT → CL END, which means a clause and a end-mark make a sentence, the previous program obtains the sequence of governors of CL and END. However, this sequence is useless to be utilized for parsing.

Problem [2] : Cooccurrence data judged as to be always wrong but easily revised in the future

In accordance with the increase of inputted pairs for the leaning program, the sequence of governors judged as to be always wrong so far may encounter a case where the sequence is judged as to be correct. Probability of reclassification for acquired cooccurrence data varies with the rewriting-rule related to the acquired data. For instance, in Fig.2, a sequence ≪well≫ for the rule NP → NOUN is the sequence judged as wrong. If the discrimination for this sequence doesn't contradict any discrimination caused by inputted data for the learning program, this sequence is judged as to be always wrong and used for the filtering. However, we can easily mention the example where this filtering works adversely.

Problem [3] : There exists the governor which is independent of a cooccurrence.

Insufficient semantic analysis causes the generation of unproper syntactic trees, like one in Fig.2. Our program compares each generated tree with the correct tree of corresponding sentence, and classifies the sequence of governors appearing on sister nodes into two classes for each rewriting-ru. When the sequence of the governors occurs on the following two conditions, the program judges the sequence as a correct cooccurrence data, otherwise judges as a wrong cooccurrence data.

(1) the same rule which fields the remarked sentence is applied in the correct syntactic tree ;

(2) In each sub-node of the applied rule, the terminal words sequence rewritten is the same as the terminal words sequence rewritten by correct applications in the correct tree.

If we assume the tree in Fig.1 is a correct syntactic tree we obtain, from the trees in Fig.1 and Fig.2, we obtain following correct cooccurrence data and wrong cooccurrence data.

Correct cooccurrence data from Fig.1 & Fig.2

≪	I	play	≫	for	CL	→	NP VP
≪		I	≫	for	NP	→	PRON
≪	play	well	≫	for	VP	→	VP ADV
≪	play	tennis	≫	for	VP	→	VP6 NP
≪		tennis	≫	for	NP	→	NOUN

Wrong cooccurrence data from Fig.1 & Fig.2

≪	play	well	≫	for	VP	→	VP6 NP
≪		well	≫	for	NP	→	NOUN
≪	tennis	well	≫	for	NOUN	→	NOUN NOUN

In accordance with this discrimination procedure, the sequence of governors may be judged as correct cooccurrence data in one example sentence and be judged as wrong cooccurrence data in another. So the program stores the sequence of the governors into three categories. First is the set of sequences being always judged as correct cooccurrence data by the discrimination procedure. The second is the set of sequences being always judged as wrong cooccurrence data. And the last is the set of sequences being judged as correct cooccurrence data in one or more cases and judged as wrong in one or more cases. Our learning program maintains these three categories through the period when example sentences and their correct structures are inputted. In this section, we simply call the sequence of governors as cooccurrence data.

<u>3.2 Experiment for acquiring cooccurrence data</u>

We make an experiment for acquiring cooccurrence data with the use of the above mentioned learning program. About 3,200 example sentences are collected from a English grammar text[3] and example sentences in a dictionary. We assume each example sentence has a situation free interpretation, so if semantics analysis is successful, very few ambiguous analysis trees are generated.

We measure the number of cooccurrence data in each category at every 50 inputted pairs of sentences and correct structures. We observe that :

(1) Each number of acquired cooccurrence data increases monotonously.

(2) Finally, from 3,200 sentences, the program acquires about 10,0000 kinds of cooccurrence data belonging to the first category, about 5,000 kinds and 4,000 kinds respectively belonging to the second and the third.

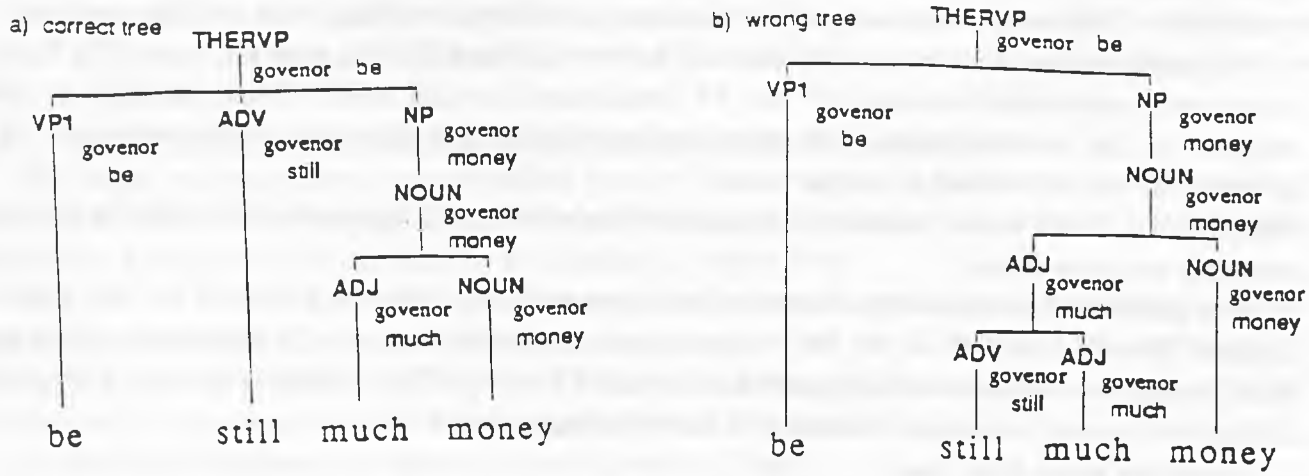
However, our detailed observation finds a part of acquired cooccurrence data purposeless or mischievous. This problem is described later in section 4.

<u>3.3 Filtering technique based on the cooccurrence data</u>

We implemented the parser which utilizes acquired cooccurrence data. When the sequence of the governors appearing on a rule application belongs to the set of acquired cooccurrence judged as to be always wrong, the parser doesn't apply the rule. This paradigm suppresses the excessive application of rules and reduces generated trees. So the probability of selecting proper analysis tree may increase. We call this paradigm 'Filtering based on cooccurrence (judged as to be always wrong).'

cooccurrence data from this wrong tree, if we consider ≪still much≫ for a rule ADJ → ADV ADJ is judged as to be always wrong.

[Figure 6 Partial trees for "There is stil much money"]

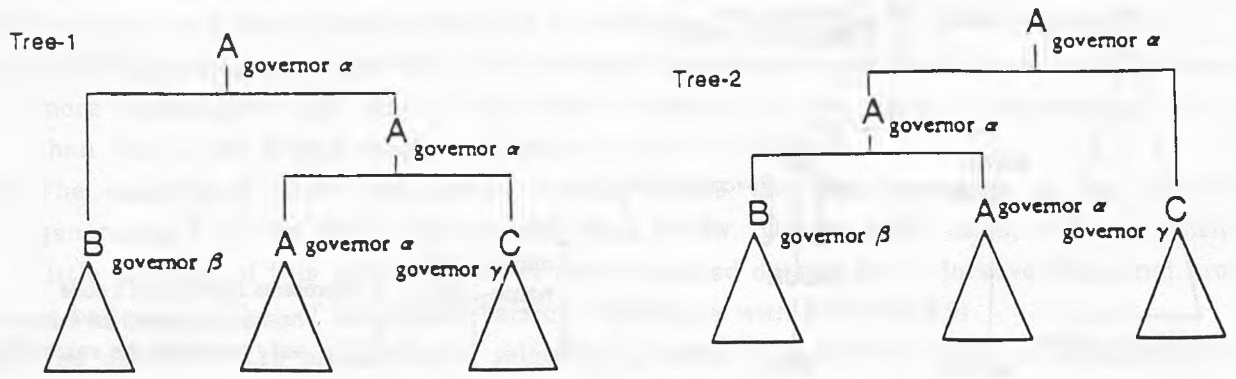


Problem [8] : Ambiguity in rule application orders causes the cooccurrence data which should be judged as correct to be judged as wrong.

We assume two rewriting rules, A → B A and → A C. If categories appear in the sequence of B A C and applications of each rule are successful, the parser generates two trees [Fig.7]. Appearance here of attributes related to the cooccurrence is assumed in Fig.7.

According to the discrimination procedure in section 3. 1, regardless of whether the tree-1 is correct or tree-2 is correct, both the cooccurrence ≪β α≫ for A → B A and the cooccurrence ≪γ α≫ for A → C A become purposeless.

[Figure 7 Two ambiguous trees]



5. Modified version of learning and usage

This section shows a modified version of the previous program. This modified version solves the problem [1]~[7] in the previous section. Only problem [8] is out of scope, but we have a basic idea to reduce this kind of ambiguity with the use of the inhibited pattern technique in section 2.

Three major modification is described bellow.

Modification [1] : Manual description of cooccurrence names and their attributes names in rewriting rules.

Rich input data is required by the system in order to determine what relations exists or what attributes are used in each relation. Therefore we consider that the kind of a cooccurrence relationship and the names of used attributes which appear in the cooccurrence should be described manually for the sake of effective learning by examples.

For this reason, we now extend the description method of the rewriting rules used in the former version. In this extended description, a cooccurrence relationship is depicted as a function of any

In the case of the rule CL → NP ADV VP, which means a noun phrase and adverb and a verb phrase make a clause, the governors of NP and VP have a cooccurrence relationship. But the governor of ADV is almost independent of this relationship.

Problem [4] : The same relationship of cooccurrence in different rewriting rules can't be dealt with.

For example, the cooccurrence relationship between NP and VP for a rule CL → NP VP and the cooccurrence relationship between NP and VP for a rule CL → NP ADV VP are identified as different relations by the previous version. However, dealing with both relationships as the same will be more advisable for the utilization of cooccurrence.

Problem [5] : There exists cooccurrence relationships which can't be represented with the sequence of governors on sister nodes.

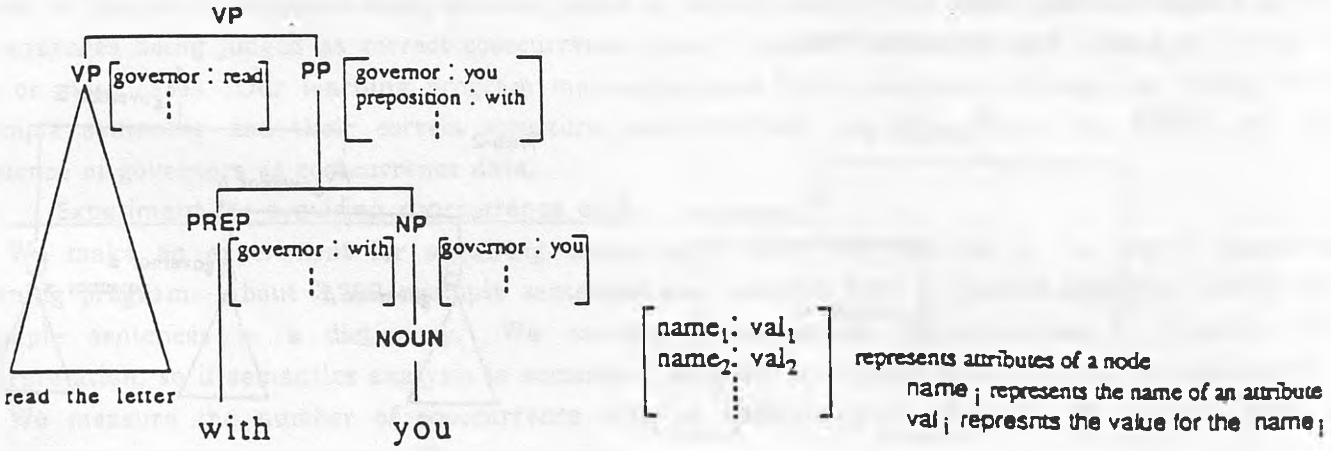
This problem is considerably affected by the grammar used. For the case of Fig.5, we explain this problem. From a rule VP → VP PP (which means a noun phrase and a prepositional phrase make a noun phrase), the sequence ≪read you≫ for the rule VP → VP PP is judged as correct, if the structure of Fig.5 is a correct structure. However, if the following sentence :

I read the letter from you.

is included in inputted pairs, a contradiction may occur. A prepositional phrase occurred in this sentence can modify a noun phrase. And if a rule VP → VP PP is applied wrongly, the sequence ≪read you≫ for the rule VP → VP PP is judged as wrong. Here, the acquired sequence becomes purposeless for to be utilized.

In this case, cooccurrence data for the rule VP → VP PP should be represented as the relation between the governor of VP, the preposition of PP, and the governor of PP.

[Figure 5 Part of a structre for "I read the letter with you"]



Problem [6] : In the previous version, information of cooccurrence data judged as to be always correct is not utilized.

Suppress of generated trees affects the selection of trees. Moreover information of correct cooccurrence data can improve the selection of a correct tree by the parser.

In the use of information of correct cooccurrence, however, following two problems become important.

Problem [7] : When a rule is applied at the occasion of an unproper application on lower level, the cooccurrence data which should be judged as correct may be judged as wrong.

We explain this problem with using Fig.6. In Fig.6, two analysis trees are generated. From a correct structure, the sequence ≪much money≫ for a rule NOUN → ADJ NOUN is judged as correct. And from a wrong structure, the same sequence is judged as wrong. So the data of this sequence becomes purposeless. If the application of a rule ADJ → ADV ADJ in the wrong structure fails, the sequence ≪much money≫ is only judged as correct in this sentence. We should not acquire are obtained in a single rewriting rule. Furthermore the result shows the number of 'cooccurrence data judged as to be correct and wrong simultaneously' is about one fourth of the simple version. This phenomenon is caused by manual descriptions for cooccurrence relationships, because these description suppress the acquisition of meaningless cooccurrence data and the acquisition of data easily reclassified.

We also examine the effect of the 2-pass acquisition. We observe that about 10% 'cooccurrence data judged as correct and wrong simultaneously' on the first phase are obtained as 'the data judged as to be always correct' on the second pass of acquisition.

6.2 Experiment of using acquiring cooccurrence data with modified version

We make two experiments with modified version like in section 3.3, in order to the transition of next three values : (a) average of the number of generated trees per a sentence (b) probability of generating a correct tree (c) probability of selecting a correct tree.

The first is under the condition that the set of sentences for acquisition is equivalent to the set of sentences for analysis. The second is for the condition that the set of sentences for acquisition is not equivalent to the set of analyzed sentences. We use the same set for acquisition and the same set for analysis as in experiments of the simple version on each two experiment.

At the first experiment we observe following results [Fig.9] :

(a) With the increase of inputted pairs, the average of generated trees decreases monotonously like in the experiment for the simple version. But at the final state, the effect of reducing the number of trees is less than that of the simple version. (Compare with A in Fig.3 ,8)

(b) When amount of inputted data are few, adverse effect of failing to generate a correct tree in the modified version is less than that in the simple version. Furthermore the range of fluctuation in the probability through this experiment is less than that in the simple version.

(c) When amount of inputted data is few, the probability of selecting a correct tree increases, which is differ from the simple version. The probability at the final state is lower than that of the simple version. (Compare with C in Fig.3,8)

And we observe following results [Fig.10] at the second experiment :

(a) With increase of inputted data, the average of generated trees also decreases. This decrease is more monotonous than that of the simple version, but the effect of suppressing trees is less than that of the simple version. (Compare with A in Fig.4,9)

(b) The experiment under the simple version shows the sever decrease of the probability of generating a correct tree, when inputted data is few. On the other hand, this experiment shows little decrease of this probability even when inputted data is few. Moreover the final probability is better than that of the simple version. (Compare with B in Fig.4,9)

(c) The decline of the probability of selecting a correct tree is very slight in comparison with the simple version, when inputted data are few. The final probability by this modified version slightly exceed that by the simple version. (Compare with C in Fig.4,9)

6.3 Performance analysis for the ratio of acquired cooccurrence data

We define the proportion of cooccurrence data obtained through the learning by examples to the cooccurrence data appearing in a parsed text as the ratio of acquired cooccurrence data. This section describe the experiment which treats the relation between analysis performance and the ratio of acquired cooccurrence data.

We choose 2,400 sentences for acquisition and six variations of sentence sets for analysis. Here, each of six sets is not equivalent to the set for acquisition. At first, we measure the ratios of acquired cooccurrence data for each of six sets, and measure performance for each of six sets with the use of acquired cooccurrence data. By these measurement we obtain following prospective view through the experiment.

When we compare, for each of those six sets, differences between the average of the number of generated trees by the parser without cooccurrence data and that with cooccurrence data, the difference

attributes in existing nodes and, moreover, these attributes used are depicted as functions of any attributes in all the nodes.
The program of modified version deals with cooccurrence data as bellow :

In the phase of acquisition, the program decide the name of cooccurrence and the names of used attributes in the cooccurrence, in accordance with the description of a rewriting rule.

Acquired cooccurrence data is judged similarly like in the previous method, and stored into three categories like in the simple version.

We show how the modified version solves the problems [1]~[5] mentioned in the previous section utilizing following examples.

Problem [1] : In the rules such as TEXT → CL END, which have no cooccurrence there should be no description of cooccurrence.

Problem [2] : In the rules such as NP → NOUN, which tend to be easily revised the cooccurrence should not be utilized.

Problem [3] : In the rule of CL → NP ADV VP, cooccurrence data should be described with an attribute of NP and an attribute of VP, because we consider cooccurrence relation exists between a governor of NP and a governor of VP.

Problem [4] : We should declare the same cooccurrence in both rules of CL → NP ADV VP and CL → NP VP.

Problem [5] : When we declare the cooccurrence in the rule VP → VP PP, we should choose the governor of VP, the preposition of PP, and the governor of the PP as the elements of the cooccurrence.

Modification [2] : Utilization of cooccurrence data judged as to be always correct in selection phase.

In problem [6] we pointed out the effect of using cooccurrence data judged as to be always correct. Hence, we implement next paradigm :

When a cooccurrence data judged as to be always correct occurs in a generated tree on the selection phase, the parser gives the tree a high priority for the selection purpose.

Modification [3] : Acquisition for cooccurrence data is executed in 2-passes.

To solve the problem [7], we modified the procedure of acquiring cooccurrence data. On the first pass of acquisition, the acquisition of cooccurrence is executed as in the previous version. After the end of the first pass, the modified program clear the both storages of 'cooccurrence judged as to be always correct' and 'cooccurrence judged as to be correct and wrong simultaneously.' This program executes the acquisition again from the beginning of inputted pairs with the filtering based on acquired cooccurrence.

In the case of Fig.6, if the sequence ≪much time≫ is judged as to be always wrong at the end of the first pass of acquisition, a wrong tree in Fig.6 can't be generated by the parser on the second pass of acquisition. For this reason, the sequence ≪much time≫ is not judged as wrong in this sentence.

6. Acquisition and usage of cooccurrence data in the modified version

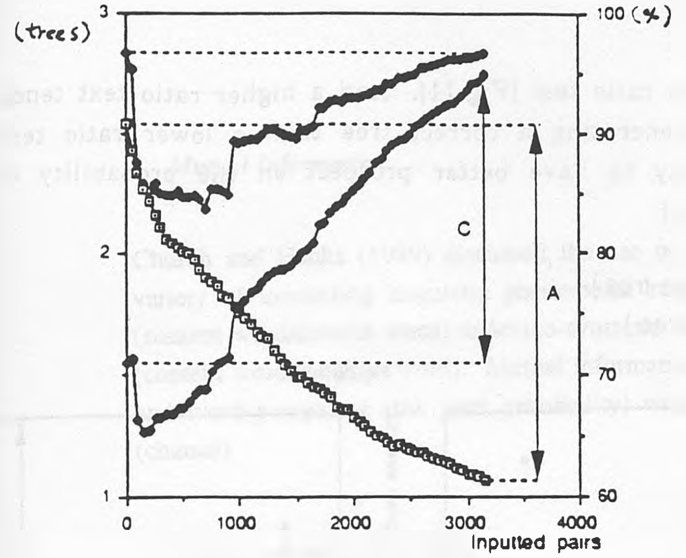
The result treated here is the one for the modified version. We make an experiment with the same example sentences as used for the simple version, but the used grammar is slightly different. The authors believe this slight difference is negligible for the comparison with the simple version and the modified version.

6.1 Experiment of acquiring cooccurrence data by the modified version

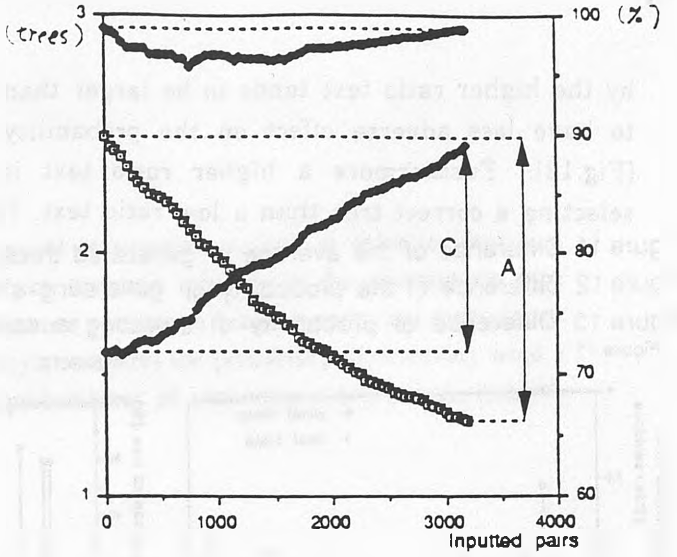
According to the same way of treatment in the simple version, we measure the number of each stored cooccurrence data for the modified version. The result shows each stored data increases monotonously with the increase of inputted pairs. [Fig.8]

The result is similar to that of the simple version in 3. 1. More 'cooccurrence data judged as to be always correct' and more 'cooccurrence data judged as to be always wrong' are obtained in the modified version than in the simple version. This may be the reason why one or more cooccurrence relationships

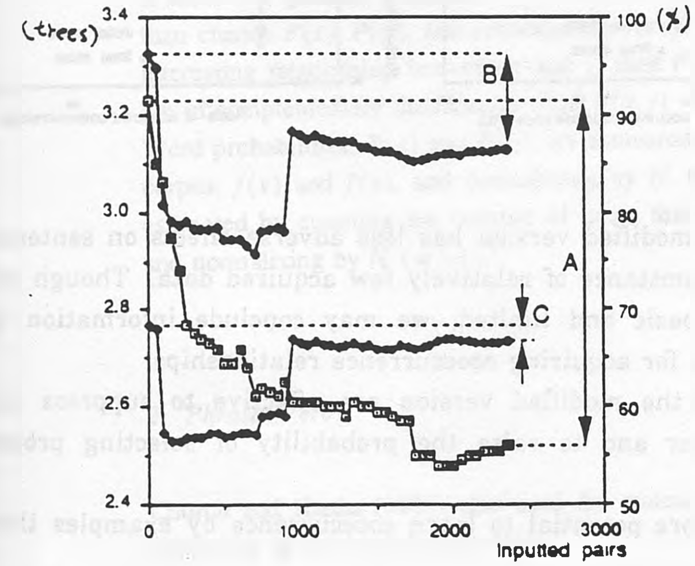
[Figure 3 Performance of the simple version]
(Learing sentences = Parsed sentences)



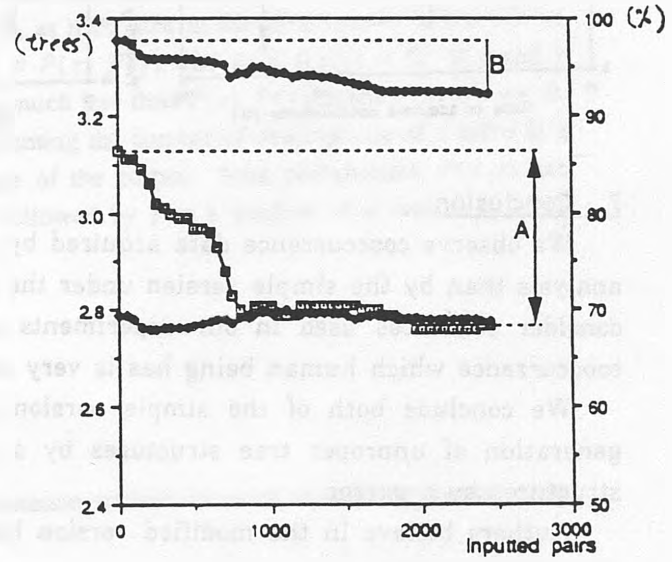
[Figure 9 Performance of the modified version]
(Learing sentences = Parsed sentences)



[Figure 4 Performance of the simple version]
(Learing sentences ≠ Parsed sentences)



[Figure 10 Performance of the modified version]
(Learing sentences ≠ Parsed sentences)



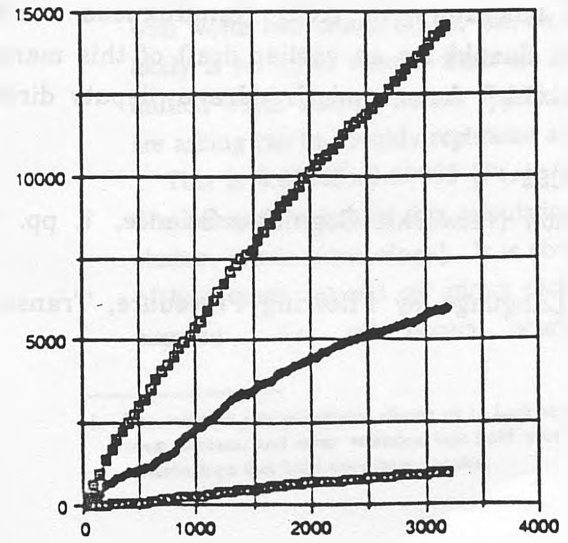
In Fig. 3 , 4 , 9 , 10

-□- Average of generated trees per a sentence

-◇- Probability of generating a correct tree (%)

-◆- Probability of selecting a correct tree (%)

[Figure 8 Number of the three kinds of cooccurrence data]



■ cooccurrence data judged as to be always correct
◆ cooccurrence data judged as to be always wrong
▣ cooccurrence data judged as to be correct and wrong simultaneoulsly

by the higher ratio text tends to be larger than a low ratio test [Fig.11]. And a higher ratio text tends to have less adverse effect on the probability of generating a correct tree than a lower ratio text [Fig.12]. Furthermore a higher ratio text is likely to have better prospect on the probability of selecting a correct tree than a low ratio text. [Fig.13]

[Figure 11 Difference of the average of generated trees]

[Figure 12 Difference of the probability of generating a correct tree]

[Figure 13 Difference of probability of selecting a correct tree]

Figure 11

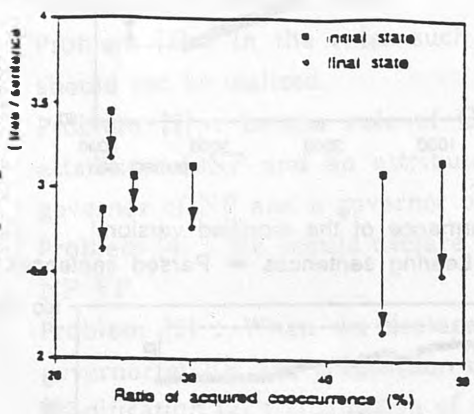


Figure 12

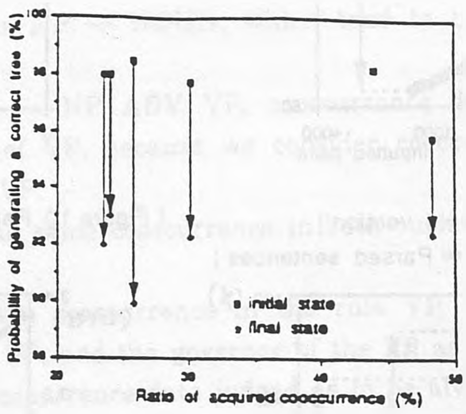
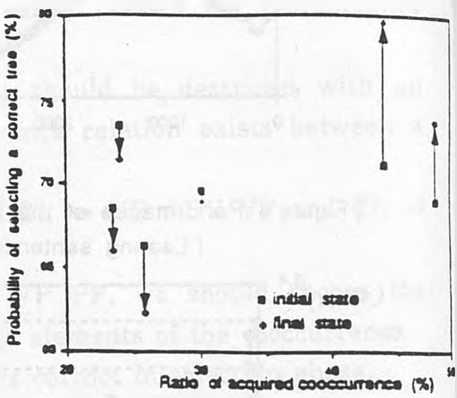


Figure 13



7. Conclusion

We observe cooccurrence data acquired by the modified version has less adverse effects on sentence analysis than by the simple version under the circumstance of relatively few acquired data. Though we consider sentences used in our experiments are basic and limited, we may conclude information of cooccurrence which human being has is very useful for acquiring cooccurrence relationships.

We conclude both of the simple version and the modified version are effective to suppress the generation of unproper tree structures by a parser and to raise the probability of selecting proper structures by a parser.

Authors believe in the modified version has more potential to learn cooccurrence by examples than the simple version.

ACKNOWLEDGEMENT

The authors wish to thank the members of AI laboratory in KDD Kamifukuoka R&D Labs, especially Tohru Asami, Kazuo Hashimoto, and Masami Suzuki for an earlier draft of this manuscript. We also thank Dr. Ono, director of KDD Kamifukuoka R&D Labs, and Dr. Urano, deputy director for giving a chance of our research and encouraging us.

REFERENCES

1. Anderson, J. : Introduction of Augmented Transition Networks, Cognitive Science, 1, pp. 125-157 (1977).
2. Sakaki, H. et.al : A Parsing method of Natural Language by Filtering Procedure, Transaction of the IECE of Japan, E69, pp, 1114-1124 (1986).

1. *Mutual Information*

Church and Hanks (1989) discussed the use of the mutual information statistic in order to identify a variety of interesting linguistic phenomena, ranging from semantic relations of the doctor/nurse type (content word/content word) to lexico-syntactic co-occurrence constraints between verbs and prepositions (content word/function word). Mutual information, $I(x;y)$, compares the probability of observing word x and word y *together* (the joint probability) with the probabilities of observing x and y *independently* (chance).

$$I(x;y) \equiv \log_2 \frac{P(x,y)}{P(x)\ P(y)}$$

If there is a genuine association between x and y, then the joint probability $P(x,y)$ will be much larger than chance $P(x)\ P(y)$, and consequently $I(x;y) >> 0$, as illustrated in the table below. If there is no interesting relationship between x and y, then $P(x,y) \approx P(x)\ P(y)$, and thus, $I(x;y) \approx 0$. If x and y are in complementary distribution, then $P(x,y)$ will be much less than $P(x)\ P(y)$, forcing $I(x;y) << 0$. Word probabilities, $P(x)$ and $P(y)$, are estimated by counting the number of observations of x and y in a corpus, $f(x)$ and $f(y)$, and normalizing by N, the size of the corpus. Joint probabilities, $P(x,y)$, are estimated by counting the number of times that x is followed by y in a window of w words, $f_w(x,y)$, and normalizing by $N\ (w-1)$.[1]

2. *Phrasal Verbs*

Church and Hanks (1989) also used the mutual information statistic in order to identify phrasal verbs, following up a remark by Sinclair:

> "How common are the phrasal verbs with *set*? *Set* is particularly rich in making combinations with words like *about, in, up, out, on, off*, and these words are themselves very common. How likely is *set off* to occur? Both are frequent words; [*set* occurs approximately 250 times in a million words and] *off* occurs approximately 556 times in a million words... [T]he question we are asking can be roughly rephrased as follows: how likely is *off* to occur immediately after *set*? ... This is 0.00025×0.00055 [$P(x)\ P(y)$], which gives us the tiny figure of 0.0000001375 ... The assumption behind this calculation is that the words are distributed at random in a text [at chance, in our terminology]. It is obvious to a linguist that this is not so, and a rough measure of how much *set* and *off* attract each other is to compare the probability with what actually happens... *Set off* occurs nearly 70 times in the 7.3 million word corpus

1. The window size parameter allows us to look at different scales. Smaller window sizes will identify fixed expressions (idioms), noun phrases, and other relations that hold over short ranges; larger window sizes will highlight semantic concepts and other relationships that hold over larger scales.

Parsing, Word Associations and Typical Predicate-Argument Relations

Kenneth Church
William Gale
Patrick Hanks
Donald Hindle

Abstract

There are a number of collocational constraints in natural languages that ought to play a more important role in natural language parsers. Thus, for example, it is hard for most parsers to take advantage of the fact that *wine* is typically *drunk, produced,* and *sold,* but (probably) not *pruned.* So too, it is hard for a parser to know which verbs go with which prepositions (e.g., *set up*) and which nouns fit together to form compound noun phrases (e.g., *computer programmer*). This paper will attempt to show that many of these types of concerns can be addressed with syntactic methods (symbol pushing), and need not require explicit semantic interpretation. We have found that it is possible to identify many of these interesting co-occurrence relations by computing simple summary statistics over millions of words of text. This paper will summarize a number of experiments carried out by various subsets of the authors over the last few years. The term *collocation* will be used quite broadly to include constraints on SVO (subject verb object) triples, phrasal verbs, compound noun phrases, and psycholinguistic notions of word association (e.g., *doctor/nurse*).

- *to/in*: alluding/vbg, adhere/vb, amounted/vbn, relating/vbg, amounting/vbg, revert/vb, reverted/vbn, resorting/vbg, relegated/vbn

- *to/to*: obligated/vbn, trying/vbg, compelled/vbn, enables/vbz, supposed/vbn, intends/vbz, vowing/vbg, tried/vbd, enabling/vbg, tends/vbz, tend/vb, intend/vb, tries/vbz

Thus, we see there is considerable leverage to be gained by preprocessing the corpus and manipulating the inventory of tokens.

4. Preprocessing with a Syntactic Parser

Hindle has found it useful to preprocess the input with the Fidditch parser (Hindle 1983) in order to ask about the typical arguments of verbs. Thus, for any of verb in the sample, we can ask what nouns it takes as subjects and objects. The following table shows the objects of the verb *drink* that appeared at least two times in a sample of six million words of AP text, in effect giving the answer to the question "what can you drink?" Calculating the co-occurrence weight for *drink*, shown in the third column, gives us a reasonable ranking of terms, with *it* near the bottom. This list of drinkable things is intuitively quite good.

Object	**Frequency**	**Mutual Information**
<quantity> beer	2	12.34
tea	4	11.75
Pepsi	2	11.75
champagne	4	11.75
liquid	2	10.53
beer	5	10.20
wine	2	9.34
water	7	7.65
anything	3	5.15
much	3	2.54
it	3	1.25
<quantity>	2	1.22

A standard alternative approach to the classification of entities is in terms of a hierarchy of types. The biological taxonomy is the canonical example: a penguin is a bird is a vertebrate and so on. Such "is-a" hierarchies have found a prominent place in natural language processing and knowledge representation because they allow generalized representation of semantic features and of rules. There is a wide range of problems and issues in using "is-a" hierarchies in natural language processing, but two especially recommend that we investigate alternative classification schemes like the one reported here. First, "is-a" hierarchies are large and complicated and expensive to acquire by hand. Attempts to automatically derive these hierarchies for words from existing dictionaries have been only partially successful (Chodorow, Byrd, and Heidorn 1985). Yet without a comprehensive hierarchy, it is difficult

Some Interesting Associations with "Doctor"
in the 1987 AP Corpus (N = 15 million; w = 6)

I(x; y)	f(x, y)	f(x)	x	f(y)	y
8.0	2.4	111	honorary	621	doctor
8.0	1.6	1105	doctors	44	dentists
8.4	6.0	1105	doctors	241	nurses
7.1	1.6	1105	doctors	154	treating
6.7	1.2	275	examined	621	doctor
6.6	1.2	1105	doctors	317	treat
6.4	5.0	621	doctor	1407	bills
6.4	1.2	621	doctor	350	visits
6.3	3.8	1105	doctors	676	hospitals
6.1	1.2	241	nurses	1105	doctors

Some Less Interesting Associations with "Doctor"

I(x; y)	f(x, y)	f(x)	x	f(y)	y
-1.3	1.2	621	doctor	73785	with
-1.4	8.2	284690	a	1105	doctors
-1.4	2.4	84716	is	1105	doctors

$[P(x,y) = 70/(7.3\ 10^6) >> P(x)\ P(y)]$. That is enough to show its main patterning and it suggests that in currently-held corpora there will be found sufficient evidence for the description of a substantial collection of phrases... (Sinclair 1987b, pp. 151-152)

It happens that *set* ... *off* was found 177 times in the 1987 AP Corpus of approximately 15 million words, about the same number of occurrences per million as Sinclair found in his (mainly British) corpus. Quantitatively, $I(set;off) = 3.7$, indicating that the probability of *set* ... *off* is $2^{3.7} = 13$ times greater than chance. This association is relatively strong; the other particles that Sinclair mentions have scores of: *about* (-0.9), *in* (0.6), *up* (4.6), *out* (2.2), *on* (1.0) in the 1987 AP Corpus of 15 million words.

3. *Preprocessing the Corpus with a Part of Speech Tagger*

Phrasal verbs involving the preposition *to* raise an interesting problem because of the possible confusion with the infinitive marker *to*. We have found that if we first tag every word in the corpus with a part of speech using a method such as Church (1988) or DeRose (1988), and then measure associations between tagged words, we can identify interesting contrasts between verbs associated with a following preposition *to/in* and verbs associated with a following infinitive marker *to/to*. (Part of speech notation is borrowed from Francis and Kucera (1982); in = preposition; to = infinitive marker; vb = bare verb; vbg = verb + ing; vbd = verb + ed; vbz = verb + s; vbn = verb + en.) The score identifies quite a number of verbs associated in an interesting way with *to*; restricting our attention to pairs with a score of 3.0 or more, there are 768 verbs associated with the preposition *to/in* and 551 verbs with the infinitive marker *to/to*. The ten verbs found to be most associated before *to/in* are:

frequencies of frequences (the number of bigrams with count r). Then r^*, the estimated expected value of r in similar corpus of the same size, is

$$r^* = N \times E(Pr(x\ y)) = (r+1)\ \frac{N_{r+1}}{N_r}$$

and the variance of r is

$$\sigma^2(r) = N^2 \sigma^2(Pr(x\ y)) = r^* \ (1 + (r+1)^* - r^*)$$

6. *Just a Powerful Tool*

Although it is clear that the statistics discussed above can be extremely powerful aids to a lexicographer, they should not be overrated. We do not aim to replace lexicographers with self-organizing statistics; we merely hope to provide a set of tools that could greatly improve their productivity. Suppose, for example, that a lexicographer wanted to find a set of words that take sentential complements. Then it might be helpful to start with a table of t-scores such as:

t	x	y
74.0	said	that
50.9	noted	that
43.3	fact	that
41.9	believe	that
40.7	found	that
40.1	is	that
40.0	reported	that
39.5	adding	that
38.6	Tuesday	that
38.4	Wednesday	that

It might be much quicker for a lexicographer to edit down this list than to construct the list from intuition alone. It doesn't take very much time to decide that *Tuesday* and *Wednesday* are less interesting than the others. Of course, it might be possible to automate some of these decisions by appropriately preprocessing the corpus with a part of speech tagger or a parser, but it will probably always be necessary to exercise some editorial judgment.

7. *Practical Applications*

The proposed statistical description has a large number of potentially important applications, including:

to use such classifications in the processing of unrestricted text. Secondly, for many purposes, even knowing the subclass-superclass relations is insufficient; it is difficult to predict which properties are inherited from a superclass and which aren't, and what properties are relevant in a particular linguistic usage. So for example, as noted above, despite the fact that both potatoes and peanuts are edible foods that grow underground, we typically *bake potatoes*, but *roast peanuts*. A distribution-based classification, if successful, promises to do better at least on these two problems.

5. Significance Levels

If the frequency counts are very small, the mutual information statistic becomes unstable. This is the reason for not reporting objects that appeared only once with the verb *drink*. Although these objects have very large mutual information scores, there is also a very large chance that they resulted from some quirk in the corpus, or a bug in the parser. For some purposes, it is desirable to measure confidence rather than likelihood. Gale and Church have investigated the use of a t-score instead of the mutual information score, as a way of identifying ''significant'' bigrams.

The following table shows a few significant bigrams ending with *potatoes*, computed from 44 million words of AP news wire from 2/12/88 until 12/31/88. The numbers in the first column indicate the confidence in standard deviations that the word sequence is interesting, and cannot be attributed to chance.

t	x	y
4.6	sweet	potatoes
4.3	mashed	potatoes
4.3	,	potatoes
4.0	and	potatoes
3.8	couch	potatoes
3.3	of	potatoes
3.3	frozen	potatoes
2.8	fresh	potatoes
2.8	small	potatoes
2.1	baked	potatoes

These numbers were computed by the following formula

$$t = \frac{E(Pr(x\ y)) - E(Pr(x)\ Pr(y))}{\sqrt{\sigma^2(Pr(x\ y)) + \sigma^2(Pr(x)\ Pr(y))}}$$

where $E(Pr(x\ y))$ and $\sigma^2(Pr(x\ y))$ are the mean and variance of the probability of seeing word x followed by word y. The means and variances are computed by the Good-Turing method (Good 1953).

Let r be the number of times that the bigram $x\ y$ was found in a corpus of N words, and let N_r be the

8. Alternatives to Collocation for Recognition Applications

There have been quite a number of attempts to use syntactic methods in speech recognition, beginning with the ARPA speech project and continuing on to the present. It might be noted, however, that there has not been very much success, perhaps because syntax alone is not a strong enough constraint on language use (performance). We believe that collocational constraints should play an important role in recognition applications, and attempts to ignore collocational constraints and use purely syntactic methods will probably run into difficulties.

Syntactic constraints, by themselves, though are probably not very important. Any psycholinguist knows that the influence of syntax on lexical retrieval is so subtle that you have to control very carefully for all the factors that really matter (e.g., word frequency, word association norms, etc.). On the other hand, collocational factors (word associations) dominate syntactic ones so much that you can easily measure the influence of word frequency and word association norms on lexical retrieval without careful controls for syntax.

There are many ways to demonstrate the relative lack of constraint imposed by syntax. Recall the old television game show, "The Match Game," where a team of players was given a sentence with a missing word, e.g, "Byzantine icons could murder the divine BLANK," and asked to fill in the blank the same way that the studio audience did. The game was 'interesting' because there are enough constraints in natural language so that there is a reasonably large probability of a match. Suppose, however, that we make our speech recognition device play the match game with a handicap; instead of giving the speech recognition device the word string, "Byzantine icons could murder the divine BLANK," we give the speech recognition device just the syntactic parse tree, [S [NP nn nns] [VP [AUX md] v [NP at jj BLANK]]], and ask it to guess the missing word. This is effectively what we are doing by limiting the language model to syntactic considerations alone. Of course, with this the handicap, the match game isn't much of a game; the recognition device doesn't have a fair chance to guess the missing word.

We believe that syntax will ultimately be a very important source of constraint, but in a more indirect way. As we have been suggesting, the real constraints will come from word frequencies and collocational constraints, but these questions will probably need to be broken out by syntactic context. How likely is it for this noun to conjoin with that noun? Is this noun a typical subject of that verb? And so on. In this way, syntax plays a crucial role in providing the relevant representation for expressing these very important constraints, but crucially, it does not provide very much useful constraint (in the information theoretic sense) all by itself.[2]

2. Much of the work on language modeling for speech recognition has tended to concentrate on search questions. Should we still be using Bates' island driving approach (Bates 1975), or should we try something newer such as Tomita's so-called generalized LR(k) parser (Tomita 1986)? We suggest that the discussion should concentrate more on describing the facts, and less on how they are enforced.

- enhancing the productivity of lexicographers in identifying normal and conventional usage,
- enhancing the productivity of computational linguists in compiling lexicons of lexico-syntactic facts,
- providing disambiguation cues for parsing highly ambiguous syntactic structures such as noun compounds, conjunctions, and prepositional phrases,
- retrieving texts from large databases (e.g., newspapers, patents), and
- constraining the language model both for speech recognition and optical character recognition (OCR).

Consider the optical character recognizer (OCR) application. Suppose that we have an OCR device such as (Kahan, Pavlidis, Baird 1987), and it has assigned about equal probability to having recognized "farm" and "form," where the context is either: (1) "federal ___ credit" or (2) "some ___ of." We doubt that the reader has any trouble specifying which alternative is more likely. By using the following probabilities for the eight bigrams in this sequence, a computer program can rely on an estimated likelihood to make the same distinction.

x	*y*	*Observations per million words*
federal	farm	0.50
federal	form	0.039
farm	credit	0.13
form	credit	0.026
some	form	4.1
some	farm	0.63
form	of	34.0
farm	of	0.81

The probability of the tri-grams can be approximated by multiplying the probabilities of the the two constituent bigrams. Thus, the probability of *federal farm credit* can be approximated as $(0.5\times10^{-6})\times(0.13\times10^{-6}) = 0.065\times10^{-12}$. Similarly, the probability for *federal form credit* can be approximiated as $(0.039\times10^{-6})\times(0.026\times10^{-6}) = 0.0010\times10^{-12}$. The ratio of these likelihoods shows that "farm" is $(0.065\times10^{-12})/(0.0010\times10^{-12}) = 65$ times more likely than "form" in this context. In the other context, "some ___ of," it turns out that "form" is 273 times more likely than "farm." This example shows how likelihood ratios can be used in an optical character recognition system to disambiguate among optically confusable words. Note that alternative disambiguation methods based on syntactic constraints such as part of speech are unlikely to help in this case since both "form" and "farm" are commonly used as nouns.

Church, K., and Hanks, P., (1989), "Word Association Norms, Mutual Information, and Lexicography," ACL Proceedings.

DeRose, S., "Grammatical Category Disambiguation by Statistical Optimization," Computational Linguistics, Vol. 14, No. 1, 1988.

Firth, J., (1957), "A Synopsis of Linguistic Theory 1930-1955" in *Studies in Linguistic Analysis*, Philological Society, Oxford; reprinted in Palmer, F., (ed. 1968), *Selected Papers of J.R. Firth*, Longman, Harlow.

Francis, W., and Kucera, H., (1982), *Frequency Analysis of English Usage*, Houghton Mifflin Company, Boston.

Good, I. J., (1953), *The Population Frequencies of Species and the Estimation of Population Parameters*, Biometrika, Vol. 40, pp. 237-264.

Hanks, P., (1987), "Definitions and Explanations," in Sinclair (1987a).

Harris, Z., (1968), "Mathematical Structures of Language," New York: Wiley.

Hirschman, L., Grishman, R., and Sager, N., (1975) "Grammatically-based automatic word class formation," *Information Processing and Management*, 11, 39-57.

Hindle, D., (1983), "User manual for Fidditch, a deterministic parser," Naval Research Laboratory Technical Memorandum #7590-142

Kahan, S., Pavlidis, T., and Baird, H., (1987) "On the Recognition of Printed Characters of any Font or Size," IEEE Transactions PAMI, pp. 274-287.

Sinclair, J., Hanks, P., Fox, G., Moon, R., Stock, P. (eds), (1987a), *Collins Cobuild English Language Dictionary*, Collins, London and Glasgow.

Sinclair, J., (1987b), "The Nature of the Evidence," in Sinclair, J. (ed.), *Looking Up: an account of the COBUILD Project in lexical computing*, Collins, London and Glasgow.

Tomita, M., (1986), *Efficient Parsing for Natural Language*, Kluwer Academic Press.

9. Conclusion

In any natural language there are restrictions on what words can appear together in the same construction, and in particular, on what can be arguments of what predicates. It is common practice in linguistics to classify words not only on the basis of their meanings but also on the basis of their co-occurrence with other words. Running through the whole Firthian tradition, for example, is the theme that "You shall know a word by the company it keeps" (Firth, 1957).

> "On the one hand, *bank* co-occurs with words and expressions such as *money, notes, loan, account, investment, clerk, official, manager, robbery, vaults, working in a, its actions, First National, of England*, and so forth. On the other hand, we find *bank* co-occurring with *river, swim, boat, east* (and of course *West* and *South*, which have acquired special meanings of their own), *on top of the*, and *of the Rhine*." (Hanks 1987, p. 127)

Harris (1968) makes this "distributional hypothesis" central to his linguistic theory. His claim is that: "the meaning of entities, and the meaning of grammatical relations among them, is related to the restriction of combinations of these entities relative to other entities," (Harris 1968:12). Granting that there must be some relationship between distribution and meaning, the exact nature of such a relationship to our received notions of meaning is nevertheless not without its complications. For example, there are some purely collocational restrictions in English that seem to enforce no semantic distinction. Thus, one can *roast chicken* and *peanuts* in an oven, but typically *fish* and *beans* are *baked* rather than *roasted*: this fact seems to be a quirk of the history of English. Polysemy provides a second kind of complication. A *sentence* can be *parsed* and a *sentence* can be *commuted*, but these are two distinct senses of the word *sentence*; we should not be misled into positing a class of things that can be both *parsed* and *commuted*.

Given these complicating factors, it is by no means obvious that the distribution of words will directly provide a useful semantic classification, at least in the absence of considerable human intervention. The work that has been done based on Harris' distributional hypothesis (most notably, the work of the associates of the Linguistic String Project (see for example, Hirschman, Grishman, and Sager 1975)) unfortunately does not provide a direct answer, since the corpora used have been small (tens of thousands of words rather than millions) and the analysis has typically involved considerable intervention by the researchers. However, with much larger corpora (10-100 million words) and robust parsers and taggers, the early results reported here and elsewhere appear extremely promising.

References

Bates, M., "Syntactic Analysis in a Speech Understanding System," BBN Report No. 3116, 1975.

Chodorow, M, Byrd, R., and Heidorn, G., (1985) "Extracting semantic hierarchies from a large on-line dictionary," ACL Proceedings.

Church, K., (1988), "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," Second Conference on Applied Natural Language Processing, Austin, Texas.

Such a system can serve as the basis of a practical, linguistically-biased system using LFG and is a prerequisite for an effective set of grammar writing and debugging tools which operate at the level of LFG itself. The parsing algorithm described here is intended for use in an automatic speech understanding (ASR) system project (currently funded by the Royal Signals and Radar Establishment), where a data driven strategy coupled with a strong TD predictive capability is highly desirable.

An Efficient, Primarily Bottom-Up Parser for Unification Grammars

Neil K. Simpkins and Peter J. Hancox
Applied Mathematics
Aston University
Aston Triangle
Birmingham, B4 7ET

Abstract

The search for efficient parsing strategies has a long history, dating back to at least the Cocke/Younger/Kusami parser of the early sixties. The publication of the Earley parser in 1970 has had a significant influence on context-free (CF) parsing for natural language processing, evidenced by the interest in the variety of chart parsers implemented since then. The development of unification grammars (with their complex feature structures) has put new life into the discussion of efficient parsing strategies, and there has been some debate on the use of essentially bottom-up or top-down strategies, the efficacy of top-down filtering and so on.

The approacn to parsing described here is suitable for complex category, unification-based grammars. The concentration here is on a unification grammar which has a context-free backbone, Lexical-Functional Grammer (LFG). The parser is designed primarily for simplicity, efficiency and practical application.

The parser outlined here results in a high-level, but still efficient, language system without making a requirement on the grammar/lexicon writer to understand its implementation details. The parsing algorithm operates in a systematic bottom-up (BU) fashion, thus taking earliest advantage of LFG's concentration of information in the lexicon and also making use of unrestricted feature structures to realize LFG's Top-Down (TD) predictive potential. While LFG can make special use of its CF backbone, the algorithm employed is not restricted to grammars having a CF backbone and is equally suited to complex-feature-based formalisms.

Additionally, the algorithm described (which is a systematic left-to-right (left corner) parsing algorithm) allows us to take full advantage of both BU and TD aspects of a unificatin-based grammar without incurring prohibitive overheads such as feature-structure comparison or subsumption checking. The use of TD prediction, which in the Earley algorithm is allowed to hypothesize new parse paths, is here restricted to confirming initial parses produced BU, and specializing these according to future (feature) expectations.

Score: see section 3.1, *Computing Preferences*, below.

Lexical Frame: see section 2.1, *Text Lexicon*, below.

Predictions and Requirements: grammatical predictions and constraints, both as found in the language object itself and as synthesized from subordinated language objects (see section 2.1, *Text Lexicon*, below).

Preceding and Remaining Text: the sentential context, or local sentence buffers; (preceding words are needed to check on grammatical predictions and requirements that can be either forward or backward in the sentence). When the *remaining text* is exhausted, the language object is a candidate sentence representation.

Subordinated Language Objects: subordination refers to the way an NP is subordinated to a PP, an AP is subordinated to an NP, and a phrase element is subordinated to the phrase head. A corresponding label list holds labels associated with each stack element.

name:	technician	(string data type)	filler: a word from the text
type:	word	(string data type)	filler: WORD or phrase: NP, AP, PP, VP, or CL (clause)
score: .	1.75	(short-float data type)	filler: preference/priority value
frame:	(POS . noun) (SENSE . 0) (GRAMMAR . noun/count) (SEM-TYPE . human) (PRAG . occupations)	(association list)	filler: a lexical semantic frame instantiated to some particular word sense of the object named
requirement:	nil	(list of atoms)	filler: codes for grammatical requirements from GRAMMAR slots
prediction:	nil	(list of atoms)	filler: codes for grammatical predictions from GRAMMAR slots
previous:	(The)	(list of words)	filler: previous words in the sentence
remainder:	(measures alternating current with an ammeter)	(list of words)	filler: subsequent words in the sentence
cases:	nil	(list of labels)	filler: case and function labels marking constituent relations
subordinate:	nil	(a LIFO stack of language objects)	filler: language objects that form the state of the parse and are linguistically subordinate to the current language object.

Fig. 1: Language object for "technician" (before coalescing).

1.2. Preference Machine Control

PREMO receives two inputs: a text to be parsed, and a lexicon of semantic objects specific to that text. The algorithm of the preference machine, after loading the priority queue with scored language objects for every sense of the first word in the sentence, is as follows (Fig. 2):

1. **Delete-Max** - retrieve the highest priority language object in the queue. (If the sentence buffer for that object is empty then go to 8).
2. **Get-Lexical** - retrieve the list of sense frames associated with the first word in the sentence buffer within the current language object
3. **Make-Language-Object(s)** - instantiate a new language object for every sense of the new word, with an appropriate initial preference score.
4. **Copy-Language-Object** - create a copy of the current high priority language object to pair with each new language object.
5. **Coalesce** - If more than a single grammar rule applies to the pair, copies of the pair are made. Combine the pairs of language objects, subordinating one to the other.

PREMO: parsing by conspicuous lexical consumption†

Brian M. Slator and Yorick Wilks*

Computing Research Laboratory
Box 30001
New Mexico State University
Las Cruces, NM 88003-0001

ABSTRACT

PREMO is a knowledge-based Preference Semantics parser with access to a large, lexical semantic knowledge base and organized along the lines of an operating system. The state of every partial parse is captured in a structure called a *language object*, and the control structure of the preference machine is a priority queue of these language objects. The language object at the front of the queue has the highest score as computed by a preference metric that weighs grammatical predictions, semantic type matching, and pragmatic coherence. The highest priority language object is the intermediate reading that is currently most preferred (the others are still "alive," but not actively pursued); in this way the preference machine avoids combinatorial explosion by following a "best-first" strategy for parsing. The system has clear extensions into parallel processing.

1. Introduction

PREMO: The **PRE**ference Machine Organization, is an architecture, modelled as an operating system, for parsing natural language. Each "ready" process in the system captures the state of a partial parse in a "process control block" structure called a *language object*. The control structure is a priority queue of competing parses, with priority given to each parse "process" on the basis of a preference semantics evaluation. The "time-slice" for each process is whatever is needed to move forward one word in a local process sentence buffer (where each process operates on a private copy of the current sentence). After every time slice, the preference/priority for the currently "running" parse is re-computed and the language object for that process is returned to the priority queue. The first process to emerge from the queue with its sentence buffer empty is declared the winner and saved. This strategy is both a run-time optimization and an application of the "Least Effort Principle" of intuitively plausible language processing. The parsing is robust in that some structure is returned for every input, no matter how ill-formed or "garden-pathological" it is.

1.1. Language Object Structures

The basic data structure manipulated by the preference machine is called a *language object* (Fig. 1). Each language object is a complex structure containing at least the following attributes:

Name and Type: every object *name* is a word from the text. Object *type* defaults to *word* until a word is found to be part of phrase, then that object is changed to type *phrase*. The *head* of a phrase is the language object with its co-members subordinated to it.

† This research was supported by the New Mexico State University Computing Research Laboratory — grateful acknowledgement is accorded to the members of the CRL Natural Language Group for their continuing interest and support.

* Present address: Department of Computer Science, North Dakota State University, Fargo, ND 58105.

2.2. Syntactic Structures

The PREMO grammatical formalism is non-standard, being not a phrase-structured production system of rewrite rules but rather a phrase-triggered system of situation-action rules. The Coalesce procedure lies at the heart of the PREMO algorithm. This routine accepts a pair of language objects:

1. the current high priority language object as retrieved (and perhaps copied) from the priority queue, and
2. a new language object representing a single word sense of the next word in the sentence buffer of the current high priority language object.

Every language object that is retrieved from the priority queue is of type *phrase*, and every new language object created from the sentence buffer of the currently running parse is of type *word*. The rules of the phrase grammar have a triple of symbols on the left hand side representing: (1) the phrase type of the current high priority language object; (2) the phrase type of the language object on the top of the stack of the current high priority language object; and (3) the syntactic category of the newly created language object. There are one or more triples of symbols on the right hand side of each grammar rule specifying: (1) the phrase type of the new language object; (2) the action to be performed on the new language object; and, (3) the location where the action is to take place (Fig. 3).

```
(phrase-type-A phrase-type-B category-C)   =>   (phrase-type-D_1 operation_1 location_1)
                                           =>   (phrase-type-D_2 operation_2 location_2)
                                           =>   . . .
                                           =>   (phrase-type-D_n operation_n location_n)
```

Fig. 3: The gloss for a generic grammar rule: if given a language object of phrase-type-A, whose top-of-stack language object is of phrase-type-B, and confronting a new language object of grammatical category-C, then for each i from 1 to n, make a copy of the pair, change the new language object copy into phrase-type-D_i, and perform operation$_i$ (either Sub, Push, or Subto), at location$_i$ (within the New, Old, or Old-Sub language object).

The set of possible phrase types is limited, at present, to these five: Adjective phrase, Noun phrase, Prepositional phrase, Verb phrase, and Clause (a generic Other phrase type). Although several action triples (all of which get executed), could appear on the right hand side of a rule, in the current implementation no rule exceeds five action triples and most are three or less. Further, the part of speech set in the lexicon derived from LDOCE has 10 members: adjective, adverb, conjunction, determiner, interjection, noun, predeterminer, preposition, pronoun, and verb. These three facts conspire to give a limiting factor to the total size of the grammar rule set.

This grammar is a phrase (or constituent) grammar and not a sentence grammar. The parser posits a sentence analysis only when its sentence buffer is consumed; until that point phrases are constructed and coalesced with each other as they are encountered, without regard to sentence level structure. The **Coalesce** decision is a syntactic one, with the resulting superordinate language object effectively assuming the status of the *head* of the entire existing structure. Either the new language object is inserted somewhere within the highest priority language object as a subordinate, or the highest priority object is subordinated to the new object (Fig. 4). In the second case the new object is effectively elevated to the status of superordinate, and it is this coalesced language object that is inserted into the priority queue (with a suitably computed preference score).

6. **Compute-Preference** - assign a new priority score to each of the language objects resulting from coalescing pairs.

7. **Enqueue** - insert the coalesced language objects onto the priority queue in preference score order. Go to 1.

8. **Inter-sentential Processes** - save the language object at the front of the priority queue and flush the queue. If there are no more sentences, return; otherwise, read the next sentence in the text and load the priority queue with scored language objects for every sense of the first word in the new sentence. Go to 1.

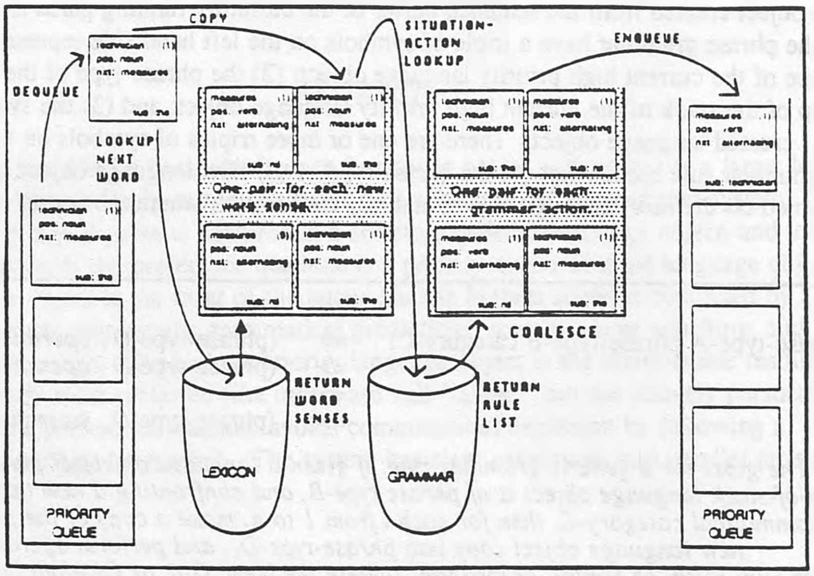

COPY
SITUATION-ACTION LOOKUP
ENQUEUE
DEQUEUE
LOOKUP NEXT WORD
technician (1) pos: noun nxt: measures sub: the
measures (1) pos: verb nxt: alternating sub: nil
measures (0) pos: noun nxt: alternating sub: nil
One pair for each new word sense.
One pair for each grammar action.
measures (1) pos: verb nxt: alternating sub: technician
COALESCE
RETURN WORD SENSES
RETURN RULE LIST
LEXICON
GRAMMAR
PRIORITY QUEUE
PRIORITY QUEUE

Fig. 2: **PREMO:** ***the PREference Machine Organization.***

2. Global Data

Global system data structures include a *text-specific lexicon*, and a *context structure* derived from Longman's Dictionary of Contemporary English (LDOCE; Procter et al. 1978), and a *phrase grammar*.

2.1. Text Lexicon

LDOCE is a full-sized dictionary in machine-readable form, designed for learners of English as a second language, and containing several non-standard features (grammar, type, and pragmatic codes). A PREMO sub-system produces text-specific lexicons from selected machine-readable dictionary definitions (Wilks, Fass, Guo, McDonald, Plate and Slator, 1987, 1988, 1989). The input to this sub-system is unconstrained text; the output is a collection of lexical semantic objects, one for every *sense* of every word in the text. Each lexical semantic object in this lexicon contains grammatical and sub-categorization information, often with general (and sometimes specific) grammatical predictions; content word objects also have semantic selection codes; and many have contextual (pragmatic) knowledge as well. As a natural side-effect of the lexicon construction, a relative contextual score is computed for each object that bears such a code; these scores provide a simple metric for comparing competing word senses for text-specific contextual coherence, and so directly address the problem of lexical ambiguity. Besides exploiting those special encodings supplied with the dictionary entries, the text of selected dictionary definitions are analyzed, through parsing and pattern matching, to further enrich the resulting representation (Slator, 1988a, 1988b; Slator and Wilks 1987, 1989).

3.1. Computing Preferences

When a new language object is first created it receives a preliminary preference score. An initial value is given between 1.0 and 2.0 that depends on the word's sense number (the lower the word sense, the higher, closer to 2.0, the score). Immediately thereafter, various attributes of the language object are evaluated and the initial score is adjusted. Adjustments to scores are either "minor," "standard," or "major" (in the current implementation these are 2%, 10%, and 50% respectively), and can be in either direction. In the current implementation preliminary scores are *decreased* in the case of an "interjection" part of speech, or for an LDOCE time-and-frequency code of "archaic" or "rare." Preliminary scores are *increased* if the language object is for a phrasal definition (such as "alternating current"), or is for a closed class word, or if it makes a grammatical prediction, or if it is a word with only a single sense definition. Finally, scores are strongly influenced, in either direction, by the position of the word with respect to a restructured pragmatic hierarchy computed for the text. For example, if the text has a scientific orientation then the scientific senses of words are given preferential increases and the other senses of those words are given decreased scores (such as the scientific senses, as opposed to the political and musical senses, of "measure").

After the Coalesce decision has been made, and one language object has been subordinated to the other, a new score is assigned to the result. These scores are computed according to the following criteria:

Predictions and Requirements: The lexical semantic frames have GRAMMAR slots that contain predictions and requirements: some general and some specific. Some general codes mark nouns as "a countable noun followed by the infinitive with *to*," and others mark verbs as "ditransitive and followed by a *that* clause."[1] There are also specific predictions: particular senses of "sat" and "lay" predict an adverb or preposition, and particularly "down." And there are some absolute requirements: one sense of "earth" requires the article "the." These are collected and checked as language objects are being coalesced, and subordinate predictions are "synthesized" into their superordinate language object. Naturally, when a prediction is fulfilled the preference/priority is increased; and when a prediction is made but not fulfilled, scores are decreased. The degree to which scores are changed is still being experimented with, the currently implemented heuristic is to effect larger changes for more specific predictions.

Subordination: When language objects are coalesced the current implementation awards minor increases to pairings that follow a notion of natural order; for example, a Verb phrase subordinating a Noun phrase is a natural event to expect, but an Adjective phrase subordinating a Verb phrase less so. Both must be permitted, since it is possible to construct an example of either.

Semantic Matching: Content words in the text lexicon have semantic codes placing them in the LDOCE type hierarchy (types like **abstract, concrete,** or **animate**). Nouns and adjectives have a single code identifying their place in the hierarchy; verbs have 1, 2, or 3 codes identifying selection restrictions on their arguments. Semantic matching is done, and scores are adjusted for semantic coherence, whenever a pair of language objects are coalesced such that (1) an adjective (or a nominal) modifies a noun phrase head, or (2) a noun phrase is being attached as an argument to a verb, or (3) a noun phrase is being attached as the object of a preposition, or (4) a prepositional phrase is being attached as an argument to a verb. In the current implementation increases (but not decreases) are computed as a function of distance in the type hierarchy. If a head prefers, say, an **animate** argument and is presented with an **abstract** word sense, the increase will be quite small as opposed to being presented with a competing **human** word sense.

3.2. Semantic Structures

When the Coalesce decision is being made, PREMO looks to see if one language object is being subordinated to the other with a "push" operation. If so, a new constituent is being started and it is appropriate to affix a semantic label onto the subordinate object, since it is about to

[1] Such as *attempt* in "an attempt to climb the mountain," and *warn* in "He warned her (that) he would come."

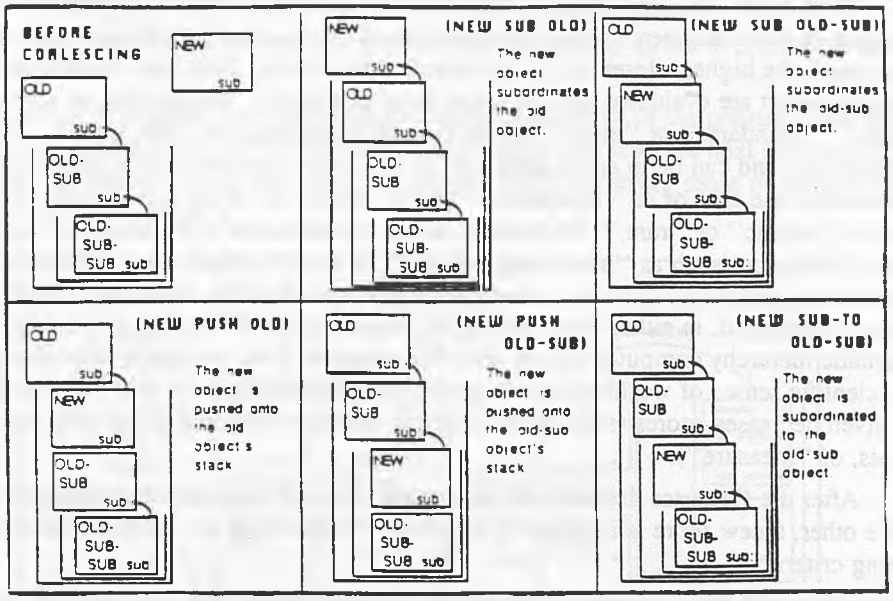

BEFORE COALESCING
NEW
sub
OLD
sub
OLD-SUB
sub
OLD-SUB-SUB sub
(NEW SUB OLD)
The new object subordinates the old object.
(NEW SUB OLD-SUB)
The new object subordinates the old-sub object.
(NEW PUSH OLD)
The new object is pushed onto the old object's stack
(NEW PUSH OLD-SUB)
The new object is pushed onto the old-sub object's stack
(NEW SUB-TO OLD-SUB)
The new object is subordinated to the old-sub object

Fig. 4: PREMO *Coalesce Operations*

At any given point in a parse, there will always be a language object construed as the *head* of the parse, whichever language object has the superordinate status, of the pair being coalesced, will become the head as per the rules of the grammar (where status is generally a reflection of the usual syntactic dominance, with PP's dominating NP's, and VP's dominating everything).

3. Preference Semantics

PREMO is a knowledge-based Preference Semantics parser (Wilks 1972, 1975a, 1975b, 1978), with access to the large, lexical semantic knowledge base created by the PREMO lexicon-provider subsystem. Preference Semantics is a theory of language in which the meaning for a text is represented by a complex semantic structure that is built up out of smaller semantic components; this compositionality is a fairly typical feature of semantic theories. The principal difference between Preference Semantics and other semantic theories is in the explicit and computational accounting of ambiguous, metaphorical, and non-standard language use.

The links between the components of the semantic structures are created on the basis of semantic preference and coherence. In text and discourse theory, coherence is generally taken to refer to the meaningfulness of text. Fass (1987) suggests that in NLP work such as Preference Semantics the notions of "satisfaction" and "violation" (of selection restrictions or preferences) and the notion of "semantic distance" (across structured type hierarchies) are different ways of characterising the meaningfulness of text; they capture different coherence relations. The original systems of Preference Semantics (Wilks 1972, 1975a, 1975b, 1978), were principally based on the coherence relation of "inclusion" (semantic preferences and selection restrictions); the emphasis in PREMO is more on the coherence relation based on semantic distance, although the original notions of coherence also survive.

In Preference Semantics the semantic representation computed for a text is the one having the most semantically *dense* structure among the competing "readings." Semantic density is a property of structures that have preferences regarding their own constituents, and satisfied preferences create density. Density is compared in terms of the existence of preference-matching features, the lack of preference-breaking features, and the length of the inference chains needed to justify each sense selection and constituent attachment decision. The job of a Preference Semantics parser, then, is to consider the various competing interpretations, of which there may be many, and to choose among them by finding the one that is the most semantically dense, and hence preferred.

code carried by *technician*.

Iteration 4: The language object [VP:*measures*] with its subordinated language object [NP:*technician,The*] on the top of its stack, is popped from the queue and the 4 senses of *alternate*, along with the language object for the phrasal *alternating current*, are instantiated; 5 copies of [VP:*measures*] are then created and paired with them. Phrasal objects are preferred by PREMO, and nothing occurs to outweigh that preference. The question to be decided is which of these two readings should be preferred:

4. (VP NP noun) => ((NP push old) <or> (NP sub old-sub))

that is, does *alternating current* represent the start of a new NP constituent, or is it the new head of the ongoing top-of-stack NP constituent (in this case [NP:*technician,The*]). The semantic code carried by *alternating current* is **abstract-physical-quality** which is a poor match with the **human** of [NP:*technician,The*] but a good match with the second argument code of [VP:*measure*], which is **abstract.** Therefore the new NP constituent reading receives the better preference/priority score and assumes the position at the head of the PQ. However, first a label must be attached to the NP constituent that is about to disappear from the top-of-stack as a result of the "push" operation. In this case, since the verb prefers and receives a **human** subject, this NP is labelled "Agentive."

Iterations 5-11: The high priority language object on the front of the PQ is [VP:*measures*] which now subordinates both [NP:*technician,The*] and [NP:*alternating current*]. The continuation of the sentence buffer in [VP:*measures*] is now *with an ammeter*. The next several iterations are concerned with pushing a PP onto the subordinate stack of [VP:*measures*] and then subordinating [NP:*ammeter,an*] as the object of the PP. The current implementation recognizes 3 senses of the preposition *with* representing the ACCOMPANIMENT, POSSESSION, and INSTRUMENT cases. Each of these is coalesced with [NP:*ammeter,an*] and pushed onto the subordinate stack of [VP:*measures*]. The INSTRUMENT reading is preferred on the basis of semantic matching between the selection restriction code on the object of the preposition, which is **concrete,** and the semantic code for *ammeter*, which is **movable-solid.**

Iteration 12: The final iteration retrieves the language object for the [VP:*measures*] from the front of the PQ and finds the sentence buffer is empty. It is at this point that the case label marking the top-of-stack element (which is [PP:*with*,[NP:*ammeter,an*]]), as the INSTRUMENT case is actually affixed. This language object is then saved as the interpretation of sentence (1), the queue is flushed and PREMO reads whatever sentence text follows, or if none follows, PREMO ends.

4.1. Toward Solving a Hard Problem

Two contrasting methods of word sense selection have been described here. The earlier method was first explored by the Cambridge Language Research Unit, beginning in the mid-1950s. This method performed a global analysis, (Masterman 1957, as described in Wilks, 1972), that relied on a thesaural resource. This method was good at choosing word senses coherent with the other words in the text, by using a system of looking to the sets of words found under common thesaural heads, and performing set intersections. The problem is that the less "coherent" word senses are missed and so, for example, in a physics text only the scientific sense of *mass* will chosen and, therefore, the word *mass* in the phrase *mass of data* will come out wrong in that text.

The other method is Preference Semantics that performs a local analysis that relies on semantic type markers. This method is good at choosing word senses that best fit with other words in a sentence, by using a system of matching and comparing among the various primitive elements that make up the meanings of words. The problem is that this can be fooled into preferring word senses that only seem to be best. The standard example, attributed to Phil Hayes, is the following.

(2) *A hunter licked his gun all over and the stock tasted good.*

In this example, the challenge is to choose the correct sense of *stock*. The problem is that the local evidence supplied by the verb *to taste* points towards the "stock as soup" reading, which is wrong. Granted, this is something of a pathological example, but it is famous and it captures the flavor of the objection.

disappear under the new top-of-stack. These labels identify the functional or semantic relations that hold between a language object and its superordinate. The LDOCE hierarchies, in conjunction with the lexical semantic frame contained within each language object, and the "frame enriching" procedures developed for the lexicon are brought to bear at this point (Slator and Wilks, 1987, 1989); as well as the hand-coded definitions for prepositions that we must admit to creating since LDOCE, from our point of view, does not include useful case information in their preposition definitions.

Every sentence in a text is eventually represented by a single language object. These language objects are named for the word seen to be the head of the dominating phrase in the sentence, and are of type *phrase* (and presumably of phrase type VP, in the usual grammatical case). Each of the subordinated phrases in the sentence is stacked within this superordinate language object, along with a corresponding relation label.

4. PREMO Example

Consider the following sentence:

(1) *The technician measures alternating current with an ammeter.*

First PREMO loads the lexicon specific to this text, which contains 26 frames for content words. These 26 frames are: *alternate* (3 adjective senses, 1 verb sense), *ammeter* (1 noun sense), *current* (3 adjectives, 4 nouns), *measure* (8 nouns, 3 verbs, 1 adjective), *technician* (1 noun sense), and the phrase "alternating current" (1 noun sense). LDOCE defines about 7,000 phrases. PREMO performs a contextual analysis by appeal to the pragmatic codes as organized into a specially restructured hierarchy. This results in the various Science and Engineering word senses receiving increased preference/priority scores while most other word senses receive decreased scores. This context setting mechanism is discussed at length in Slator (1988a, 1988b), Slator and Wilks (1987, 1989) and Fowler and Slator (1989).

Then, PREMO initializes the priority queue (PQ) with language objects for both the adverbial and definite determiner senses of *The* (the first word in the sentence), at which point the loop of the algorithm in section 1.2, *Preference Machine Control* is entered. In the first iteration the determiner is instantiated as an Adjective phrase [AP:*The*], as is the adverb reading, according to the rules of the grammar. The analysis of sentence (1) requires a total of 12 iterations through this loop. Notice that sentence (1) is 336-way ambiguous if just the number of senses of polysemous content words are multiplied out (*alternate*=4, times *current*=7, times *measure*=12, equals 336), and that that number grows to 1008 if the three cases of *with* are included.

Iteration 2: The PQ contains three language objects for *The*, of which the definite determiner is slightly preferred. This object is popped from the queue, and the next word in its sentence buffer, *technician* is retrieved from the lexicon and instantiated as a language object. There is only a single sense of *technician* and only a single grammar rule action for the situation:

1. (AP nil noun) => (NP sub old).

This means *technician* becomes an NP with the determiner *The* subordinated to it [NP:*technician,The*]. This act of coalescing results in a minor increase being assigned to [NP:*technician,The*], and it returns to the priority queue.

Iteration 3: The language object [NP:*technician,The*] is popped from the queue and the 11 senses of the next word in its sentence buffer, *measure*, are instantiated; then 11 copies of [NP:*technician,The*] are created and paired with them. The two major grammar competitors during this iteration are:

2. (NP AP noun) => (NP sub old)
3. (NP AP verb) => (VP sub old)

that is, *measures* as a noun becoming the new head of [NP:*technician,The*], as opposed to *measures* as the head of a verb phrase taking [NP:*technician,The*] as an argument.

The criteria for comparing these readings (and the fact that PREMO prefers the second, VP analysis), reduces to the fact that the noun *measures* carries an **abstract** semantic code which does not match well with the **human** semantic code carried by *technician;* while the verb *measures* carries a **human** selection restriction for its first argument, which matches exactly with the semantic

PREMO analysis for text (4) *Current can be measured.*

```
((measured VP 4.757666S0 "v" "0300" (INFL SUBJECT))
 (((be VP 4.077573S0 "v" "0008" (INFL))
   ((can VP 3.028026S0 "v" "0100" nil)))
  (Current NP 2.763158S0 "n" "0100" nil)))
```

PREMO analysis for *The geographer measures river basin flow near a lake.*

```
((measures VP 4.354593S0 "v" "0100" (LOCATIVE OBJECT1 AGENTIVE))
 (((near PP 2.406232S0 "prep" "0000" (OBJECT1))
   (((lake NP 2.719298S0 "n" "0000" (DET))
     ((a AP 0.9075S0 "indefinite" "0100" nil)))))
  ((flow NP 1.433986S0 "n" "0500" (KIND-OF))
   (((river*basin NP 2.178159S0 "n" "0000" (DET))
     ((the AP 0.9982499S0 "definite" "0100" nil)))))
  ((geographer NP 1.693873S0 "n" "0000" (DET))
   ((The AP 0.9982499S0 "definite" "0100" nil)))))
```

Fig. 5: PREMO Analyses for Texts (3) and (4).

The other two language objects displayed both show the first sense of *measure* each with three language objects on the subordinate stack. In each case these subordinate language object constituents represent the AGENT (*technician* and *geographer*), and the OBJECT (*alternating current* and *river basin flow*), of the measuring action. Text (3) also has a prepositional phrase attached in the INSTRUMENT case while text (4) has a prepositional phrase attached in the LOCATIVE case.

In spite of this success, the problem of mediating the tension between global and local sources of information is still not completely solved. PREMO assumes text coherence and so while Preference Semantics provides a naturally local sort of analysis procedure these local effects can be overcome by appeal to global context. However, it is still possible to find examples, such as text (2) above, that do not *have* any context (a common situation in the computational linguistics literature, where space constraints and custom preclude long examples). And in the absence of this global information PREMO will not perform any better than any other system of analysis. That is, if text (2) were embedded in a longer exposition about hunters and guns, then the probability is high that the correct sense of *stock* would be chosen. If however, text (2) were embedded in an exposition about food and cooking, PREMO would almost certainly get this wrong. And in the absence of context PREMO will choose the ''soup stock'' reading because the notion of gun stocks having a taste is not one that finds much support in a system of semantic analysis.

5. Comparison to Other Work

The original Preference Semantics implementations (Wilks 1972, 1975a, 1975b, 1978), operated over a hand coded lexicon of semantic formulae. Input strings were segmented into phrases beforehand, and coherence was essentially a matter of counting ''semantic ties'' inferred by pattern matching between formulae. PREMO operates over a machine-readable lexicon that is at once much broader and shallower than the original. To make up for this, preference scoring in PREMO is much more finely grained and takes more into account (grammatical predictions, pragmatic context, etc.). If anything, PREMO is grammatically weaker than the original work, while being more robust in the sense that syntactic anomalies and ill-formed input are processed the same as anything else.

A group at Martin Marietta (Johnson, Kim, Sekine, and White, 1988; White 1988), built a language understander based on Preference Semantics, but modified by their own interpretation of Wilks, Huang, and Fass (1985). Their NLI system is frame-based and much of the system's knowledge resides in the lexicon, which is constructed by hand. The parsing process is separated

PREMO attempts to tackle these contrasting analysis problems by bringing together both global and local information. To demonstrate this, consider the analysis of the following two short texts. The first is familiar from the example in Section 4, above.

(3) *Current can be measured.*
The technician measures alternating current with an ammeter.

The following text is intended to parallel the first one.

(4) *Current can be measured.*
The geographer measures river basin flow near a lake.

The point at issue is choosing the correct sense of *current* in each case. In text (3) it is the **engineering/electrical** sense that should be chosen. In text (4), however, it is the **geology-and-geography** sense of *current* that is correct. In the absence of other evidence, the **geology-and-geography** sense is the one most systems would choose, since this is the more common usage of the word in the language (and the lowest numbered word sense in LDOCE, which reflects this notion of default preference). And since most systems have no notion of global text coherence, most would get the wrong sense of *current* in text (3) at first reading. It is conceivable that the sense selection for *current* could be corrected after further text has been processed, but few if any systems attempt this, and it is far from obvious how this should be done in general. PREMO gets both of these texts right, by choosing the correct sense of *current* in each case, and making all of the other word sense and attachment decisions (see fig. 5).

In Fig. 5, each line element represents a condensation of a language object. Language objects are complex items which this diplay merely summarizes. The form of these displays is as follows:

(<name> <phrase-type> <score> <part-of-speech> <sense-number> <stack-labels>)

and the lower language objects are on stacks, as indicated by their indentations. And so, the first language object reads as this: the third sense of the verb *measure* (the linking verb sense), has two elements on its subordinate stack, one marked as a verb inflection (the language object for *be*, which itself has a language object on its internal stack marking verb inflection, the language object for *can*), and the other stack element (note, the second sense of *current*), marked as the subject of the measuring. The third language object in Fig. 5 has the identical interpretation, except that the subject of the measuring is the first sense of *current* rather than the second.

PREMO analysis for text (3) *Current can be measured.*

```
((measured VP 4.855569S0 "v" "0300" (INFL SUBJECT))
 (((be VP 4.197S0 "v" "0008" (INFL))
   ((can VP 3.174081S0 "v" "0100" nil)))
  (Current NP 2.985294S0 "n" "0200" nil)))
```

PREMO analysis for *The technician measures alternating current with an ammeter.*

```
((measures VP 4.68685S0 "v" "0100" (INSTRUMENT OBJECT1 AGENTIVE))
 (((with PP 3.726117S0 "prep" "0000" (OBJECT1))
   (((ammeter NP 2.814706S0 "n" "0000" (DET))
     ((an AP 1.815S0 "indef" "0000" nil)))))
  (alternating*current NP 3.838235S0 "n" "0000" nil)
  ((technician NP 0.8368717S0 "n" "0000" (DET))
   ((The AP 0.9982499S0 "definite" "0100" nil)))))
```

References

BOGURAEV, BRANIMIR K. (1979). Automatic Resolution of Linguistic Ambiguities. *University of Cambridge Computer Laboratory Technical Report*. (No.11). Cambridge, UK: University of Cambridge Computer Laboratory.

CARTER, DAVID M. (1984). An Approach to General Machine Translation Based on Preference Semantics and Local Focussing. *Proceedings of the 6th European Conference on AI (ECAI-84)*, pp. 231-238. Pisa, Italy.

CARTER, DAVID M. (1987). *Interpreting Anaphors in Natural Language Texts*. Chichester, UK: Ellis Horwood.

FASS, DAN C. (1986). Collative Semantics: An Approach to Coherence. *Computing Research Laboratory Memorandum*. (MCCS-86-56). Las Cruces, NM: New Mexico State University.

FASS, DAN C. (1987). Semantic Relations, Metonymy, and Lexical Ambiguity Resolution : A Coherence-Based Account. *Proceedings of the 9th Annual Cognitive Science Society Conference*, pp. 575-586. Seattle, WA: University of Washington.

FASS, DAN C. (1988). An Account of Coherence, Semantic Relations, Metonymy, and Lexical Ambiguity Resolution. In *Lexical Ambiguity Resolution in the Comprehension of Human Language*. Edited by Steve L. Small, Gary W. Cottrell, and Michael K. Tanenhaus. pp. 151-178. Los Altos, CA: Morgan Kaufmann.

FOWLER, RICHARD H. AND BRIAN M. SLATOR (1989). Information Retrieval and Natural Language Analysis. *Proceedings of the 4th Annual Rocky Mountain Conference on Artificial Intelligence (RMCAI-89)*, pp. 129-136. Denver, CO. June 8-9

HUANG, XIUMING (1984). A Computational Treatment of Gapping, Right Node Raising and Reduced Conjunction. *Proceedings of the 10th International Conference on Computational Linguistics (COLING-84)*, pp. 243-246. Stanford, CA.

HUANG, XIUMING (1988). XTRA: The Design and Implementation of A Fully Automatic Machine Translation System. *Computing Research Laboratory Memorandum*. (MCCS-88-121). Las Cruces, NM: New Mexico State University.

JOHNSON, HEIDI, G. KIM, Y. SEKINE, AND JOHN S. WHITE (1988). Application of Natural Language Interface to a Machine Translation Problem. *Proceedings of the Second International Conference on Theoretical and Methodological Issues in Machine Translation*. Pittsburgh, PA. June.

MASTERMAN, M. (1957). The Thesaurus in Syntax and Semantics. *Mechanical Translation*, 4, 1-2.

PROCTER, PAUL ET AL. (1978). *Longman Dictionary of Contemporary English (LDOCE)*. Harlow, Essex, UK: Longman Group Limited.

SLATOR, BRIAN M. (1988a). Constructing Contextually Organized Lexical Semantic Knowledge-Bases. *Proceedings of the Third Annual Rocky Mountain Conference on Artificial Intelligence*, pp. 142-148. Denver, CO. June, 13-15.

SLATOR, BRIAN M. (1988b). Lexical Semantics and a Preference Semantics Analysis. *Memoranda in Computer and Cognitive Science*. (MCCS-88-143). Las Cruces, NM: Computing Research Laboratory, New Mexico State University. (Doctoral Dissertation).

SLATOR, BRIAN M. (1988c). PREMO: the PREference Machine Organization. *Proceedings of the Third Annual Rocky Mountain Conference on Artificial Intelligence*, pp. 258-265. Denver, CO. June, 13-15.

SLATOR, BRIAN M. AND YORICK A. WILKS (1987). Towards Semantic Structures from Dictionary Entries. *Proceedings of the Second Annual Rocky Mountain Conference on Artificial Intelligence*, pp. 85-96. Boulder, CO. Also as CRL Memo MCCS-87-96.

SLATOR, BRIAN M. AND YORICK A. WILKS (Forthcoming - 1989). Towards Semantic Structures from Dictionary Entries. In *Linguistic Approaches to Artificial Intelligence*. Edited by Andreas Kunz and Ulrich Schmitz. Frankfurt: Peter Lang Publishing House. Revision of RMCAI-87 and CRL-MCCS-87-96.

WHITE, JOHN S. (1988). Advantages of Modularity in Natural Language Interface. *Proceedings of the Third Annual Rocky Mountain Conference on Artificial Intelligence*, pp. 248-257. Denver, CO. June, 13-15.

WILKS, YORICK A. (1972). *Grammar, Meaning, and the Machine Analysis of Language*. London: Routledge and Kegan Paul.

WILKS, YORICK A. (1975a). An Intelligent Analyzer and Understander of English. *Communications of the ACM*, 18, 5, pp. 264-274. Reprinted in "Readings in Natural Language Processing," Edited by Barbara J. Grosz, Karen Sparck-Jones and Bonnie Lynn Webber, Los Altos: Morgan Kaufmann, 1986, pp. 193-203.

WILKS, YORICK A. (1975b). A Preferential Pattern-Seeking Semantics for Natural Language Inference. *Artificial Intelligence*, 6, pp. 53-74.

WILKS, YORICK A. (1978). Making Preferences More Active. *Artificial Intelligence*, 11, pp. 75-97.

WILKS, YORICK A., DAN C. FASS, CHENG-MING GUO, JAMES E. MCDONALD, TONY PLATE, AND BRIAN M. SLATOR (1987). A Tractable Machine Dictionary as a Resource for Computational Semantics. *Proceedings of the Workshop on Natural Language Technology Planning*, Sept. 20-23. Blue Mountain Lake, NY. Also as CRL Memo MCCS-87-105. To appear in "Computational Lexicography for Natural Language Processing." Branimir K. Boguraev and Ted Briscoe (Eds.). Harlow, Essex, UK: Longman. 1988

into three autonomous modules: BuildRep (a constituent parser), Validate (a sort of constituent filter), and Unify (which incrementally builds a semantic structure from validated constituents). The principle differences between their system and PREMO lies in their criteria for abandoning non-productive paths (their domain constraints allow them to prune on the basis of semantic implausibility), and in their lack of a high level control structure (it is possible in there system for every parse to be abandoned and nothing returned).

Preference Semantics has also been used for parsing by Boguraev (1979), Carter (1984, 1987), and Huang (1984, 1988). However, this work uses a conventional parsing strategy in which syntax drives the parsing process depth-first and Preference Semantics is used within a semantics component that provides semantic verification of syntactic constituents. PREMO has a more flexible, more breadth-first parsing strategy in which syntax, semantics, and pragmatics interact more freely. The Meta5 semantic analyzer of Fass (1986, 1987, 1988), based on the system of Collative Semantics, which extends Preference Semantics, operates over a rich hand-coded lexicon comprised of a network of "sense frames." The principal goal of this system is to identify and resolve metaphorical and metonymous relations and, with its rich semantic knowledge base, Meta5 is able to produce deep semantic analyses which are quite impressive, although constrained to a somewhat narrow range of examples.

6. Discussion

PREMO employs a uniform representation at the word, phrase, and sentence levels. Further, at every step in the process there is a dominating language object visible; that is, there is always a "well-formed partial parse" extant. This gives an appealing processing model (of a language understander that stands ready to accept the next word, whatever it may be), and a real-time flavor, where the next word is understood in the context of existing structure. PREMO intentionally exploits everything that LDOCE offers, particularly in the area of grammatical predictions, and also in terms of the TYPE hierarchy as given, and the PRAGMATIC hierarchy as restructured, as well as extracting semantic information from the text of definitions.

One of the PREMO design principles is "always return something" and that policy is guaranteed by keeping every possibility open, if unexplored (this is the PREMO approximation to back-tracking). Another design principle is to cut every conceivable corner by making "smart" preference evaluations. The potential remains however, for worst case performance, where the preference/priority scores work out so that every newly coalesced pair immediately gets shoved to the bottom of the priority queue. If this happens the algorithm reduces to a brute search of the entire problem space.

By exploiting the operating system metaphor for control, PREMO inherits some very attractive features. First, PREMO avoids combinatorial explosion by ordering the potential parse paths and only pursuing the one that seems the best. This is antithetical to the operating system principle of "fairness," a point where the metaphor is intentionally abandoned in favor of a scheme that has some faint traces of intuitive plausibility. The competition between parses, based as it is on the tension between the various preference/priority criteria is vaguely reminiscent of a "spreading activation" system where the various interpretations "fight it out" for prominence. The PREMO architecture is, of course, utterly different in implementation detail, and it is not at all obvious how it could be equivalently converted, or that this metaphor is even a fruitful one. Second, the operating system metaphor is an extendible one; that is, it is possible to conceive of PREMO actually being implemented on a dedicated machine. Further, since the multiplication factor at each cycle through the algorithm is small (in the 40-60 range for the near-worst case of 10-12 word senses times 4-5 applicable grammar rules), and since each of these pairings is independent, it is easy to imagine PREMO implemented on a parallel processor (like a Hypercube). Each of the pairs would be distributed out to the (cube) processing elements where the coalescing and preference/priority scoring would be done in parallel.

Wilks, Yorick A., Dan C. Fass, Cheng-Ming Guo, James E. Mcdonald, Tony Plate, and Brian M. Slator (1988). Machine Tractable Dictionaries as Tools and Resources for Natural Language Processing. *Proceedings of the 12th International Conference on Computational Linguistics (COLING-88).* Budapest, Hungary. Aug. 22-27.

Wilks, Yorick A., Dan C. Fass, Cheng-Ming Guo, James E. Mcdonald, Tony Plate, and Brian M. Slator (Forthcoming - 1989). Providing Machine Tractable Dictionary Tools. In *Theoretical and Computational Issues in Lexical Semantics.* Edited by James Pustejovsky. Cambridge, MA: MIT Press.

Wilks, Yorick A., Xiuming Huang, and Dan C. Fass (1985). Syntax, Preference, and Right Attachment. *Proceedings of IJCAI-85,* 2, pp. 779-784. Los Angeles, CA.

```
A -->  B  C                (1)
```

are used to combine horizontally adjacent regions. In addition, rules like

```
       B
A -->                      (2)
       C
```

can be used in the 2-dimensional context-free grammar to combine vertically adjacent regions.

A region can be represented with a non-terminal symbol and 4 positional parameters: x, y, X and Y, which determine the upper-left position and the lower-right position of the rectangle (assuming that the coordinate origin is the upper-left corner of the input text).

Horizontally adjacent regions, (B, x_B, y_B, X_B, Y_B) and (C, x_C, y_C, X_C, Y_C), can be combined only if

- $y_B = y_C$,
- $Y_B = Y_C$, and
- $X_B = x_C$.

The first two conditions say that B and C must have the same vertical position and the same height, and the last condition says that B and C are horizontally adjoining.

Similarly, vertically adjacent regions, B and C, can be combined only if

- $x_B = x_C$,
- $X_B = X_C$, and
- $Y_B = y_C$.

A new region, (A, x_B, y_B, X_C, Y_C), is then formed. Figure 1-1 shows examples of adjacent regions, and figure 1-2 shows the results of combining them using rules (2) and (1).

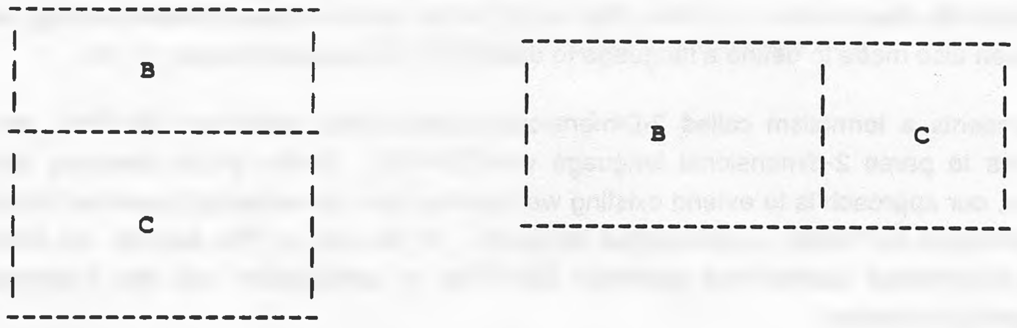


Figure 1-1: Examples of Adjacent Regions

Let G be a 2D-CFG (N, Σ, P_H, P_V, S), where

N: a set of non-terminal symbols
Σ: is a set of terminal symbols
P_H: a set of horizontal production rules
P_V: a set of vertical production rules
S: start symbol

Let LEFT(p) be the left hand side symbol of p. Let RIGHT(p, i) be the i-th right hand side symbol of p. Without loss of generality, we assume each rule in P_H is either in the form of

```
A --> B C    or    A --> b
```

Parsing 2-Dimensional Language

Masaru Tomita
Computer Science Department
and
Center for Machine Translation
Carnegie-Mellon University
Pittsburgh, PA 15213[1]

Abstract

2-Dimensional Context-Free Grammar (2D-CFG) for 2-dimensional input text is introduced and efficient parsing algorithms for 2D-CFG are presented. In 2D-CFG, a grammar rule's right hand side symbols can be placed not only horizontally but also vertically. Terminal symbols in a 2-dimensional input text are combined to form a rectangular *region*, and regions are combined to form a larger region using a 2-dimensional phrase structure rule. The parsing algorithms presented in this paper are the 2D-Earley algorithm and 2D-LR algorithm, which are 2-dimensionally extended versions of Earley's algorithm and the LR(0) algorithm, respectively.

1. Introduction

Existing grammar formalisms and formal language theories, as well as parsing algorithms, deal only with one-dimensional strings. However, 2-dimensional layout information plays an important role in understanding a text. It is especially crucial for such texts as title pages of articles, business cards, announcements and formal letters to be read by an optical character reader (OCR). A number of projects [11, 6, 7, 2], most notably by Fujisawa *et al.* [4], try to analyze and utilize the 2-dimensional layout information. Fujisawa *et al.*, unlike others, uses a procedural language called Form Definition Language (FDL) [5, 12] to specify layout rules. On the other hand, in the area of image understanding, several attempts have been also made to define a language to describe 2-dimensional images [3, 10].

This paper presents a formalism called 2-Dimensional Context-Free Grammar (2D-CFG), and two parsing algorithms to parse 2-dimensional language with 2D-CFG. Unlike all the previous attempts mentioned above, our approach is to extend existing well-studied (one dimensional) grammar formalisms and parsing techniques to handle 2-dimensional language. In the rest of this section, we informally describe the 2-dimensional context-free grammar (2D-CFG) in comparison with the 1-dimensional traditional context-free grammar.

Input to the traditional context-free grammar is a string, or *sentence*; namely a one-dimensional array of terminal symbols. Input to the 2-dimensional context-free grammar, on the other hand, is a rectangular block of symbols, or *text*; namely, a 2-dimensional array of terminal symbols.

In the traditional context-free grammar, a non-terminal symbol represents a *phrase*, which is a substring of the original input string. A grammar rule is applied to combine adjoining phrases to form a larger phrase. In the 2-dimensional context-free grammar, on the other hand, a non-terminal represents a *region*, which is a rectangular sub-block of the input text. A grammar rule is applied to combine two adjoining regions to form a larger region. Rules like

[1]This research was supported by the National Science Foundation under contract IRI-8858085.

Method:

For each $p \in P_H \cup P_V$ such that LEFT(p) = S, add an item (p, 0, 0, 0, n, m) to I_{00}.

```
For each item (p, d, x, y, X, Y) in $I_{ij}$,
    If d = |p|, do COMPLETOR
    If RIGHT(p, d+1) $\in$ N, do PREDICTOR
    If RIGHT(p, d+1) $\in \Sigma$, do SHIFTER
```

PREDICTOR: For all $q \in P_H \wedge P_V$ such that LEFT(q) = RIGHT(p, d+1), add an item (q, 0, i, j, X, Y) to I_{ij}.

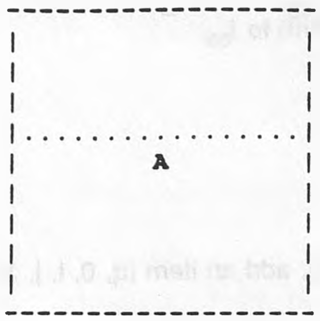
SHIFTER: If $a_{i+1,j+1}$ = RIGHT(p, d+1), and if $i<X \wedge j<Y$, then add an item (p, d+1, i, j, X, j+1) to $I_{i+1,j}$.

COMPLETOR: For all items (p', d', x', y', X', Y') in I_{xy} such that RIGHT(p', d'+1) = LEFT(p), do the following:

- **Case 1.** $p \in P_H \wedge p' \in P_H$ ---- Add an item (p', d'+1, x', y' X', Y) to I_{ij}, if Y'=Y $\vee$ d'=0.
- **Case 2.** $p \in P_V \wedge p' \in P_H$ ---- Add an item (p', d'+1, x', y' X', Y) to I_{Xy}, if Y'=Y $\vee$ d'=0.
- **Case 3.** $p \in P_H \wedge p' \in P_V$ ---- Add an item (p', d'+1, x', y' X, Y') to I_{xY}, if X'=X $\vee$ d'=0.
- **Case 4.** $p \in P_V \wedge p' \in P_V$ ---- Add an item (p', d'+1, x', y' X, Y') to I_{ij}, if X'=X $\vee$ d'=0.

```
(1)  S --> A A        (3)  B --> b               b b
                                                 c d
(2)  A --> B          (4)  C --> c
           C
                      (5)  C --> d
```

Figure 2-1: Example Grammar and Text



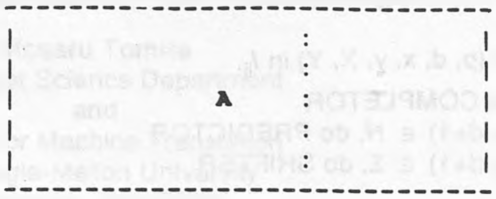


Figure 1-2: After applying rule (2) and (1), respectively

and each rule in P_V is in the form of

```
A --> B
      C
```

Where A,B,C $\in$ N and b $\in$ Σ. This form of grammar is called *2-dimensional Chomsky Normal Form (2D-CNF)*, and an arbitrary 2D-CFG can be converted into 2D-CNF. The conversion algorithm is very similar to the standard CNF conversion algorithm, and we do not describe the algorithm in this paper.

The subsequent two sections present two efficient 2D parsing algorithms: 2D-Earley and 2D-LR.

2. The 2D-Earley Parsing Algorithm

Input:

2D-CFG G = (N, Σ, P_H, P_V, S) and an input text

$$\begin{array}{llll} a_{11} & a_{21} & \cdots\cdots & a_{n1} \\ a_{12} & a_{22} & \cdots\cdots & a_{n2} \\ \cdots & \cdots & \cdots\cdots & \cdots \\ a_{1m} & a_{2m} & \cdots\cdots & a_{nm} \end{array}$$

where $a_{ij} \in \Sigma$.

Output:

A parse table

$$\begin{array}{llll} I_{00} & I_{10} & \cdots\cdots & I_{n0} \\ I_{01} & I_{11} & \cdots\cdots & I_{n1} \\ \cdots & \cdots & \cdots\cdots & \cdots \\ I_{0m} & I_{1m} & \cdots\cdots & I_{nm} \end{array}$$

I_{ij} is a set of *items* and each item is (p, d, x, y, X, Y), where p is a rule in P_H or P_V, d is an integer to represent its dot position ($0 \le d \le |p|$, where $|p|$ represents the length of p's left hand side). The integers x and y represent the item's origin (x,y) or the upper-left corner of the region being constructed by the item. The integers X and Y represent its perspective lower-right corner, and the parser's horizontal (vertical) position should never exceed X (Y) until the item is completed.

3. The 2D-LR Parsing Algorithm

A 2D-LR(0) parsing table consists of three parts: ACTION, GOTO-RIGHT and GOTO-DOWN. Figure 3-1 is a 2D-LR(0) table obtained from the grammar in Figure 2-1.

ST	ACTION				GOTO-RIGHT				GOTO-DOWN			
	b	c	d	$	S	A	B	C	S	A	B	C
0	sh3				8	1					4	
1	sh3					2					4	
2	re1	re1	re1	re1								
3	re3	re3	re3	re3								
4		sh6	sh7									5
5	re2	re2	re2	re2								
6	re4	re4	re4	re4								
7	re5	re5	re5	re5								
8				acc								

Figure 3-1: A 2D-LR Parsing Table

As in Standard LR parsing, the runtime parser performs shift-reduce parsing with a stack guided by this 2D-LR table. Unlike standard LR(0), however, each item in the stack is represented as (s, x, y, X, Y), where s is an LR state number, and (x,y) represents the current position in the input text. X and Y represent right and lower limits, respectively, and no positions beyond these limits should ever be explored until this state is popped off the stack.

Initially the stack has an item (0, 0, 0, n, m), where n and m are the number of columns and rows in the input text, respectively.

Now let the current elements in the stack be

$$\ldots \text{---} (s_3, x_3, y_3, X_3, Y_3) \text{---} B_2 \text{---} (s_2, x_2, y_2, X_2, Y_2) \text{---} B_1 \text{---} (s_1, x_1, y_1, X_1, Y_1)$$

where the right most element is the top of the stack. Also assume that the current input symbol a_{ij} is b, where $i = x_1+1$ and $j = y_1+1$. According to the parsing table, we perform SHIFT, REDUCE or ACCEPT.

SHIFT:

If ACTION(s_1, b) = sh s_0, then if $x_1 < X_1 \wedge y_1 < Y_1$, push b and ($s_0$, x_1+1, y_1, X_1, y_1+1) onto the stack.

REDUCE:

If ACTION(s_1, b) = re p, then let k be |p|+1 and do the following:

- **Case 1.** $p \in P_H$ and GOTO-RIGHT(s_k, LEFT(p)) = s_0 ---- If $Y_{k-1} = Y_1$ then pop 2*|p| elements from the stack, and push LEFT(p) and (s_0, x_1, y_1, X_k, Y_1).
- **Case 2.** $p \in P_H$ and GOTO-DOWN(s_k, LEFT(p)) = s_0] ---- If $Y_{k-1} = Y_1$ then pop 2*|p| elements from the stack, and push LEFT(p) and (s_0, x_k, Y_1, x_1, Y_k).
- **Case 3.** $p \in P_V$ and GOTO-RIGHT(s_k, LEFT(p)) = s_0 ---- If $X_{k-1} = X_1$ then pop 2*|p| elements from the stack, and push LEFT(p) and (s_0, X_1, y_k, X_k, y_1).
- **Case 4.** $p \in P_V$ and GOTO-DOWN(s_k, LEFT(p)) = s_0 ---- If $X_{k-1} = X_1$ then pop 2*|p| elements from the stack, and push LEFT(p) and (s_0, x_1, y_1, X_1, Y_k).

Figure 3-2 shows an example trace of 2D-LR parsing with the grammar in Figure 2-1.

```
             .              |      .              |        .
        S --> A A  0,0,2,2  | B --> b    0,0,2,1  | B --> b    1,0,2,1
                            |        .            |
                            | S --> A A  0,0,2,2  |
             .              |                     |          .
        A --> B    0,0,2,2  |                     | S --> A A  0,0,2,2
0             C             |                     |
                            |      .              |
                            | A --> B    1,0,2,2  |
                            |       C             |
             .              |                     |
        B --> b    0,0,2,2  |                     |
                            |      .              |
                            | B --> b    1,0,2,2  |
        --------------------b---------------------b--------------------
                            |        .            |        .
        A --> .B   0,0,1,2  | C --> c    0,1,1,2  | C --> d    1,1,2,2
               C            |                     |
                            | A --> .B   1,0,2,2  |
                            |        C            |
             .              |                     |
1       C --> c    0,1,1,2  |                     |
                            |      .              |
                            | C --> c    1,1,2,2  |
             .              |                     |
        C --> d    0,1,1,2  |                     |
                            |      .              |
                            | C --> d    1,1,2,2  |
        --------------------c---------------------d--------------------
                            |                     |
        A --> B    0,0,1,2  | A --> B    1,0,2,2  |
             .C             |      .C             |
                            |                     |
2                           |                     |
                            |                     |
                            |                     |
                 0                     1                      2
```

Figure 2-2: An Example of 2D-Earley Parsing

```
A1 --> c                        B1 --> b                        C1 --> c

A2 --> B1 A1 B1                        b                               c
                                B1 --> B1                       C1 --> C1
                                       b                               c
       B2
A3 --> A2
       B2                       B2 --> b                        C2 --> c

A4 --> C1 A3 C1                 B2 --> b B2 b                   C2 --> c C2 c

       C2
A1 --> A4                       START --> A1
       C2
```

```
                                   ccccccccccccc
                                   cbbbbbbbbbbbc
                        ccccccccc  cbcccccccccbc
               bbbbbbb  cbbbbbbbc  cbcbbbbbbbcbc
        ccccc  bcccccb  cbcccccbc  cbcbcccccbcbc
   bbb  cbbbc  bcbbbcb  cbcbbbcbc  cbcbcbbbcbcbc
c  bcb  cbcbc  bcbcbcb  cbcbcbcbc  cbcbcbcbcbcbc
   bbb  cbbbc  bcbbbcb  cbcbbbcbc  cbcbcbbbcbcbc
        ccccc  bcccccb  cbcccccbc  cbcbcccccbcbc
               bbbbbbb  cbbbbbbbc  cbcbbbbbbbcbc
                        ccccccccc  cbcccccccccbc
                                   cbbbbbbbbbbbc
                                   ccccccccccccc
```

Figure 4-1: Example Grammar I

```
A1 --> c                        B1 --> b                        C1 --> c

A2 --> A1 B1                           b                               c
                                B1 --> B1                       C1 --> C1
       B2
A3 --> A2                       B2 --> b                        C2 --> c

A4 --> C1 A3                    B2 --> b B2                     C2 --> c C2

A1 --> A4                       START --> A1
       C2
```

```
                                   cbbbbbbbbbbbb
                                   ccbbbbbbbbbbb
                        cbbbbbbbb  cccbbbbbbbbbb
               cbbbbbb  ccbbbbbbb  ccccbbbbbbbbb
        cbbbb  ccbbbbb  cccbbbbbb  cccccbbbbbbbb
   cbb  ccbbb  cccbbbb  ccccbbbbb  ccccccbbbbbbb
c  ccb  cccbb  ccccbbb  cccccbbbb  cccccccbbbbbb
   ccc  ccccb  cccccbb  ccccccbbb  ccccccccbbbbb
        ccccc  ccccccb  cccccccbb  cccccccccbbbb
               ccccccc  ccccccccb  ccccccccccbbb
                        ccccccccc  cccccccccccbb
                                   ccccccccccccb
                                   ccccccccccccc
```

Figure 4-2: Example Grammar II

```
(0,0,0,2,2)
(0,0,0,2,2)  b  (3,0,1,2,1)
(0,0,0,2,2)  B  (4,0,1,1,2)
(0,0,0,2,2)  B  (4,0,1,1,2)  c  (6,1,1,1,2)
(0,0,0,2,2)  B  (4,0,1,1,2)  C  (5,0,2,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  b  (3,2,0,2,1)
(0,0,0,2,2)  A  (1,1,0,2,2)  B  (4,1,1,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  B  (4,1,1,2,2)  d  (7,2,1,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  B  (4,1,1,2,2)  C  (5,1,2,2,2)
(0,0,0,2,2)  A  (1,1,0,2,2)  A  (2,2,0,2,2)
(0,0,0,2,2)  S  (8,2,0,2,2)
```

Figure 3-2: Example Trace of 2D-LR Parsing

4. More Interesting 2D Grammars

This section presents a couple of more interesting example grammars and texts. Example Grammar I generates nested rectangles of b's and c's, one after the other. In the grammar, B1 represents vertical bars (sequences) of b's, and B2 represents horizontal bars of b's. Similarly, C1 and C2 represent vertical and horizontal bars of c's, respectively. A1 then represents rectangles surrounded by c's. A2 represents rectangles surrounded by c's which are sandwiched by two vertical bars of b's. A3 further sandwiches A2 with two horizontal b bars, representing rectangles surrounded by b's. Similarly, A4 sandwiches A3 with two vertical c bars, and A1 further sandwiches A4 with two horizontal c bars, representing rectangles surrounded by c's.

A similar analysis can be made for Grammar II, which generates triangles of b's and c's.

Grammar III generates all rectangles of a's which have exactly 2 b's somewhere in them. Xn represents horizontal lines of a's with n b's. Thus, X0, X1 and X2 represent lines of a's, keeping track of how many b's are inside. Yn then combines those lines vertically, keeping track of how many a's have been seen thus far (n being the number of b's). Therefore, Y2 contains exactly two b's.

The example given in this section is totally deterministic. In general, however, a 2D-LR table may have multiple entries, or both GOTO-DOWN and GOTO-RIGHT may be defined from an identical state with an identical symbol. Such nondeterminism can also be handled efficiently using a *graph-structured stack* as in Generalized LR Parsing [8, 9].

5. Concluding Remarks

In this paper, 2D-CFG, 2-dimensional context-free grammar, has been introduced, and two efficient parsing algorithms for 2D-CFG have been presented. Traditional one-dimensional context-free grammars are well studied and well understood (e.g. [1]), and many of their theorems and techniques might be extended and adopted for 2D-CFG, as we have done in this paper for Earley's algorithm and LR parsing.

```
X0 --> [empty]             Y0 --> [empty]             Y2 --> Y0
                                                             X2
X0 --> X0 a                Y0 --> Y0
                                  X0                  Y2 --> Y1
X1 --> X0 b                                                  X1

                           Y1 --> Y0
X1 --> X1 a                       X1                  Y2 --> Y2
                                                             X0

X2 --> X1 b                Y1 --> Y1
                                  X0                  START --> Y2
X2 --> X2 a

        aaaaaaaaaaaaa             aaa                 aaaaaaa
        aaabaaaaaaaaa             aaa                 aaaabaa
        aaaaaaaaaaaaa             bba                 aabaaaa
        aaaaaaaaaaaab             aaa
        aaaaaaaaaaaaa             aaa
                                  aaa
                                  aaa
                                  aaa
```

Figure 4-3: Example Grammar III

References

[1] Aho, A. V. and Ullman, J. D.
The Theory of Parsing, Translation and Compiling.
Prentice-Hall, Englewood Cliffs, N. J., 1972.

[2] Akiyama, T. and Masuda, I.
A Method of Document-Image Segmentation Based on Projection Profiles, Stroke Density and Circumscribed Rectangles.
Trans. IECE J69-D(8):1187-1196, 1986.

[3] K. S. Fu.
Syntactic Pattern Recognition.
Springer-Verlag, 1977.

[4] Fujisawa, H. et al.
Document Analysis and Decomposition Method for Multimedia Contents Retrieval.
Proc. 2nd International Symposium on Interoperable Information Systems :231, 1988.

[5] Higashino, J., Fujisawa, H., Nakano, Y. and Ejiri, M.
A Knowledge-Based Segmentation Method for Document Understanding.
Proc. 8th Int. Conf. Pattern Recognition :745-748, Oct., 1986.

[6] Inagaki, K., Kato, T., Hiroshima, T. and Sakai, T.
MACSYM: A Hierarchical Image Processing System for Event-Driven Pattern Understanding of Documents.
Pattern Recognition 17(1):85-108, 1984.

[7] Kubota, K. et al.
Document Understanding System.
In *Proc. 7th Int. Conf. Pattern Recognition*, pages 612-614. , 1984.

[8] Tomita, M.
Efficient Parsing for Natural Language.
Kluwer Academic Publishers, Boston, MA, 1985.

[9] Tomita, M.
An Efficient Augmented-Context-Free Parsing Algorithm.
Computational Linguistics 13(1-2):31-46, January-June, 1987.

[10] Watanabe, S. (ed.).
Frontiers of Pattern Recognition.
Academic Press, 1972.

[11] Wong, K., Casey, R. and Wahl, F.
Document Analysis System.
IBM J. Research and Development 26(6):647-656, 1982.

[12] Yashiro, H. et.al.
A New Method of Document Structure Extraction.
In *International Workshop on Industrial Applications of Machine Intelligence and Vision (MIV-89)*, pages 282. , April, 1989.

is small. Only reduced, streamlined feature information is available in each entry; subcategorization, or valency, information is not distinguished by word senses.

2. The second dictionary access (for reattachment) consults a far richer source than before. For English, we make central use of online dictionary entries -- both their definitions and their example sentences. W7 and the *Longman Dictionary of Contemporary English* (LDOCE) are available to us. We can parse the definitions and examples with PEG, and use the syntactic information that PEG provides in order to bootstrap our way into semantics. The amount of information per word obtainable during this second access is huge -- much greater than what is typically described, even for lexicalist systems.

3. The third access (for paragraph modeling) again includes full natural language text. Since this component is only at a very early stage, there is not much to be said about it. We envision a NL knowledge base that contains information from every available source, from word lists to dictionaries and beyond, to encyclopedias.

It is interesting that the purposes of the separate components divide so neatly along linguistic levels: syntax, semantics, discourse. We do not mean to insist that the ultimate version of this system would need to have its components so cleanly divided. Neither has separation of the components been done for reasons of theoretical elegance or symmetry, but simply because the necessities of broad-coverage NLP have brought it about.

1. A syntactic sketch: PEG

PEG is an augmented phrase structure grammar which has been useful in a number of different settings -- text critiquing and machine translation, to name two. PEG's significant characteristics include:

- binary rules, in most cases (Jensen 1987);
- a wealth of conditions on the operation of the rules -- conditions that range from those that are strongly general, and express real grammatical patterns of the language, to those that are quite specific, and are intended to filter out certain semantically anomalous parses;
- a "relaxed" or "textual" approach to parsing, which means that we consistently avoid the use of selectional ("semantic") information to condition the parse, and that we also try, in so far as possible, to avoid, or at least to soften, the use of subcategorization (valency) information for that purpose. We assume, for example, that almost any verb can have a sense which will fit almost any frame; and that almost any noun might be used as an argument to almost any verb; and that the job of a computational parsing grammar is not to separate grammatical and ungrammatical sentences, but to provide the most reasonable analysis for any input string. The system is certainly able to distinguish grammatical from ungrammatical input, but this can be done by commenting on, rather than by failing to accept, an ungrammatical string.

The lexicon that supports this initial syntactic parse started out, in 1981, as a list of all the main entries in W7 -- minus, of course, morphological variants that could be productively described by rules. W7 claims to have 130,000 entries; after morphological variants were subtracted, the list contained 63,850 entries. That number has been increased from time to time; it now stands at roughly 70,000. As stated earlier, the goal of this lexicon is to supply useful syntactic information for every word of the language, including neologisms.

Because it contains so many entries, this lexicon provides very broad coverage. However, for each entry it contains only very limited information. The information is for parts of speech, morphology (tense, number, etc.), and word class features (transitive, ditransitive, factive, etc.). The features are mostly binary (present or absent), but include some lists, such as lists of verbal particles.

Word class features are valency features -- granted. But both the presentation and the use of these features are different from what is described for most other parsing systems. First, no attempt is made to specify the nature of the valency arguments. Second, although different parts of speech for a single word are listed and marked separately, all other sense distinctions, within each part of speech, are collapsed. One lexical item might have many, often contradictory, feature markings. The word "go," for example, appears in the lexicon as follows:

A Broad-coverage Natural Language Analysis System

Karen Jensen
IBM
April, 1989

0. Introduction

This paper discusses the components of our broad-coverage natural language analysis system, as they appear at this time.

A broad-coverage goal requires a robust and flexible natural language processing base, one that is adaptable to linguistic needs and also to the exigencies of computation. The Programming Language for Natural Language Processing (PLNLP: Heidorn 1972) is well suited for this task. PLNLP provides a general programming capability, including a rule-writing formalism and algorithms for both parsing ("decoding") and generation ("encoding"). Although linguistic scholarship and linguistic intuitions motivate our system strongly, we have chosen not to commit our computational formalism to any of the reigning linguistic theories. To quote Ron Kaplan:

> the problem is that, at least in the current state of the art, (linguists) don't know which generalizations and restrictions are really going to be true and correct, and which are either accidental, uninteresting or false. The data just isn't in... (Kaplan 1985, p. 5)

So our work is experimental, descriptive, and data-driven. This does not mean that it has no theoretical implications. Any functioning unit of this size is an embodiment of some theory. The theory behind this program of grammar development just hasn't been thoroughly articulated yet.

The system that is emerging has, so far, three components:

1. The PLNLP English Grammar (PEG) makes an initial syntactic analysis for each input sentence (Jensen 1986).

2. The reattachment component takes syntactically consistent, but semantically inaccurate, parses, and then reattaches constituents, when necessary, based on information gained from a rich semantic data base (Jensen and Binot 1987).

3. The paragraph modelling component receives sentence parses and, for connected text, builds them into logically consistent and coherent models of the chunks of discourse that are typically called paragraphs (Zadrozny and Jensen 1989).

Hand-in-hand with each of these components goes a separate dictionary access.

1. The first dictionary access (for PEG) is to a lexicon that is essentially just a glorified word list. However, it is a word list that, when coupled with morphological rules and a default strategy provided by the access mechanism, aims at supplying an entry for every word of the language, including neologisms. We started with the full online *Webster's Seventh New Collegiate Dictionary* (W7). We have modified this word list somewhat, but only to enlarge it -- never to reduce its scope. Although the word coverage is great, the amount of information per word

to other records. For example, the value of the PRMODS attribute is a pointer to the noun phrase (NP1) which covers the noun "geometry."

All of the analysis information is carried in the record structure. For ease of recognition, however, we also display a variant of the standard parse tree:

```
------------------------------------------------------------------
DECL1  NP1     NOUN1*  "geometry"
       VERB1*  "is"
       NP2     DETP1   ADJ1*   "a"
               AJP1    AVP1    ADV1*   "very"
                       ADJ2*   "old"
               NOUN2*  "science"
       PUNC1   "."
------------------------------------------------------------------
```

Figure 2. Parse tree for the same sentence

Note that the start node presents the value of the SEGTYP2 attribute from Fig. 1, plus a number (each node is numbered for easy reference). The other, fairly standard, node names are the values of the SEGTYP2 attributes in their corresponding records. Trees are produced by a routine that uses just five attributes from the record structure: PRMODS, HEAD, PSMODS, SEGTYP2, and STR. Since such a tree is conventionally said to depict phrase- or constituent-structure, it might be said that these five attributes make up the *constituent structure* for the parse.

More than constituent structure is contained in the records, however. During the operation of the grammar rules, attributes are assigned that point to subject, object, indirect object, predicate nominative, etc. In other parlance, these might be assigned by "...a function that goes from the nodes of a tree into f-structure space" (Kaplan 1985, p. 11). Figure 1 shows two examples, SUBJECT and PREDNOM. Such attributes, and their values, could be said to present the *functional structure*. The TOPIC of the sentence is also computed, based on some exploratory work done in Davison 1984. Other attributes will be added during further processing, and these attributes will define higher levels of analysis. Progress in the analysis seems not to involve jumping between levels, but rather a smooth accumulation (and sometimes an erasing) of attributes and values.

Now, some people might object that the same analysis could be obtained by using subcategorization frames (together, perhaps, with selectional features on NPs), either as conditions on the rules or, within a lexicalist framework, as statements within the dictionary, to be honored by the rules. According to this way of thinking, we would control multiple parses by exercising valency information, not by ignoring it. From experience, we have found this to be a dangerous path, for several reasons. The most forceful reason is that real text (at least, real English text) just does not behave in the well-disciplined fashion that such specifications would require. If we really want to do broad-coverage parsing, then we have to be prepared for many imaginative uses of words to occur; and strict subcategorization does not allow for that.

Strict subcategorization expects, for example, that verbs will occur in well-defined contexts. "Give" should be either transitive or ditransitive, surely not intransitive. But what about the sentence "I gave at the office"? It's no good saying that there is an "understood" NP; if the computational grammar depends on the presence of at least one object in context, then this sentence will fail to parse. And even though there are subcategorizational differences between "go" and "know" (by our own earlier definitions), it is possible to use "go" with a *that*-complement, as in:

I said, no. And then he goes, "See you later."

or with a *wh*-complement, as in:

We'll go whatever amount (i.e., bail) is necessary.

These real-life facts of language tend in one direction: stated in extreme form, any word can, and might, be used in any context. But to mark every verb in the lexicon with every possible subcat-

go(NOUN SING)
go(VERB COPL INF PLUR PRES TRAN)

The first definition of "go," as a SINGular NOUN, collapses two different noun entries for "go" in W7. One is the Japanese game; the other has seven subsenses, including "the act or manner of going"; "the height of fashion"; etc. The definition of "go" as a VERB collapses 19 intransitive or COPLulative senses (e.g., "to go crazy"), and six TRANsitive senses (e.g., "to go his way," "to go bail for").

The word "know" also has two entries:

know(NOUN SING)
know(VERB INF NPTOV PLUR PRES THATCOMP TRAN WHCOMP)

This means that "know" can be a singular noun ("in the know") or a verb. If it is a verb, besides being INFinitive, PLURal, and PRESent, it might be expected, with fair frequency, to have one of the following complementation types:

NPTOV: We know him to be a good man.
THATCOMP: We know that he is here.
TRAN: We know him.
WHCOMP: We know what he wants.

The great advantage to this collapsing strategy (affectionately known as "smooshing") is that it helps to avoid multiple parses in a simple, straightforward way. And this is no trivial accomplishment: a broad-coverage, bottom-up parallel parser can easily strangle on proliferating parses. With simple lexical information, however, we can expect a manageable number of parses, even in the worst case. We aim for a single parse that carries forward all of the necessary data. We like to think of this as a syntactic sketch; we have also called it an "approximate parse." The techniques for writing this kind of grammar are varied, and use all sorts of syntactic and morphological hooks. We can exploit the presence of valency features, but we try to blunt their force, using them to favor one situation over another, rather than as strict necessary conditions for the success of a certain rule.

The result of the operation of PEG's augmented phrase structure rules, coupled with the streamlined lexicon just described, is an attribute-value data structure (in PLNLP terms, a "record structure"). Here is a somewhat pared-down example of the top-level record produced from the simple input sentence, "Geometry is a very old science":

```
SEGTYPE    'SENT'
SEGTYP2    'DECL'
STR        " geometry is a very old science"
RULES      4000 4080 5080 7200
BASE       'BE'
POS        VERB
INDIC      SING PRES COPL PERS3
PRMODS     NP1 "geometry"
HEAD       VERB1 "is"
PSMODS     NP2 "a very old science"
PSMODS     PUNC1 "."
SUBJECT    NP1 "geometry"
PREDNOM    NP2 "a very old science"
TOPIC      NP1 "geometry"
```

Figure 1. PLNLP record for "Geometry is a very old science"

Attribute names are in the left-hand column; their values are to the right. The attributes SEGTYPE and SEGTYP2 refer to different labelings of the topmost node; STR has as its value the character string covered by this node; and RULES contains a list of rule numbers, a derivational history for the parse at this level. POS indicates the possible parts of speech of the BASE; the INDICator features are fairly self-explanatory. Most of the values in Fig. 1 are actually pointers

The question mark indicates doubt about the acceptability of the coordinate NP inside PP5: "the river Nile and the consequent destroying of the boundaries of farm lands." Should NP4, "the consequent destroying...," be *and*-ed with NP2, "the river Nile," or with the NP in PP3, "the annual overflow..."?

Question marks are placed at various points in the parse tree by a routine that is sensitive to problematic constructions in English. We could have produced two separate analyses; but, given the large number of such attachment situations, this approach would have led straight to the fatal trap of proliferating parses. The question marks, in effect, collapse different possible parses, and allow for efficient handling of ambiguities (Jensen 1986, pp. 22-23).

Human readers of the sentence will not hesitate to say that the NP attachment shown in PP3 of Figure 3 is not the intended one; the attachment indicated by the question mark is what we want. Our problem is how to enable the computer to determine that.

The sort of information that enables the right decision to be made, in this and similar cases, generally falls under the rubric of "background" or "commonsense" knowledge. The usual method for making such knowledge available to a computer program has been to hand-code the relevant concepts, in whatever format. Although some hand-coding will undoubtedly be necessary and valuable, we approach the problem from another angle.

Written text is itself a rich source of information. It can be viewed as a knowledge base; the language that it is written in, even though this is a natural language, is a knowledge representation language. In particular, reference works like dictionaries actually contain a storehouse of commonsense knowledge. We can parse the entries in an online dictionary with a syntactic grammar, and retrieve a surprising amount of the information that is necessary to resolve syntactic ambiguities, like the one displayed in Fig. 3 (Binot and Jensen 1987, Jensen and Binot 1988).

The problem presented in Fig. 3 reduces to a question: which of the following pairs is more likely?

- *overflow* and *destroying*
- *Nile* and *destroying*

Bearing in mind the old adage that "likes conjoin," we will consider that pair more likely whose terms can be more easily related through dictionary entries -- including both definitions and example sentences. (Das Gupta 1987 also uses dictionary entries for interpreting conjoined words.)

Decisions on where to start these search procedures will ultimately be important, but here we avoid them. Assume that we start with the first pair, first word. The noun definition for "overflow" in W7 begins:

overflow...n 1: a flowing over: INUNDATION

Here "inundation" is asserted to be a synonym for "overflow." The noun "inundation" has no definition of its own, but is merely listed under the verb "inundate":

inundate...vt...: to cover with a flood: OVERFLOW

The circularity of the synonym definitions is no problem, because now we can infer something new about "overflow": it involves the act of covering by means of a flood. The definition of "flood" in W7 is not much help, but in LDOCE, the first example sentence quoted in the entry for the noun "flood," when analyzed by PEG, takes us right where we want to go:

flood..n...1...The town was destroyed by the floods after the storm.

Focusing on only the relevant information, these dictionary entries present a small part of a conceptual network:

egorization frame would be absurd, of course. And to add some sort of 'recovery' procedures into the grammar would be costly. The most sensible way to regard subcategorization (valency frames) is as codified frequency information. A verb that is marked transitive is quite frequently used in its transitive sense -- that's all.

This does not mean that we ignore the semantic implications of valencies. On the contrary, what we do is postpone the differentiation of word senses until after the initial syntactic sketch is completed. This strategy allows us to get our hands on any input string, assign it some (reasonable, we hope) structure, and then interpret the input, whatever it might be. Before making the interpretation, however, the parse may have to pass through the reattachment component.

2. Semantic readjustment

No matter how clever the grammarian's exploitation of word order, word class, and morphological hooks is, there are many analyses in English that just will not yield a correct analysis from syntax alone. Among these are the correct attachment of prepositional phrases and of relative and other embedded clauses; the optimal structure of complex noun phrases; and the degree of structural ambiguity exhibited by coordinated elements (Langendoen, p.c.). There are no markers, in English, that serve to disambiguate these constructions; the plain fact is that semantic (or even broader, contextual) information is required.

Consider the following parse, summarized in Fig. 3 by its tree structure. Where the correct structure cannot be determined by syntax, attachment is arbitrarily made to the closest available node, encouraging right branching.

```
-----------------------------------------------------------
DECL2  NP6     DETP7   ADJ1*   "this"
               NOUN9*  "re-measuring"
               PP1     PP2     PREP1*  "of"
                       DETP2   ADJ2*   "the"
                       NOUN1*  "land"
       VERB2*  "was"
       AJP1    ADJ3*   "necessary"
               PP3     PP4     PREP2*  "due to"
                       DETP3   ADJ4*   "the"
                       AJP2    ADJ5*   "annual"
                       NOUN2*  "overflow"
                       PP5     PP6     PREP3*  "of"
                               NP2     DETP4   ADJ6*   "the"
                                       NP3     NOUN3*  "river"
                                       NOUN4*  "Nile"
                         ?     CONJ1*  "and"
                               NP4     DETP5   ADJ7*   "the"
                                       AJP3    ADJ8*   "consequent"
                                       NOUN5*  "destroying"
                                       PP7     PP8     PREP4*  "of"
                                               DETP6   ADJ9*   "the"
                                               NOUN6*  "boundaries"
                                               PP9     PP10    PREP5*  "of"
                                                       NP5     NOUN7*  "farm"
                                                       NOUN8*  "lands"
       PUNC1   "."
-----------------------------------------------------------
```

Figure 3. Parse tree for a sentence with structural ambiguity

We have not yet implemented this particular disambiguation, although it is similar to work reported on in Jensen and Binot 1987. Many technical issues remain to be investigated. For one example, there is the problem of how to combine two (or more) dictionaries -- in this case, W7 and LDOCE -- in a way that allows for efficient access to, and processing of, all the information that they contain. We want to set such problems aside for the moment, and assume that they will be solved. The point is that vast, rich, and potentially rewarding networks of information exist in written text, and much of that information is of the hitherto elusive "commonsense" sort.

This is our second dictionary access. The amount of information available at this stage of processing is immense and complexly structured. It is, needless to say, much greater than what is afforded by any of the current lexicalist frameworks. It avoids the pitfalls of straight hand-coding -- incompleteness, and time required -- and it points to a new way of looking at knowledge bases. The prospect of a system that uses natural language in order to understand natural language is pleasingly recursive. Words may yet prove to be the most adequate knowledge representation tools.

3. The paragraph as a discourse unit

Beyond the semantic readjustment component lies the whole world of connected text processing. This area is generally referred to as "discourse." We take the paragraph (loosely defined) to be the first formal unit of discourse. It is the smallest reasonable domain of anaphora resolution, and the smallest domain in which topic and coherence can be reliably defined (Zadrozny and Jensen 1989, p. 1, pp. 4ff).

The sentences in Figures 2 and 3 are actually part of a paragraph taken from a reading comprehension exercise in a well-known series used by countless prospective college students who want to prepare for the standard Scholastic Aptitude Test (Brownstein et al. 1987, pp. 144-5). Here is the complete text:

> Geometry is a very old science. We are told by Herodotus, a Greek historian, that geometry had its origin in Egypt along the banks of the river Nile. The first record we have of its study is found in a manuscript written by Ahmes, an Egyptian scholar, about 1550 B.C. This manuscript is believed to be a copy of a treatise which dated back probably more than a thousand years, and describes the use of geometry at that time in a very crude form of surveying or measurement. In fact, geometry, which means "earth measurement," received its name in this manner. This re-measuring of the land was necessary due to the annual overflow of the river Nile and the consequent destroying of the boundaries of farm lands. This early geometry was very largely a list of rules or formulas for finding the areas of plane figures. Many of these rules were inaccurate, but, in the main, they were fairly satisfactory.

Figure 6. Paragraph from Barron's,*How to prepare for the SAT*

PEG parse trees for the paragraph in Fig. 6, sentence by sentence, are presented in Appendix A.

If we are going to make discourse sense of this text, however, we need something more than a linear concatenation of syntactic sentence parses -- just as, in order to make syntactic sense out of a sentence, we need something more than a linear concatenation of words. A popular and effective way of modeling this non-linear set of sentence relationships is as a network with nodes connected by arcs (e.g., Sowa 1984). We can label the nodes with content words and the arcs with function (or relation) names, for a simple beginning. For now, we use a fairly intuitive set of relation names, rather than take the time to explain precisely how each arc gets labeled.

The basic network for one sentence derives not directly from the surface syntactic structure, but from the underlying predicate-argument structure, which itself is derived from the surface structure, after all necessary readjustments have been made (Jensen forthcoming). Here is a network representation, or model, for the first sentence in the geometry paragraph:

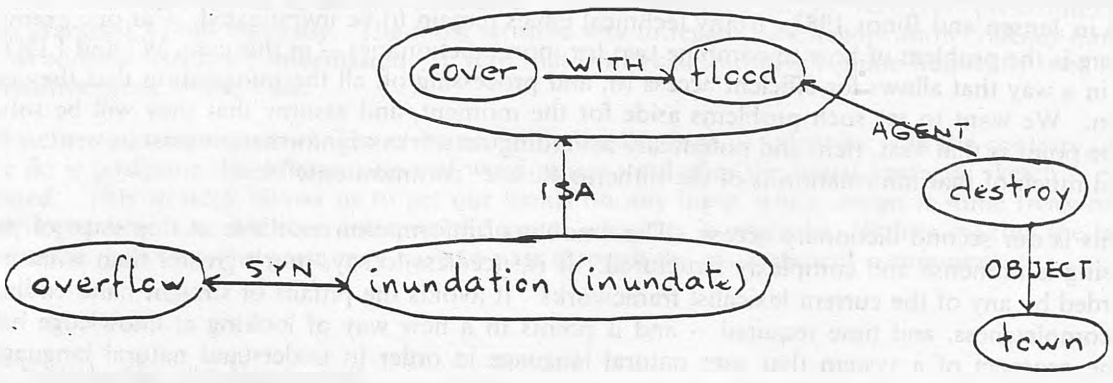


Figure 4. Network connecting "overflow" to "destroying"

and the path from "overflow" to "destroying" is clear in three steps.

Any attempt to connect "Nile" with "destroying" is bound to take longer. We can link "Nile" with "river" (this link is actually present in W7, in the Pronouncing Gazetteer): but we still have to get from "river" to "water," and then from "water" to "flood," and from "flood" to "destroy" (a total of four steps). The link between "water" and "flood" is also likely to incur a penalty, since moving from "water" to "flood" is difficult (i.e., "flood" does not appear in the definition of "water"), although moving in the reverse direction is easy ("water" does appear in the definition of "flood"). On this basis, we can revise the analysis of the sentence in Fig. 3 to reflect the more likely coordinate structure:

```
---------------------------------------------------------------------
DECL2   NP6     DETP7   ADJ1*   "this"
                NOUN9*  "re-measuring"
                PP1     PP2     PREP1*  "of"
                        DETP2   ADJ2*   "the"
                        NOUN1*  "land"
        VERB2*  "was"
        AJP1    ADJ3*   "necessary"
                PP3     PP4     PREP2*  "due to"
                        NP2     DETP3   ADJ4*   "the"
                                AJP2    ADJ5*   "annual"
                                NOUN2*  "overflow"
                                PP5     PP6     PREP3*  "of"
                                        NP3     DETP4   ADJ6*   "the"
                                                NP4     NOUN3*  "river"
                                                NOUN4*  "Nile"
                        CONJ1*  "and"
                        NP5     DETP5   ADJ7*   "the"
                                AJP3    ADJ8*   "consequent"
                                NOUN5*  "destroying"
                                PP7     PP8     PREP4*  "of"
                                        DETP6   ADJ9*   "the"
                                        NOUN6*  "boundaries"
                                        PP9     PP10    PREP5*  "of"
                                                NP6     NOUN7*  "farm"
                                                NOUN8*  "lands"
        PUNC1   "."
---------------------------------------------------------------------
```

Figure 5. Readjusted parse for sentence in Figure 3

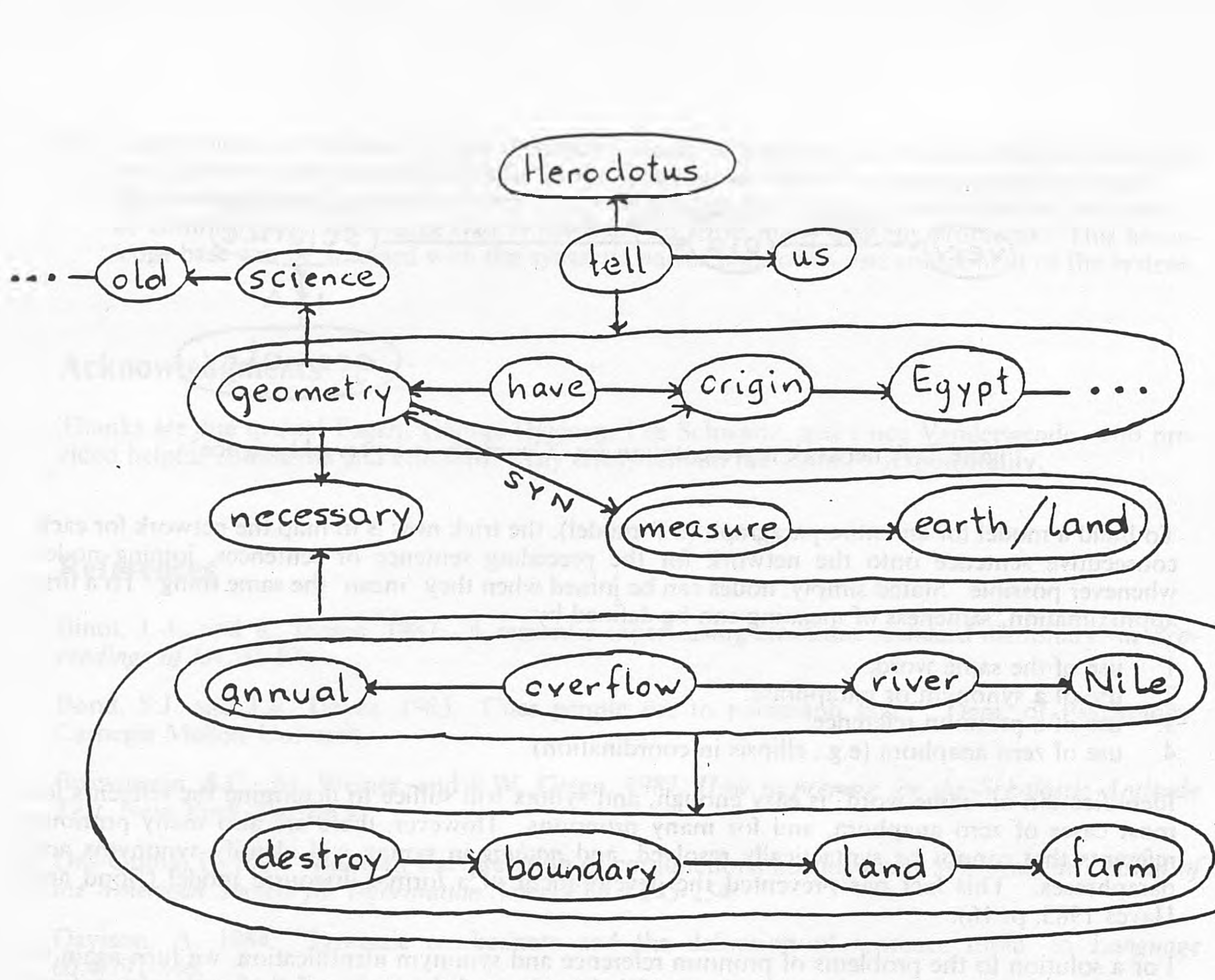


Figure 8. Partial P-model for the text in Figure 6

In order to build the link between "necessary" and "geometry," we have to know that "re-measuring of the land" is a paraphrase for "geometry." We are told that "earth measurement" is a synonym for "geometry" in the fifth sentence. Syntax allows us to say that "NOUN measurement" and "measurement of NOUN" are possible equals. If we can establish that "earth measurement" and "land re-measuring" are equals, then the problem is solved. "Measurement" and "re-measuring" are transparently related, so the problem reduces to finding a link between "earth" and "land."

This, of course, is quite easy to find in dictionaries and thesauri. In LDOCE, one definition of "earth" contains "land" as a synonym, and vice versa (actually, the first four definitions for "land" contain the word "earth" in a critical position in the parse). Similar conditions exist in W7. *Roget's Thesaurus* (RT) lists "land" as a synonym for "earth" and "earth" as a synonym for "land." Q.E.D.

The intended purpose for paragraphs like the one we have been playing with, of course, is to test a reader's comprehension ability by requiring sensible answers to questions based on the information in the paragraph. In Brownstein et al., the first test concerning our paragraph is

(1) The title below that best expresses the ideas of this passage is

and the possible solutions are

(A) Plane Figures
(B) Beginnings of Geometry
(C) Manuscript of Ahmes
(D) Surveying in Egypt
(E) Importance of the Study of Geometry

It is tempting to ask whether a program that is able to build and manipulate the P-model in Fig. 8 could also answer (1) successfully.

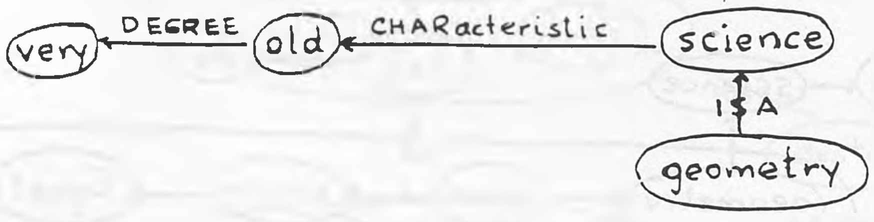


Figure 7. A network representation for "Geometry is a very old science"

To build a model for an entire paragraph (a P-model), the trick now is to map the network for each consecutive sentence onto the network for the preceding sentence or sentences, joining nodes whenever possible. Stated simply, nodes can be joined when they "mean" the same thing. To a first approximation, sameness of meaning can be defined by:

1. use of the same word;
2. use of a synonym or paraphrase;
3. use of a pronoun reference;
4. use of zero anaphora (e.g., ellipsis in coordination).

Identification of "same word" is easy enough, and syntax will suffice to determine the referents for most cases of zero anaphora, and for many pronouns. However, there are also many pronoun referents that cannot be syntactically resolved, and *nothing* in syntax will identify synonyms and paraphrases. This fact has prevented the development of a formal discourse model (Bond and Hayes 1983, p. 16).

For a solution to the problems of pronoun reference and synonym identification, we turn again to reference works written in natural language. Dictionaries and thesauri are full of such information.

Here is part of the model that can be built for the paragraph in Fig. 6. It includes information from only the first, second, fifth, and sixth sentences in that paragraph. Even so, many details have been left out:

- Each component makes its own dictionary access or accesses, and the dictionaries associated with different components will differ in the type and amount of information they contain.
- The written text of standard reference works is used as a repository for much of the background or commonsense knowledge that is necessary to solve many analysis problems. This knowledge base can be accessed with the syntactic parser that forms one component of the system.

Acknowledgments

Thanks are due to Joel Fagan, George Heidorn, Lee Schwartz, and Lucy Vanderwende, who provided helpful comments and criticism. Any errors remain the author's responsibility.

References

Binot, J.-L. and K. Jensen. 1987. "A semantic expert using an online standard dictionary" in *Proceedings of IJCAI-87*.

Bond, S.J. and J.R. Hayes. 1983. "Cues people use to paragraph text." Dept. of Psychology, Carnegie Mellon University.

Brownstein, S.C., M. Weiner, and S.W. Green. 1987. *How to prepare for the Scholastic Aptitude Test*. New York, Barron's.

Das-Gupta, P. 1987. "Boolean interpretation of conjunctions for document retrieval" in *Journal of the American Society for Information Science* 38.4.245-254.

Davison, A. 1984. "Syntactic markedness and the definition of sentence topic" in *Language* 60.4.797-846.

Heidorn, G.E. 1972. "Natural Language Inputs to a Simulation Programming System." Ph.D. dissertation, Yale University.

Jensen, K. 1986. "PEG 1986: A broad-coverage computational syntax of English." Unpublished paper.

Jensen, K. 1987. "Binary rules and non-binary trees" in A. Manaster-Ramer (ed.), *Mathematics of Language*. Amsterdam, John Benjamins, pp. 65-86.

Jensen, K. forthcoming. "PEGASUS: deriving predicate-argument structures from a syntactic parse."

Jensen, K. and J.-L. Binot. 1987. "Disambiguating Prepositional Phrase Attachments by Using On-Line Dictionary Definitions." in CL 13.3-4.251-60.

Kaplan, R. 1985. "Three seductions of computational psycholinguistics" in P. Whitelock, M.M. Woods, H.L. Somers, R. Johnson, P. Bennett (eds.), *Linguistic Theory and Computer Applications*. London, Academic Press, 1987, pp. 149-81.

Longman Dictionary of Contemporary English. 1978. Harlow and London, Longman Group Limited.

Roget's Thesaurus of English Words and Phrases. 1962. New York, St. Martin's Press.

Sowa, J.F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Reading, MA; Addison-Wesley.

Webster's Seventh New Collegiate Dictionary. 1967. Springfield, Mass., G. & C. Merriam Co.

Zadrozny, W. and K. Jensen. 1989. "Semantics of paragraphs." Unpublished paper.

Without going into any formal explanation of topic definition, let's assume that we can identify the node labeled "geometry" as the main idea, or topic, of the paragraph. (Note that it occupies a central position in the network.) So we discard all possible answers to (1) except for those that contain the word "geometry." This leaves us with two candidates, (B) and (E). We then search the graph around the "geometry" node, looking for related nodes that express either "beginnings" or "importance of the study of." The latter alternative is not easy to find. But the "origin" node can be immediately identified with "beginnings." In W7, the entry for "beginning" has "origin" as a synonym, and the second sense definition for "origin" is "rise, beginning, or derivation from a source..." Furthermore, "origin" and "beginning" are mutual synonyms in RT.

Resolving the referent for the possessive pronoun "its" in the second sentence of our test paragraph allowed us to draw the arc between the "geometry" and "origin" nodes in Fig. 8, which we now label:

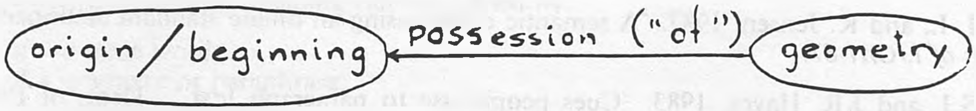


Figure 9. Network for the answer to (1)

In this subgraph, the preferred answer to question (1) is clear: the title that best expresses the ideas in the test passage is (B), "Beginnings of Geometry."

Obviously a tremendous amount of important detail has been left out in order to produce this blueprint for a formal model of a discourse unit. The challenges of implementation lie ahead. But the general structure seems promising, and most promising of all is the possibility of finding a repository of background knowledge, already coded for us, in online natural language sources.

Here is another comprehension question on the same paragraph:

(2) It can be inferred that one of the most important factors in the development of geometry as a science was

An answer must be picked from the following alternatives:

(A) Ahmes' treatise
(B) the inaccuracy of the early rules and formulas
(C) the annual flooding of the Nile Valley
(D) the destruction of farm crops by the Nile
(E) an ancient manuscript copied by Ahmes

We suggest that the preferred answer to (2) can also be found by using the P-model in Fig. 8, in conjunction with a good dictionary and thesaurus; and we leave this as an exercise for the interested reader.

4. Conclusion

This paper contains an overview of our broad-coverage NL analysis system, including components that already exist, that are currently being worked on, and that are projected for the future. Some aspects of our system that differentiate it from other NL analysis systems are

- It is not modeled along the lines of any currently accepted linguistic theory; rather it is highly experimental and data-driven.
- Separate components are emerging from this experimental process; they coincide roughly with the accepted linguistic levels: syntax, semantics, discourse.

```
Sentence 3:
--------------------------------------------------------------
DECL1   NP1     DETP1   ADJ1*   "the"
                AJP1    ADJ2*   "first"
                NOUN1*  "record"
                RELCL1  NP2     PRON1*  "we"
                        VERB1*  "have"
                        PP1     PP2     PREP1*  "of"
                                DETP2   ADJ3*   "its"
                                NOUN2*  "study"
        AUXP1   VERB2*  "is"
        VERB3*  "found"
        PP3     PP4     PREP2*  "in"
                DETP3   ADJ4*   "a"
                NOUN3*  "manuscript"
                PTPRTCL1VERB4*  "written"
         ?       ?      PP5     PP6     PREP3*  "by"
                                NOUN4*  "Ahmes"
                                PUNC1   ","
                                NAPPOS1 DETP4   ADJ5*   "an"
                                        NP3     NOUN5*  "Egyptian"
                                        NOUN6*  "scholar"
                                        PUNC2   ","
         ?       ?       ?       ?      PP7     PP8     PREP4*  "about"
                                                YEAR1*  "1550"
                                                LABEL1  NOUN7*  "B.C."
        PUNC3   "."
--------------------------------------------------------------
```

Appendix A

Sentence 1:

```
-----------------------------------------------------------------
DECL1  NP1     NOUN1*  "geometry"
       VERB1*  "is"
       NP2     DETP1   ADJ1*   "a"
               AJP1    AVP1    ADV1*   "very"
                       ADJ2*   "old"
               NOUN2*  "science"
       PUNC1   "."
-----------------------------------------------------------------
```

Sentence 2:

```
-----------------------------------------------------------------
DECL1  NP1     PRON1*  "we"
       AUXP1   VERB1*  "are"
       VERB2*  "told"
       PP1     PP2     PREP1*  "by"
               NOUN1*  "Herodotus"
               PUNC1   ","
               NAPPOS1 DETP1   ADJ1*   "a"
                       NP2     NOUN2*  "Greek"
                       NOUN3*  "historian"
                       PUNC2   ","
       VP1     COMPL1  "that"
               NP3     NOUN4*  "geometry"
               VERB3*  "had"
               NP4     DETP2   ADJ2*   "its"
                       NOUN5*  "origin"
                ?      PP3     PP4     PREP2*  "in"
                               NOUN6*  "Egypt"
                ?       ?      PP5     PP6     PREP3*  "along"
                                       DETP3   ADJ3*   "the"
                                       NOUN7*  "banks"
                                       PP7     PP8     PREP4*  "of"
                                               DETP4   ADJ4*   "the"
                                               NP5     NOUN8*  "river"
                                               NOUN9*  "Nile"
       PUNC3   "."
-----------------------------------------------------------------
```

Sentence 5:

```
-------------------------------------------------------------
DECL1  PP1      PP2      PREP1*   "in"
                NOUN1*   "fact"
                PUNC1    ","
       NP1      NOUN2*   "geometry"
                PUNC2    ","
        ?       RELCL1   NP2      PRON1*   "which"
                         VERB1*   "means"
                         NP3      PUNC3    """"
                                  NP4      NOUN3*   "earth"
                                  NOUN4*   "measurement"
                                  PUNC4    """ ,"
       VERB2*   "received"
       NP5      DETP1    ADJ1*    "its"
                NOUN5*   "name"
        ?       PP3      PP4      PREP2*   "in"
                         DETP2    ADJ2*    "this"
                         NOUN6*   "manner"
       PUNC5    "."
-------------------------------------------------------------
```

Sentence 6:

```
-------------------------------------------------------------
DECL1  NP1      DETP1    ADJ1*    "this"
                NOUN1*   "re-measuring"
                PP1      PP2      PREP1*   "of"
                         DETP2    ADJ2*    "the"
                         NOUN2*   "land"
       VERB1*   "was"
       AJP1     ADJ3*    "necessary"
                PP3      PP4      PREP2*   "due to"
                         DETP3    ADJ4*    "the"
                         AJP2     ADJ5*    "annual"
                         NOUN3*   "overflow"
                         PP5      PP6      PREP3*   "of"
                                  NP2      DETP4    ADJ6*    "the"
                                           NP3      NOUN4*   "river"
                                           NOUN5*   "Nile"
                          ?       CONJ1*   "and"
                                  NP4      DETP5    ADJ7*    "the"
                                           AJP3     ADJ8*    "consequent"
                                           NOUN6*   "destroying"
                                           PP7      PP8      PREP4*   "of"
                                                    DETP6    ADJ9*    "the"
                                                    NOUN7*   "boundaries"
                                                    PP9      PP10     PREP5*   "of"
                                                             NP5      NOUN8*   "farm"
                                                             NOUN9*   "lands"
       PUNC1    "."
-------------------------------------------------------------
```

```
Sentence 4:
--------------------------------------------------------------
DECL1  NP1     DETP1   ADJ1*   "this"
               NOUN1*  "manuscript"
       VP1     AUXP1   VERB1*  "is"
               VERB2*  "believed"
               INFCL1  INFTO1  "to"
                       VERB3*  "be"
                       NP2     DETP2   ADJ2*   "a"
                               NOUN2*  "copy"
                               PP1     PP2     PREP1*  "of"
                                       DETP3   ADJ3*   "a"
                                       NOUN3*  "treatise"
                                ?      RELCL1  NP3     PRON1*  "which"
                                               VERB4*  "dated"
                                               AVP1    ADV1*   "back"
                                               AVP2    AVP3    ADV2*
          "probably"
                                                       ADV3*   "more"
                                                       PP3     PP4     PREP2*
          "than"
                                                               QUANP1  ADJ4*
          "a thousand"
                                                               NOUN4*  "years"
                                                               PUNC1   ","
       CONJ1*  "and"
       VP2     VERB5*  "describes"
               NP4     DETP4   ADJ5*   "the"
                       NOUN5*  "use"
                       PP5     PP6     PREP3*  "of"
                               NOUN6*  "geometry"
                ?       ?      PP7     AVP4    ADV4*   "at that time"
                ?       ?       ?      PP8     PREP4*  "in"
                                       DETP5   ADJ6*   "a"
                                       AJP1    AVP5    ADV5*   "very"
                                               ADJ7*   "crude"
                                       NOUN7*  "form"
                                       PP9     PP10    PREP5*  "of"
                                               NP5     NOUN8*  "surveying"
                ?               ?       ?      CONJ2*  "or"
                                               NP6     NOUN9*  "measurement"
       PUNC2   "."
--------------------------------------------------------------
```

Sentence 7:

```
-------------------------------------------------------------
DECL1  NP1      DETP1   ADJ1*   "this"
                AJP1    ADJ2*   "early"
                NOUN1*  "geometry"
       VERB1*   "was"
       AVP1     AVP2    ADV1*   "very"
                ADV2*   "largely"
       NP2      DETP2   ADJ3*   "a"
                NOUN2*  "list"
                PP1     PP2     PREP1*  "of"
                        NP3     NOUN3*  "rules"
        ?               CONJ1*  "or"
                        NP4     NOUN4*  "formulas"
                 ?              PP3     PREP2   "for"
                                        VERB2*  "finding"
                                        NP5     DETP3   ADJ4*   "the"
                                                NOUN5*  "areas"
                                                PP4     PP5     PREP3*  "of"
                                                        AJP2    ADJ5*   "plane"
                                                        NOUN6*  "figures"
       PUNC1    "."
-------------------------------------------------------------
```

Sentence 8:

```
--- ---------------------------------------------------------
CMPD1  DECL1    NP1     QUANP1  ADJ1*   "many of"
                        DETP1   ADJ2*   "these"
                        NOUN1*  "rules"
                VERB1*  "were"
                AJP1    ADJ3*   "inaccurate"
                        PUNC1   ","
       CONJ1*   CONJ2*  "but"
                PUNC2   ","
       DECL2    PP1     PP2     PREP1*  "in"
                        DETP2   ADJ4*   "the"
                        NOUN2*  "main"
                        PUNC3   ","
        ?       NP2     PRON1*  "they"
                VERB2*  "were"
                AJP2    AVP1    ADV1*   "fairly"
                        ADJ5*   "satisfactory"
       PUNC4    "."
-------------------------------------------------------------
```

c) Figure 3 shows three performance curves as follows.

(i) Figre 3 (A) maps size of the polarized data base as a percentage of the original data base alng the Y axis as the structure of the data base is varied from fully structured to free form along the X axis for a constant data base size.

(ii) Figure 3 (B) maps size of the polarized data base along the Y axis as a function of the size of the original data base along the x axis for a data base of contstant structure.

(iii) Figure 3 (C) shows access time above a certain threshold is independent of the size of the original data base.

PSEUDO-PARSING ALGORITHM

The Pseudo-parsing SWIFT-ANSWER algorithm of this invention comprises the following steps.

a) Separation of a natural language text into senteneces, phrases and words.

b) Separation of Words into non-context and context words.

c) Separation of non-context words into noise words such as pronouns and prepositions and common words such as common words appearing too frequently in a file or data base.

d) Alphabetizing all context words.

e) Mapping frequency and location of all non-context words with respect to source data original files.

NOTE: The above mentioned five steps of the Pseudo parsing algorithm are applied to the natural language unstructred files in the batch mode and then again to the spontaneous convoluted questions in the real time mode.

f) performing mathematical operations such as taking highest common factor and lowest common multiple of the statistical information that correponds to context words in the question.

PSEUDO PARSING SWIFT-ANSWER ALGORITHM

by
(c) S Pal Asija 1989
Patent Attorney & Professional Engineer
7 Woonsocket Ave, Shelton, Conn. 06484
PH: (203)-736-9934 or 736-0774

INTRODUCTION

Pseudo parsing SWIFT-ANSWER algorithm is a subset of patented (4,270,182) SWIFT-ANWER algorithm which is based on the first pure software algorithm patent ever issued anywhere in the world. It is also a federal Trademark registered in the principal register of the Unites States Patent and Trademark Office. It is an acronym which stands for Special Word Index Full Text Alpha Numeric Storage With Easy Retrieval. It is called Pseudo parsing because it deviates substantially from conventional parsing algorithms, even though it accomplishes confusingly similar objectives. It is not a software package nor a key word search system.

NOT A KEYWORD SYSTEM

Some AI(Artificial Intelligence) experts and computational linguists have erroneously perceived this system as a keyword system and therefore have evaluated and crticized it as such. But in reality it is not a keyword system. In fact the system never asks the user for the keywords. Keywords if any are automatically created and managed by this system. It is strictly internal to the system and therefore completely transparent to the user.

Just as in human to human communications in this human/machine communication system also, neither the machine nor the human being is conscious of any keywords. When you ask a human being a question he or she does not ask you for key words, even though the respondant may subconsciously select and use some keywords to properly respond to you. Just as the user does not care what the subconscious of the respondant does, the user also does not care what the internal software of the system does to properly respond to the users communications.

BRIEF DESCRIPTION OF THE DRAWING

a) Figure 1 shows the program flow chart of the SWIFT-ANSWER algorithm.

b) Figure 2 shows the SWIFT-ANSWER Data Flo Diagram.

This unique algorithm creates the illusion of artificial intelligence without even using a conventional "Spell-Check" dictionary let alone spoon feeding rules of parsing, grammar, programming and knowledge of the world. The artificial intelligence if any in this system is inherent in the structure of the algorithm which makes it independent of the knowledge domanin of the data base.

FIVE PAHSE ALGORITHM

a) Installation phase during which the computer asks you a series of questions to get to know your computer environment and your applicational needs including your data bases, so that it can load appropriate operating system, interface modules and drivers.

b) Batch Phase during which the algorithm pre-processes each file specified in phase (a) and extracts certain things from them.

c) Real Time Phase during which the same algorithm is applied to the question as was applied to the files in phase (b).

d) Priority phase during which the prioritizing algorithm ranks all possible answers without going to the data base files.

e) Presentation Phase during which the algorithm presents answers in the order established in the priority phase (a) supra by fetching them from the original files.

POWERFUL ALGORITHM

The power of the algorithm can be traced to the following precepts and principles.

a) All natural languages saturate. As a language saturates and the data base grows larger the probability of a new word appearing goes down and the proababability of a repeat word goes up. The length of the index does not increase although volume of cross-indexing does.

b) Some words such as most pronouns and propositions do not mean much even to people let alone computers

Note: This step in turn generates a lsit of prioritized answer which specified that the best answer based upon the totality of this question begins on disc so and so , sector so and so and is so many bytes long and the second best answer begins on disc so and so, sector so and so and is so many bytes long and so on.

g) applying the user transparent boolean logic to different permutations and combinations of the context words in the question.

NOTE: The pseudo parsing is completed at this step. The remaining steps described in the patent deal with fetching and presenting the right answer in the right format to the user.

UNIQUE ALGORITHM

The algorithm is unique compared to the prior art because it is the only software that responds to a users erroneous spontaneous questions primarily because it performs the following user transparent functions automatically.

a) Automatic LIUs (Logical Information Units)
b) Automatic & Unlimited Dictionary.
c) Automatic Key Words
d) Automatic Boolean Logic
e) Automatic Prioritizing of Answers
f) Automatic Fault Tolerance
g) Automatic Context Determination.
h) Automatic DBM (Data Base Management)

The following functions and features are not automated.

a) Questions & Reframing of questions
b) Interpretation of Answers
c) Specification of the USER environment
d) Creation of the Unstructured Source Data Base
e) Selection of Special Features
f) Inputting of additional context dependant common words and 'synonyms & antonyms' ie searchonyms.

c) HCF (Highest Common Factors) and LCM (Least Common Multiple) of all data across dictionary words which the computer has not been pre-told as meaningless to human beings from the question contain valuable information.

d) Prioritizing sub-algorithm is based on hierarchical relevence of decreasing order. Most relevent being the shortest LIU containing all words of the question most number of times closest together. The algorithm computes and gives starting access location and length.

e) Fault tolerance by left and right shift with and without addition of dummy characters and deletion of characters. The extent being proportional to the size of the word and the degree of fault tolerance specified by the user in phase (a) in para 14 supra.

f) The power of Binary Search and Boolean logic can be made user transparent.

g) Everything people type or put on a machine readable media probably means something to them notwithstanding lively demo given sometimes by the inventor.

h) Synonyms and antonyms both refer to the same context.

i) Von-Neumann serial computer is no match for the parallel processing brain which is not too well understood to begin with..

j) Most words in most data bases and concomitant software are spelled correctly.

k) Its naive to think that knowledge engineers can spoonfeed knowledge of the universe to the computer.

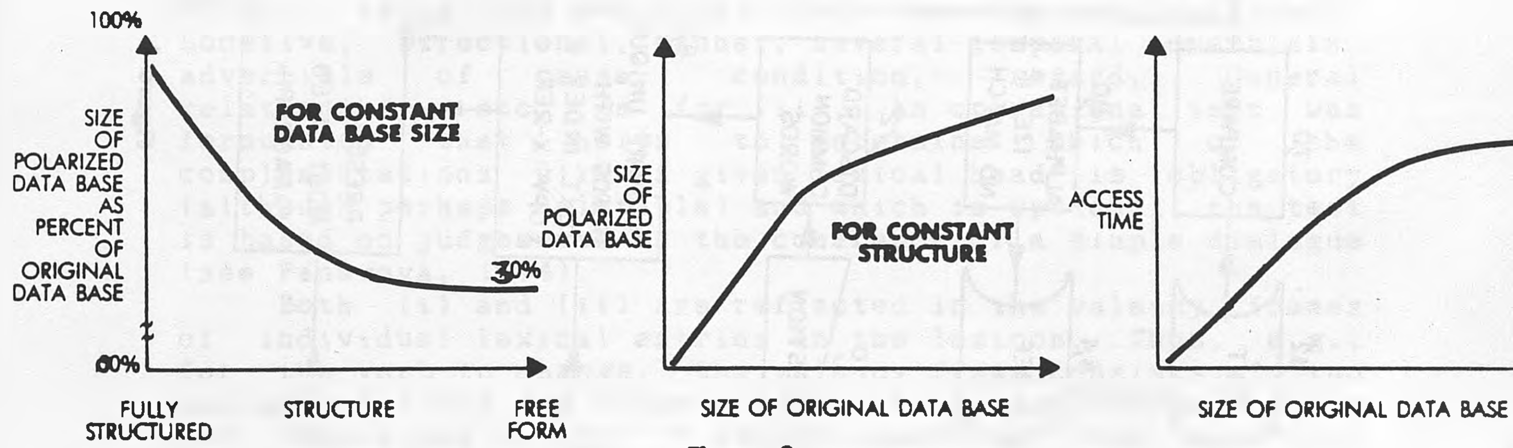


Figure 3

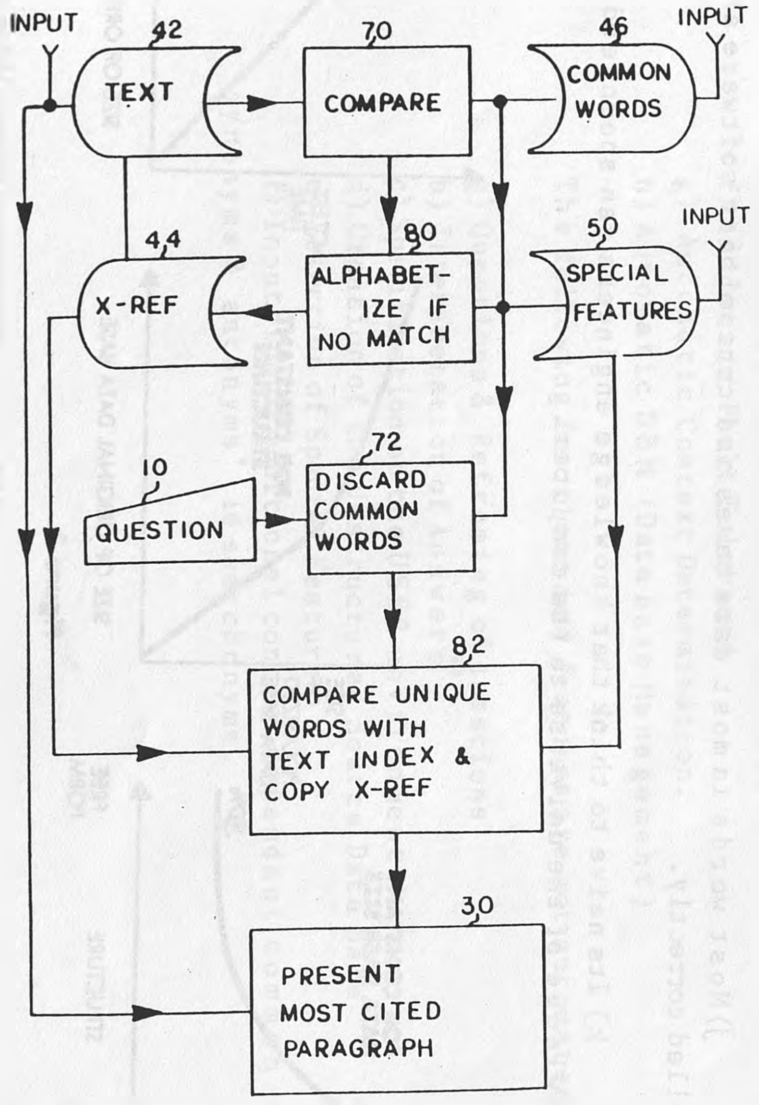


Figure 1

SWIFT-ANSWER DATA FLOW DIAGRAM

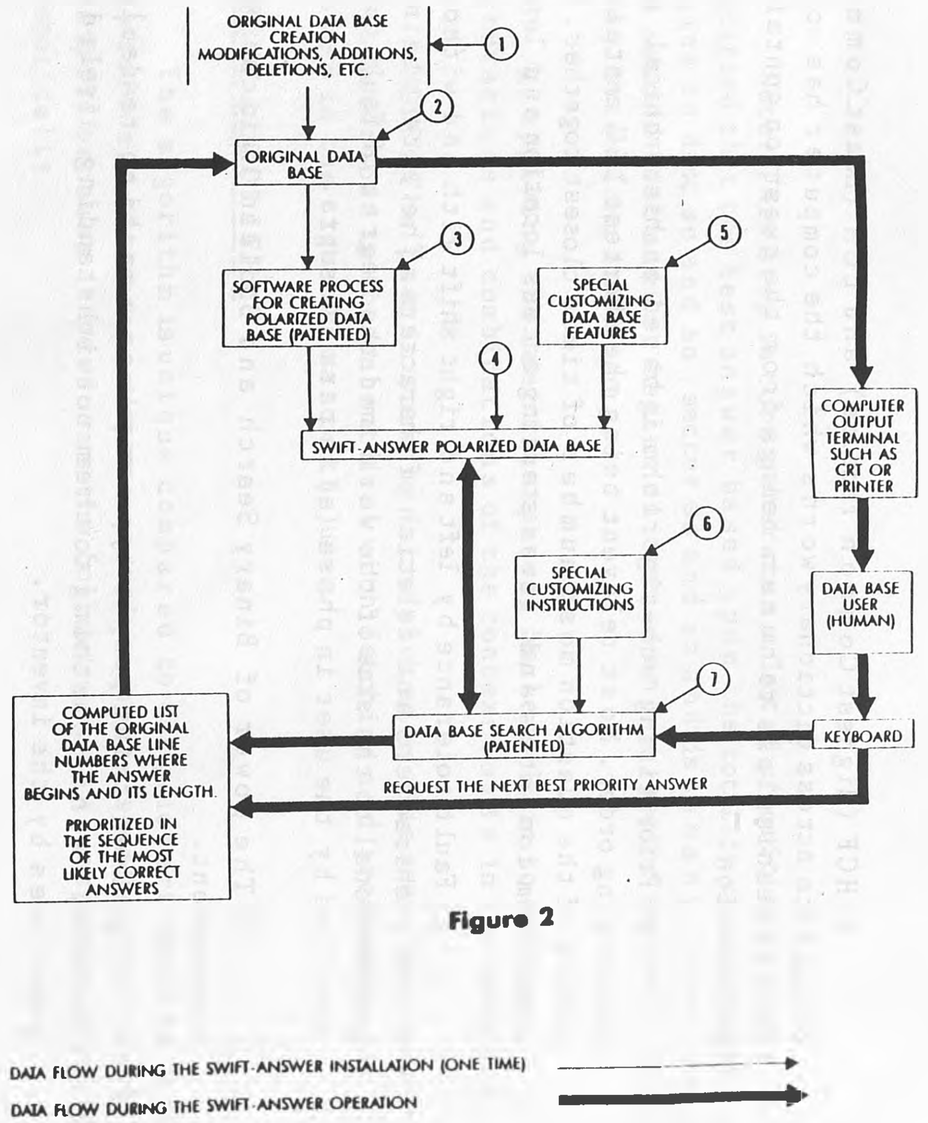

ORIGINAL DATA BASE CREATION MODIFICATIONS, ADDITIONS, DELETIONS, ETC.
1
2
ORIGINAL DATA BASE
3
SOFTWARE PROCESS FOR CREATING POLARIZED DATA BASE (PATENTED)
5
SPECIAL CUSTOMIZING DATA BASE FEATURES
4
SWIFT-ANSWER POLARIZED DATA BASE
COMPUTER OUTPUT TERMINAL SUCH AS CRT OR PRINTER
6
SPECIAL CUSTOMIZING INSTRUCTIONS
DATA BASE USER (HUMAN)
7
DATA BASE SEARCH ALGORITHM (PATENTED)
KEYBOARD
COMPUTED LIST OF THE ORIGINAL DATA BASE LINE NUMBERS WHERE THE ANSWER BEGINS AND ITS LENGTH.
PRIORITIZED IN THE SEQUENCE OF THE MOST LIKELY CORRECT ANSWERS
REQUEST THE NEXT BEST PRIORITY ANSWER
DATA FLOW DURING THE SWIFT-ANSWER INSTALLATION (ONE TIME)
DATA FLOW DURING THE SWIFT-ANSWER OPERATION

Figure 2

which will be clarified in Sect. 3 below) the type of representations characterized till now, we present in Fig. 1 an underlying representation of the sentence (1).

(1) In August, a seminar on parsing technologies will be organized by CMU in Pittsburgh.

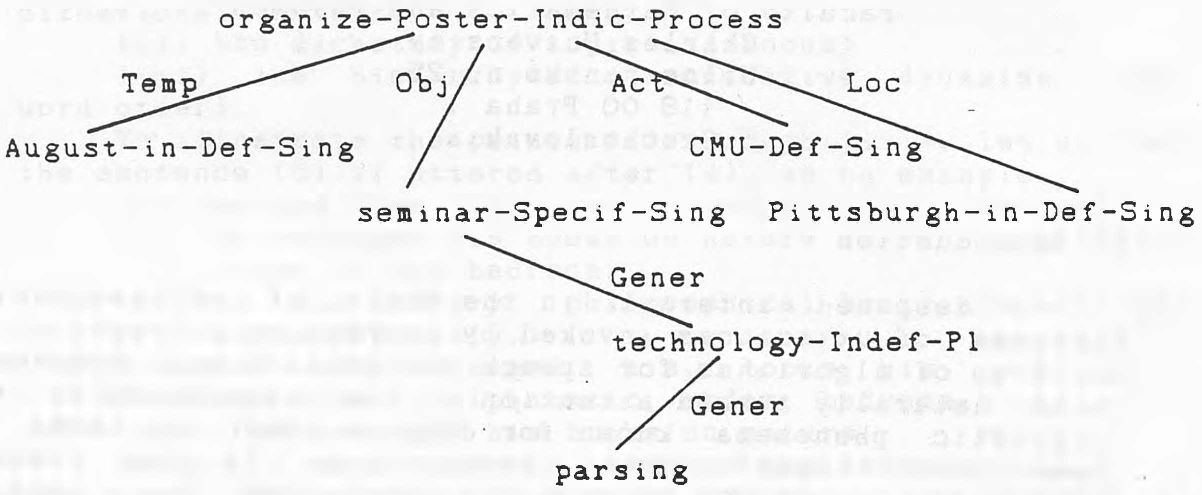


Fig.1

2.2 A dependency oriented account of syntactic(o-semantic) relations offers a rather straightforward way for a formulation of a lexically-driven parsing procedure, since a great part of the relevant information is projected from the frames belonging to the lexical entries of the heads. In the description we subscribe to, valency slots are not understood just in the sense of obligatory or regular kinds of complementation, but are classified into

(i) inner participants (theta roles, each of which can be present at most once with a single head token) and free modifications;

(ii) obligatory and optional; this distinction can be made with both kinds of complementations quoted under (i) depending on the specific heads.

As for (i), five inner participants are being distinguished (for motivation, see Panevová, 1974; Hajičová and Panevová, 1984), namely deep subject (Actor), deep object (Patient, Objective), Addressee, Origin (Source) and Effect; among free modifications, there belong Instrument, Locative, Directional, Manner, several temporal adverbials, adverbials of cause, condition, regard, General relationship, etc. As for (ii), an operational test was formulated that helps to determine which of the complementations with a given lexical head is obligatory (although perhaps deletable) and which is optional; the test is based on judgements on the coherence of a simple dialogue (see Panevová, 1974).

Both (i) and (ii) are reflected in the valency frames of individual lexical entries in the lexicon. Thus, e.g., for the verb *to change*, the valency frame consists of two obligatory slots for Actor and Objective, two optional slots for Source and Effect (*to change something from something into something*) and a list of free modifications, which can be stated once for all the verbs. If one of the free

A Dependency-Based Parser for Topic and Focus

Eva Hajičová
Faculty of Mathematics and Physics
Charles University
Malostranské n. 25
118 00 Praha 1
Czechoslovakia

1. Introduction

A deepened interest in the study of suprasegmental features of utterances invoked by increasing attempts at a build-up of algorithms for speech recognition and synthesis quite naturally turned attention of the researchers to the linguistic phenomena known for decades under the terms of theme-rheme, topic-comment, topic-focus. In the present paper we propose a linguistic procedure for parsing utterances in a "free word order" language, the resulting structure of which is a labelled W-rooted tree that represents (one of) the (literal) meaning(s) of the parsed utterance. Main attention will be paid to the written form of language; however, due regard will be also paid to (at least some of) the suprasegmental features and additional remarks will be made with respect to parsing strategies for written and spoken English.

2. Dependency-Based Output Structures

2.1. The procedure is based on the linguistic theory of functional generative description as proposed by Sgall (cf. Sgall, 1964, 1967; Sgall et al., 1986). The representation of the meaning(s) of the sentence - i.e. the output of the analysis - is a projective rooted tree with the root labelled by a complex symbol of a verb and its daughter nodes by those of the complementations of the verb, i.e. participants (or - in another terminology - the cases, theta-roles, valency), as well as adverbials. The relation between the governor (the verb) and the dependants (its daughter nodes) is a kind of dependency between the two nodes. The complementations of the daughter nodes (and their respective complementations, etc.) are again connected with their governors by an edge labelled by a type of dependency relation. The top-down dimension of the tree thus reflects the structural characteristics of the sentences. The left-to-right dimension represents the deep word order, see Sect. 3 below. Structures with coordination may be then represented by complex dependency structures (no longer of a tree character) with a third dimension added to the tree structure (Plátek, Sgall and Sgall, 1984), or, alternatively, nodes of quite special properties can be added to the tree itself (Močkořová, 1989). Such a type of description can dispense with problems of constituency and "spurious" ambiguity and offers an effective and economic way of representing sentence meaning.

To illustrate (with several simplifications, some of

(3)(a) I do linguistics on *Sundays*.
(3)(b) On Sundays, I do *linguistics*.

In the representations of meaning as characterized in Sect. 2, we distinguish:

(i) contextually bound (CB) and non-bound (CN) nodes, where "contextually" covers both verbal co-text and situational context;

(ii) the dichotomy of topic and focus;

(iii) the hierarchy of communicative dynamism (deep word order).

To illustrate the points (i) through (iii), let us take the sentence (5) if uttered after (4), as na example.

(4) How did John organize the books in his library?
(5) He arranged his books on nature in an alphabetic order in his bedroom.
(In his library, philosophical books are arranged chronologically.)

(i) CB nodes: he, arranged, his, books, his
NB nodes: nature, alphabetic, order, bedroom
(ii) topic: he arranged his books on nature
focus: in an alphabetic order in his bedroom
(iii)deep word order (dots stand for the modifications of the nodes explicitly mentioned)
he - ...books... - arranged - order... - ...bedroom

3.2 The impact of the three aspects (i) through (iii) can be illustrated by the examples (6) through (8), respectively:

(6)(a) (You have just listened to our night concert.) The compositions of Chopin were played by S. *Richter*. We will devote to him also our next *programme*.
him = Richter
(6)(b) (You listen to our night concert.) Chopin's compositions were played by S. *Richter*. We will devote to him also our next *programme*.
him = Chopin
(7)(a) Staff only behind the *counter*.
(7)(b) *Staff* only behind the counter.
(8)(a) It was *John* who talked to few girls in many towns.
(8)(b) It was *John* who talked in many towns to few girls.

The distinction between (a) and (b) in (6) consists in the different preference of anaphoric use of referring expressions if the possible referent is mentioned in the previous context by an NB or a CB element (as *Chopin* in (a) or in (b), respectively); in both cases, the anaphoric elements are in the topic part of the sentence.

The sentence (7)(a) differs from (7)(b) only in that *the counter* is in the focus part of (a), while *staff* is in the focus part of (b), which difference leads to a significant distinction in interpretation: (a) holds true if the members of the staff are (to stay) only behind the counter and nowhere else, while (b) holds true if the space behind the counter is (to be) occupied only by the members of the staff; in contrast to (b), the sentence (a) holds true also if there is somebody else than a member of the staff in that space. In (7), the relevant semantic distinction is rendered by a different placement of the intonation center; in (3) above, the same effect results modifications is obligatory with a certain head (e.g. Directional with *arrive*, Appurtanance with *brother*, Material with *full*), this has to be indicated in the valency frame of the relevant head. 2.3 Dependency can be operationally defined on the basis of endocentricity (cf. Sgall and Panevová, 1989, following Kulagina ,1958). If in a syntactic construction one of two members of the construction can be left out, while the other retains the distributional properties characteristic for the given pair, then the member that can be omitted is considered to depend on the other: e.g., in *Jim read* a *book* the sentence part a *book* can be omitted without the sentence losing its grammaticality; thus, the verb rather than *the book* is the head of the construction. The set of word classes that is determined on independent grounds can then be used to identify the "direction of dependency" in other (exocentric) constructions: though in *Jim bought* a *book* the sentence part a *book* cannot be omitted, *buy* and *read* are assigned a single word class (on independent morphemic and syntactic criteria) and thus it may be postulated that *bought* rather than a *book* is the governor (head) of the construction *bought* a *book*. In a similar vein, a construction such as *Jim read* can be substituted in its syntactic position (as constituting a sentence) by a subjectless verb in many languages (cf. Latin *Pluit*; also in English *It rains* the surface subject *it* has no semantic value: it cannot be freely substituted by a noun or by another pronoun and is equivalent to the Latin ending).

2.4 It is not our objective in the present paper to contrast dependency structures with those of phrase structure grammar. Let us only mention in conclusion of this section, that among the main advantages of dependency trees there is the relatively small number of nodes; the basic syntactic hierarchy can be described without any non-terminal nodes occurring in the representations of sentences,although in their derivations non-terminals can be used without the limitations characteristic of Gaifman's approach to dependency. In addition, if function words are understood as mere grammatical morphemes having no syntactic autonomy, then their values can be treated as indices, i.e. parts of complex labels of nodes, as illustrated in Fig. 1 above. In this way, the component parts of syntactically autonomous units can be represented correctly as having other syntactic properties than the autonomous units themselves, and the representations do not get necessarily complicated.

3. The Semantic Impact of Topic-Focus Articulation

3.1 The topic-focus articulation of an utterance has an impact on the semantic interpretation of the given utterance. It is important to notice that (a) and (b) are two different sentences in (2) as well as in (3), though the semantic difference is much more important in (3) than in (2). With (2) the two sets of propositions to which the two sentences correspond assign the value "true" to the same subset of possible worlds, which is not the case with (3)[1]. (The intonation center is denoted by italics.).

(2)(a) Mother is *coming*.

(2)(b) *Mother* is coming.

order is determined first of all by the scale of communicative dynamism, it is evident that the former cases in (A) and (B) do not present so many difficulties for the recognition procedure as the latter cases do.

A written "sentence" corresponds, in general, to several spoken sentences which differ in the placement of their intonation center, cf., e.g., ex. (3) above. In languages with the "free" word order this fact does not bring about serious complications with written technical texts, since there is a strong tendency to arrange the sentences in such texts so that the intonation center falls on the last word of the sentence (if this word is not enclitical).

4.31 A procedure for the identification of topic and focus in Czech written texts can then be formulated as follows (we use the term 'complementation' or 'sentence part' to denote a subtree occupying the position of a participant or free modification as discussed in Sect. 2 above):

(i)(a) If the verb is the last word of the surface shape of the sentence (SS), it always belongs to the focus.

(i)(b) If the verb is not the last word of the SS, it belongs either to the topic, or to the focus.

Note: The ambiguity accounted for by the rule (i)(b) can be partially resolved (esp. for the purposes of the practical systems) on the basis of the features of the verb in the preceding sentence: if the verb of the analyzed sentence is identical with the verb of the preceding sentence, or if a relation of synonymy or meaning inclusion holds between the two verbs, then V belongs to the topic. Also, a semantically weak, general verb such as *to be*, *to become*, *to carry out*, most often can be understood as belonging to the topic. In other cases the primary position of the verb is in the focus.

(ii) The complementations preceding the verb are included in the topic.

(iii) As for the complementations following the verb, the boundary between topic (to the left) and focus (to the right) may be drawn between any two complementations, provided that those belonging to the focus are arranged in the surface word order in accordance with the systemic ordering.

(iv) If the sentence contains a rhematizer (such as *even*, *also*, *only*), then in the primary case the complementation following the rhematizer belongs to the focus and the rest of the sentence belongs to the topic.

Note. This concerns such sentences as *Here even* a *device of the first type can be used.*; in a secondary case the rhematizer may occur in the topic,e.g., if it together with the sentence part in its scope is repeated from the preceding co-text.

4.32 Similar regularities hold for the analysis of spoken sentences with normal intonation. However, if a non-final complementation carries the intonation center (IC), then

(a) the bearer of the IC belongs to the focus and all the complementations standing after IC belong to the topic;

(b) rules (ii) and (iii) apply for the elements from a word order change.

The clefting in (8) univocally points to *John* as the focus of the sentence, the rest being its topic; the two sentences (a) and (b) differ as to the (deep) order of Locative and Addressee. This distinction again has an important semantic impact: with (a), there was a group of girls who were few, and the same group was talked to in many towns, while with (b) John talked in each of the many towns with (maybe) a (different) small group of girls. This difference need not be reflected in the surface word order: the same effect is reached by a shift of intonation center, see (9)(a) and (b).

(9)(a) John talked to few girls in many *towns*.

(9)(b) John talked to few *girls* in many towns.

4. Parsing Procedure for Topic and Focus

4.1 The proposed procedure of automatic identification of topic and focus is based on two rather strong hypotheses:
(i) the boundary between topic and focus is always placed so that there is such an item A in the representation of meaning that every item of this representation that is less (more) dynamic than A belongs to the topic (focus); in the primary case the verb meets the condition on A and is itself included in the focus;
(ii) the grammar of the particular language determines an ordering of the kinds of complementations (dependency relations) of the verb, of the noun, etc., called 'systemic ordering' (SO). The deep word order within focus is determined by this ordering; with sentences comprising contextually bound items, these items stand to the left in the hierarchy of communicative dynamism and their order (with respect to their governors) is determined by other factors. An examination of Czech in comparison with English and several other languages has led to the conclusion that the SO of some of the main complementations is identical for many languages, having the form Actor - Addressee - Objective, As for Instrument, Origin, Locative, it seems that English differs from Czech in that these three complementations follow Objective in English, though they precede it in Czech. It need not be surprising that languages differ in such semantically relevant details of their grammatical structures as those concerning SO - similarly as they appear to differ in the semantics of verbal aspects, of the articles, of dual number, etc.

We assume further that every sentence has a focus, since otherwise it would convey no information relevant for communication; however, there are sentences without topic.

4.2 For an automatic recognition of topic, focus and the degrees of CD, two points are crucial:

(A) Either the input is a spoken discourse (and the recognition procedure includes an acoustic analysis), or written (printed) texts are analyzed.

(B) Either the input language has (a considerable degree of) the so-called free word order (as in Czech, Russian, Latin, Warlpiri) or its word order is determined mainly by the grammatical relations (as in English, French).

Since written texts usually do not indicate the position of intonation center and since the "free" word on automatic identification of topic and focus in spoken utterances only the position of the intonation center; a question naturally arises whether other features of intonation patterns such as tune and phrasing (in terms of Pierrehumbert) can help as clues for sentence disambiguation as for its topic and focus. Schmerling (1971) was the first, to our knowledge, to propose that the different interpretations of Chomsky's 'range of permissible focus' (which basically corresponds to our 'deep word order', see Hajičová and Sgall, 1975) are rendered on the surface by different intonation patterns; most recently, Pierrehumbert and Hirschberg (1989, Note 5) express a suspicion that the accented word in such cases (within an NP) need not have the same prominence in all the interpretations; they also admit that similar constraints on the accenting of parts of a VP are even less understood.

5. Parsing Sentences in a Text

To resolve some complicated issues such as the ambiguity of pronominal reference, a whole co-text rather than a single sentence should be taken into account. Several heuristics have been proposed to solve this problem; e.g., Hobbs (1976) specifies as a common heuristics for pronominal resolution the determination of the antecedent on the basis of the hearer's preference of the subject NP to an NP in the object position (in a similar vein, Sidner ,1981, in her basic rule tests first the possibility of co-specification with what she calls 'actor focus'), the other strategy including inferencing and factual knowledge. Following up our investigation of the hierarchy of activation of items of the stock of knowledge shared by the speaker and the hearer (see Hajičová and Vrbová, 1982; Hajičová, 1987; Hoskovec, 1989; Hajičová and Hoskovec, 1989), we maintain that also this hierarchy should be registered for parsing sentences in a text. We propose to use a partially ordered storage space, reflecting the changes of the activation (prominence) of the elements of the information shared by the speaker and the hearer. The rules assigning the degrees of activation after each utterance take into account the following factors:
(i) whether the given item was mentioned in the topic part or in the focus part of the previous utterance: mentioning in the focus part gives the item the highest prominence, mentioning in the topic part is assumed to assign a one degree lower activation to the given item;
(ii) grammatical means by which the given item is rendered in the surface shape of the utterance: mentioning by means of a (weak) pronoun gives a lower prominence than mentioning by means of a noun;
(iii) association with the items explicitly mentioned in the utterance: items which are associated with the items explicitly mentioned in the preceding utterance get a certain level of prominence, though lower than those mentioned explicitly; it is assumed that the association relations can be classified according to the 'closeness' of the items in question so that some types of associations receive higher degrees of activation than others (e.g., is-a relation is 'closer' in this sense than the part-of relation);[3]
(iv) non-mentioning of a previously mentioned item: an item

standing before the bearer of the intonation center;
(c) the rule (i)(b) is applied to the verb (if it does not carry the IC).

4.33 As for the identification of topic and focus in an English written sentence, the situation is more complicated due to the fact that the surface word order is to a great extent determined by rules of grammar, so that intonation plays a more substantial role and the written form of the sentence displays much richer ambiguity. For English texts from polytechnical and scientific domains the rules stated for Czech in Sect. 4.31 should be modified in the following ways:
(i)(a) holds, if the surface subject of the sentence is a definite NP; if the subject has an indefinite article, then it mostly belongs to the focus, and the verb to the topic; however, marginal cases with both subject and verb in the focus, or with subject (though indefinite) in the topic and the verb in the focus are not excluded;[2]
(i)(b) holds, including the rules of thumb contained in the note;
(ii) holds, only the surface subject and a temporal adverbial can belong to the focus, if they do not have the form of definite NP's;
(iii) holds, with the following modifications:
(a) if the rightmost complementation is a local or temporal complementation, then it should be checked whether its lexical meaning is specific (its head being a proper name, a narrower term, or a term not belonging to the subject domain of the given text) or general (a pronoun, a broader term); in the former case it is probable that such a modification bears the IC and belongs to the focus, while in the latter case it rather belongs to the topic;
(b) if the verb is followed by more than one complementation and if the sentence final position is occupied by a definite NP or a pronoun, this rightmost complementation probably is not the bearer of IC and it thusfinite NP or a pronoun, this rightmost complementation probably is not the bearer of IC and it thus belongs to the topic;
(c) if (a) or (b) apply, then it is also checked which pair of complementations disagreeing in their word order with their places under systemic ordering is closest (from the left) to IC (i.e. to the end of the focus); the boundary between the (left-hand part of the) topic and the focus can then be drawn between any two complementations beginning with the given pair;
(iv) holds.

4.34 If a spoken sentence of English is analyzed, the position of IC can be determined more safely, so that it is easier to identify the end of the focus than with written sentences and the modifications to rule (iii) are no longer necessary. The procedure can be based on the regularities stated in Sect. 4.32.

Up to now, we have taken into account in our discussion

Hajičová, E. and P. Sgall (1975), Topic and Focus in Transformational Grammar, Papers in Linguistics 8, 3-58.

Hajičová, E. and J. Vrbová (1982), On the Role of Hierarchy of Activation in the Process of Natural Language Understanding, in Horecký (1982), 107-113.

Hobbs, J. R. (1976), Pronoun Resolution. Rep. 76-1, Dept. of Computer Science, City College, City Univ. of New York.

Horecký, J., ed. (1982), Coling 82 - Proceedings of the Ninth Int. Conf. on Computational Linguistics, Prague - Amsterdam.

Hoskovec, T. (1989), Modelling a Pragmatical Background of Discourse. In: AI '89, Prague, 289-296.

Kulagina, O. S. (1958), Ob odnom sposobe opredelenija grammatičeskich ponjatij, Problemy kibernetiki 1, 203-214.

Močkořová, Z. (1989), Generalizivané podkladové závislostni struktury (Generalozed Underlying Dependency Structures), diploma theses

Panevová, J. (1974), On Verbal Frames in Functional Generative Description I, Prague Bulletin of Mathematical Linguistics 22, 3-40; II, 23 (1975), 17-52.

Pierrehumbert J. and J. Hirschberg (1989), The Meaning of Intonational Contours in the Interpretation of Discourse.

Plátek M., Sgall, J. and P. Sgall (1984), A Dependency Base for a Linguistic Description. In: Sgall (1984), 63-97.

Schmerling ,S. F. (1971), Presupposition and the Notion of Normal Stress. In: Papers from the Seventh Regional Meeting. Chicago Linguistic Society , 242-253.

Sgall, P. (1964), Generative Beschreibung und die Ebenen des Sprachsystems, presented at the Second International Symposium in Magdeburg, printed in Zeichen und System der Sprache III, 1966, Berlin, 225-239.

Sgall, P. (1967), Functional Sentence Perspective in a Generative Description. In: Prague Studies in Mathematical Linguistics 2, 203-225.

Sgall, P., ed. (1984), Contributions to Functional Syntax, Semantics, and Language Comprehension, Amsterdam - Prague.

Sgall, P., Hajičová, E. and J. Panevová (1986), The Meaning of the Sentence in Its Semantic and Pragmatic Aspects, Dordrecht - Prague.

Sgall, P. and J. Panevová (1989), Dependency Syntax - A Challenge, Linguistics 15.

Sidner, C. L. (1981), Focusing for Interpretation of Pronouns. American Journal of Computational Linguistics 7, 217-231.

that has been introduced into the activated part of the stock of shared knowledge but is not mentioned in the subsequent utterances loses step by step its prominence;
(v) not only the immediate degree of activation after the given utternace is relevant for the assignment of reference but also the sequence of degrees of salience from the whole preceding part of the text; thus if an item is being mentioned subsequently for several times in the topic of the sentence, its salience is maintained on a high level and it is more likely an antecedent for pronominal reference than an item that appeared in the focus part (with no prominence history) and received thus the highest degree of activation.

6. Concluding Remarks

Since even in such languages as English or French, surface word order corresponds to the scale of communicative dynamism to a high degree (although such grammatical means as passivization, or the inversion of *make out of* to *make into* , etc., often are necessary here to achieve this correspondence), it is useful in automatic language processing to reflect the word order of the input at least in its surface form. If the effects of the known surface rules on the verb placement, on the position of adjectives, genitives[4], etc., before (or after) nouns, and so on, are handled, and if the items mentioned in the preceding utterance are stored (to help decide which expressions are contextually bound), then the results may be satisfactory.

Notes.

1 With (2) as well as with (3) the presuppositions triggered by (a) and (b) differ, so that different subsets of possible worlds get the value 'false'; e.g., (2)(b) differs from (2)(a) in presupposing that someone is coming.

2 For the solution of such cases, it again is useful to "remember" the lexical units contained in the preceding utterance, cf. the Note to (i)(b) in Sect. 4.31 above.

3 It is more exact to understand the association relationships in terms of natural language inferencing (concerning the occurrence of a single associated item) than in terms of the activation of the whole set of items associated with an occurrence of a possible 'antecedent'.

4 This has been done, at least to a certain degree, in the experimental systems of English-to-Czech and Czech-to-Russian translation, implemented in Prague.

References

Hajičová, E. (1987), Focussing - A Meeting Point of Linguistics and Artificial Intelligence. In: Artificial Intelligence II - Methodology, Systems, Applications (ed. by Ph. Jorrand and V. Sgurev), Amsterdam, 311-322.

Hajičová, E. and T. Hoskovec (1989), On Some Aspects of Discourse Modelling. In: Fifth Int. Conference on Artificial Intelligence and Information-Control Systems of Robots (eds. I. Plander and J. Mikloško), Amsterdam.

Hajičová, E. and J. Panevová (1984), Valency (Case) Frames of Verbs. In: Sgall (1984), 147-188.

incorporating all of the above. We describe the parser in the sections that follow.

1.1 EXISTING PARERS AND OUR APPROACH

All the implementations of GPSG reported in the literature use a rather straight forward approach of first expanding the entire rule set by using the available metarules, in the process augmenting the set of rules, and finally the normal context free parser is run on this new set of rules.

Thus there are two basic steps involved :-

1. Rule expansion using the available metarules
2. Actual parsing using the expanded set of rules.

It should be noted that in such an implementation one does not need to bother about the metarules after the first stage.

An inherent drawback with this approach is that if the initial set of rules is of sizeable cardinality, then a number of rules may get added to the set, (a large number of these rules may never get used during the actual parse of a sentence), thus not only causing memory storage problems, but also slowing down the system considerably.

The main motivation of this paper is to describe a method for parsing GPSG without initial expansion (i.e. our implementation expands metarules as and when necessary). Further, in our implementation we have assumed in ID-LP format for the rules, thus making them more compact.

Because of the above reasons, it has become necessary to make some changes to an ordinary context-free parsing algorithm to suit our requirements (i.e. to incorporate dynamic expansion and the ID-LP format of rules).

2 PARSING ALGORITHM

The essential characteristics of our approach towards a solution of the problem has been listed over the next few pages.

2.1 DYNAMIC STRUCTURE OF RULES

As has been mentioned earlier, our implementation gets new rules from old ones as the parsing proceeds. Under such a situation, it becomes necessary to suspend parsing temporarily, only to return to it after a rule of the appropriate type has been generated by expanding sing one or more metarules some appropriate rule from the already available set.

At this stage a decision has to be taken as to whether the rule which was recently derived should be stored for further use or should be discarded. Here the choice should be guided by the

Parsing Generalized Phrase Structure Grammar with Dynamic Expansion

Navin Budhiraja
Subrata Mitra
Harish Karnick
Rajeev Sangal

Department of Computer Science and Engineering
Indian Institute of Technology Kanpur
Kanpur 208 016 India

SUMMARY

A parser is described here based on the Cocke-Young-Kassami algorithm which uses immediate dominance and linear precedence rules together with various feature inheritance conventions. The meta rules in the grammar are not applied beforehand but only when needed. This ensures that the rule set is kept to a minimum. At the same time, determining what rule to expand by applying which meta-rule is done in an efficient manner using the meta-rule reference table. Since this table is generated during "compilation" stage, its generation does not add to parsing time.

1 INTRODUCTION

GPSG as introduced by Gazdar et.al. gives a formalism to parse natural languages assuming they are context free. The phrase structure rules are like the normal CFG rules, except that features are added to the categories. These features are used by Feature Co-occurence Restrictions, Feature specification Defaults, Head Feature Convention, Foot Feature Principle and Control Agreement Principle, during parsing.

The second important feature of GPSG, and towards which this paper is mainly directed, is the metarule. A major problem of a complete natural language grammar is its size, which causes difficulties as far as memory requirements and efficiency of any practical parser are concerned. GPSG tries to overcome this, partly, by keeping the rule set to the minimum. In addition to the minimal set of rules, it has certain metagrammatical structures to generate rules from the previously defined minimal set. Thus the number of rules at any time are the minimum possible, reducing the search time of the parser. In addition, this captures certain linguistic generalisations (e.g. active-passive).

Lastly GPSG goes to the thematic representation directly from the c-structure (in contrast to other formalisms like LFG). The IL formula is built up as parsing proceeds.

Our endeavour is, thus, to build a natural language parser

(b) Get the position of the current rule in the rule table corresponding to I1. Let this be I2.
(c) Get the position of the current metarule m in the metarule list. Let this be M1.

3.4 Now append the triplet (I1 I2 M1) to the contents of the meta reference table entry pointed to be the index found in step 3.2 above.

The above takes care of cases where one level of expansion of metarules is sufficient. But in general a rule could be expanded successively more than once by the same or by different metarules before it can be used for parsing. Thus it is necessary to extend the meta-reference entries to handle the problem.

Basically a triplet (I1 I2 M1) as defined above corresponds to a rule which is produced by applying metarule M1 to the I2th. entry in the I1th. sub-structure of the rule table. Let the resultant rule be R1. Now R1 may expand some metarule whose position is M2 to produce a rule R2, and so on, until at some stage we get a rule Rn which is not meta expandable any further. The termination is guaranteed because GPSG is equivalent in power to CFG.

In the compilation stage we must now make entries in the meta-reference table for each of R1... Rn, because any of them may be necessary during parsing. This can be done as follows:-

```
    R1 is the rule corresponding to (I1 I2 M1)

    For i:=1 to n do
       Ri  is Mi applied on R(i-1),
     Get an index to the meta-reference table using rhs
                                 categories of Ri
     To this entry append ((I1 I2 M1) M2...Mi)
```

Here M1,M2...,Mi gives the successive position in the metarule list of the metarules to be applied.

An example will clarify the situation:
consider a metarule of the form
(VP--> W NP)====>(VP--->W (optional(PP[by]))).
where W is any set of categories.
This generates the passive counterparts of active sentences.

Now if we have a rule of the type
0..VP--> V NP NP,
/* The features etc. have been omitted for simplicity */
after first expansion we shall get two rules, namely:

1..VP--> V NP,
2..VP--> V NP PP.

Further, because of the given structure of the intermediate rules and the metarules under consideration, a second expansion is possible. Consider the rule VP--> V NP PP (rule 2 above). When the above metarule is expanded using this rule, we get the relative gain in time by storing the rule (as opposed to re-expanding) against the storage overhead. The type of sentences to be parsed may also play a role in this decision. For example, it may be worthwhile to store the rule which gets generated during parsing. Equivalently the other approach may be tried.

For this type of implementation, metarules become in important part of parsing. Further justification on this issue is given in the next section.

2.2 TABLE BUILDING

We have seen in the previous section that an important aspect of our parser implementation is the generation of rules at an intermediate stage.

A native way to tackle the problem would be to go over the entire set of rules and metarules when a failure occurs, and try each metarule-rule combination to find one which produces a rule of the required type, and then carry on with the parser. But this will obviously be highly inefficient.

To cut down the time of generating rules and trying them out, a table can be constructed to help us select the metarule-rule combination. This is what is done.

In the first stage (called the metarule compilation stage) we go over all the rule-metarule combinations to build up a reference table which can be consulted by the parser (during the second stage) to get the required rules efficiently. Compilation is a one time job and, therefore, does not affect the complexity of the actual parser.

The rule set can be structured for faster access to the relevant rules. In our current implementation we have structured the rule set on the basis of the number of categories (non-terminals) on the right-hand-side (rhs henceforth).

The metarule reference table is built up in the compilation stage as follows :-

```
 1. For each rule r do the following.
 2. Store the rule in the appropriate entry of a new table
    called the RULE TABLE, which is the one that the parser
    refers to. (For our case store it with all other rules
    which have the same number of rhs categories).
 3. For each metarule m that can be applied on r do
    3.1 apply m on r yielding a new rule s
    3.2 hash the rhs categories of the newly produced rule s to
        get an index into another table called the META-REFERENCE
        table.
    3.3 Build up a meta reference entry as follows :-
     (a) Get index (here number of rhs categories) of the input
         rule r. Let this be I1.
```

1) make the algorithm work for an ID/LP grammar,
2) make the algorithm work for grammars not in Chomsky Normal Form (CNF),
3) allow for meta-rule expansion during parsing

We discuss these one by one.

1) In order to handle ID/LP grammars, we have to just look for a rule with the required nonterminals on the right hand side, with no importance attached to the order (except of course,for precedence relations)

2)In order to account for grammars which are not in CNF, we had to increase the nesting of the loops which handle rules of the form

    A----> B1 B2

in the CYK algorithm. The loop depth should now be (k-1) in order to handle rules like

    A----> B1 B2. ..Bk

(See algorithm extract given below)

3) In order to get new rules from the old, we have to make some additions to the CYK algorithm. A part of the algorithm is given below :-

```
/*
     The algorithm to handle grammars not in CNF and to
     allow for metarule application during parsing is
     shown below. This handles all rules which have k
     nonterminals on the right hand side.
*/
procedure length_k(i,j) ;
begin
     for a1 := 1 to j-k+1 do
        for b1 := 1 to j-a1-k+2 do
           for c1 := 1 to j-a1-b1-k+3 do
                  .
                  .
                  .
              for j1 := 1 to j-a1-b1-c1...-i1-k+j do
                   RULESET := RULESET U { new rules obtained by
                                          expanding the metarules
                                          as required by the
                                          parser }
                   /* it is in the above line that we get the new
                      set of rules as demanded by the parser */
                    CYK(i,j) :=
                                 CYK(i,j) U {A | A---->B1B2...Bk
                                    is a production, and
                                       B1 is in CYK(i,a1),
                                       B2 is in CYK(i+a1,b1)
                                          .
                                          .
                                          .
                                       Bk is in CYK(i+a1+b1...+j1,
```

following pair of rules:

3.. VP--> V PP
4.. VP --> V PP PP

Similarly the rule VP --> V NP (rule 1 above) will generate two rules of the form:

5..VP --> V
6..VP --> V PP

Now since none of the newly generated rules are meta-expandable the process will stop. The meta referencea table entries will be of the following nature :

/* Let us assume that there is just this one metarule in the meta-list, and that the initial rule (rule 0 above) is the only one present in the rule array corresponding to length of rhs three,i.e,

I1 is 3
I2 is 1, and
M1 is 1

*/ (a) Corresponding to rhs <V,NP,PP> and <V,NP> we shall have entries of the type ((3 1 1)),while
(b) Corresponding to any other possible collection of rhs categories, for example <V,PP>, the entry will look like ((3 1 1) 1), which incorporates two levels of meta expansion.

A point of importance is that since one expansion of a metarule can produce more than one output rule (e.g. rules (1) & (2) from rule (0) above) the meta expander must check for category names before returning the generated rule.

For example if meta expansion is called with parameters (V,NP)in the above situation, then only rule (1) should be returned, the other has to be discarded .

Another change could be incorporated regarding the structuring of the set of input rules. One can use a hashing technique similar to the one used for storing meta reference entries. Thus, rules would be stored not by the number of rhs categories, but hashed according to the categories present in the rhs. This would make rule access at parse time much faster and direct because during a bottom-up parse we have to reduce a given set of rhs categories into the corresponding left hand side. This would however mean keeping more entries in the rule table.

2.3 THE PARSING ALGORITHM

The parsing algorithm we have used is the well known Cocke-Young-Kassami (CYK) algorithm, with a few modifications. The differences are for the following requirements. We have to :

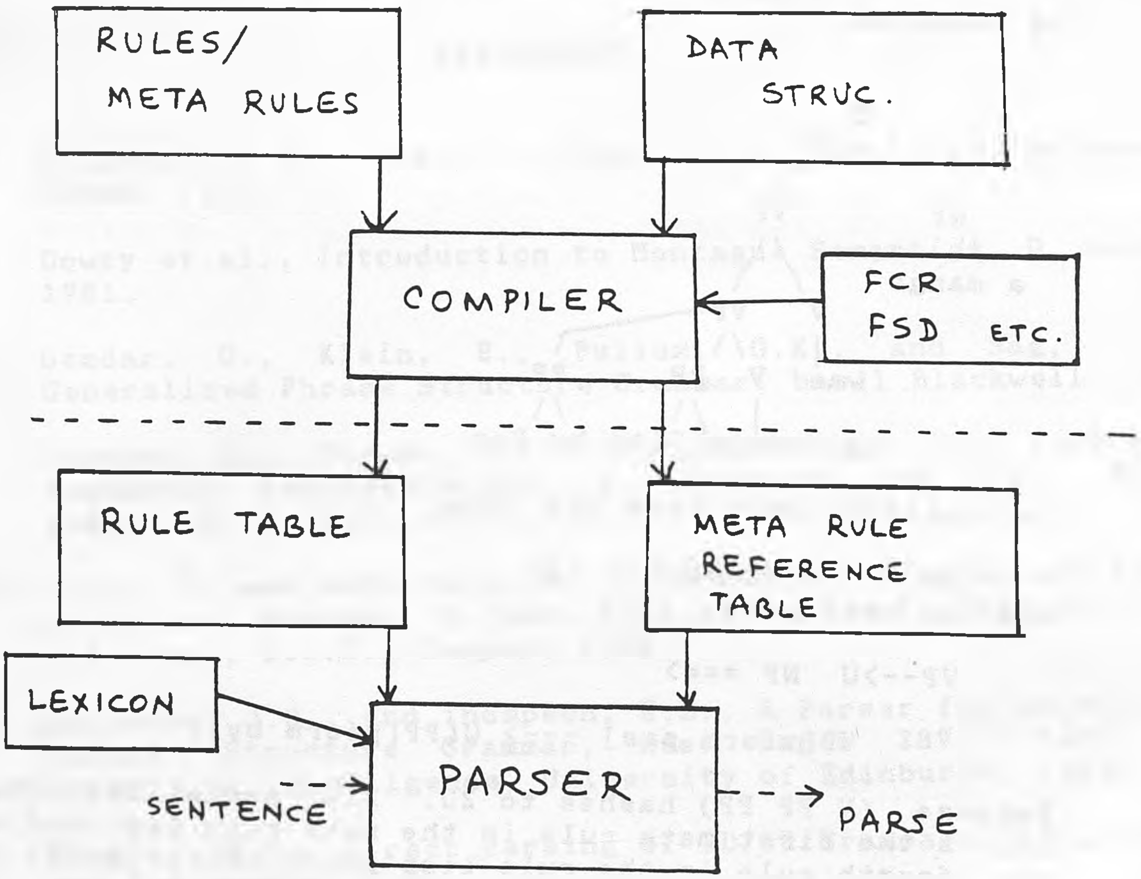
have three right hand sides. Now when the parser sees that it requires a rule containing V PP PP (as in (1)) it makes the following call

```
(return-meta-expanded '(V PP PP))
```

The triplet that is obtained from the table lookup is (3 4 1) which calls the meta rule expander to apply (3) to (2) which returns (1). The parser then continues its normal course after adding the generated rule to the appropriate rule-list.

2.4 THE SYSTEM STRUCTURE

The block structure of the compiler and parser is given below with the dotted-line separating the two. The part above the dotted line is done only once when the grammar is "fed" in. First the rules specified by the grammar designer are stored in an appropriate data structure. The compiler then applies the various feature restriction principles to this rule set (similar to the Edinburgh approach described in Philips (86)), makes the feature bindings and then indexes them according to the number of categories on the right hand sides. In addition it also creates the all important Meta-rule reference table. Both these tables are then passed to the parser which then, using the lexicon, works as described before.



```
                                                  j-a1-b1...-j1)  }
          end;
```

As can be seen from the algorithm extract given above, the meta-rules are expanded here. Once we have the required RHS (B1,B2...Bk), it is hashed to a value in the meta-rule reference table which returns us a triplet of the form (I1 I2 M1) where

> I1 stands for the index of the rule-table i.e the table which contains all the rules according to the number of right hand sides they have.
>
> I2 stands for the number of the rule in the I1 entry of the rule index table
>
> M1 stands for the number of the metarule which needs to be used.

We discuss an example to illustrtate the algorithm.

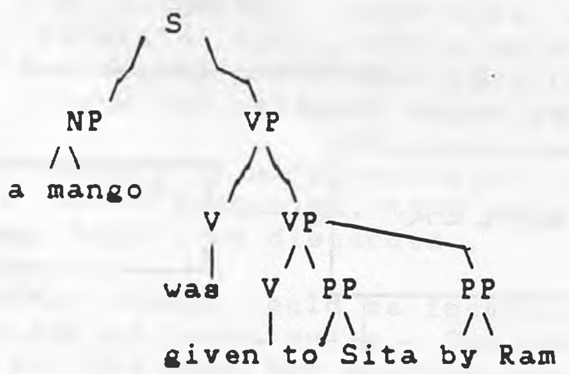
Example: In the parsing of the sentence

    A mango was given to Sita by Ram

the rule

```
     VP--> V PP[pform to ] PP[pform by ] --- (1)
```

is required.



Initially we only have the rule

```
     VP --> V NP PP --- (2)
```

and the meta rule

```
     VP-->W  NP ===>

          VP[vform pas] ---> W(PP[pform by]) --- (3)
```

Suppose (V PP PP) hashes to 20. Also assume that the meta rule (3) is the first meta rule in the meta rule list and rule (2) is the fourth rule in the rule list which contains all rules which beforehand but only when needed. This ensures that the rule set is kept to a minimum. At the same time, determining what rule to expand by applying which meta-rule is done in an efficient manner using the meta-rule reference table.

The Cocke-Young-Kassami algorithm has been modified to work on the context free grammar without converting it to Chomsky Normal form. Conversion would lead to an increase in number of rules, and would also affect the dominance relationships. The modified algorithm continues to be a polynomial time algorithm on the length of the input sentence.

The implementation of the parser has been tested with a small grammar and with a small number of meta rules. To get performance figures, it needs to be tested more extensively. Experiments can also be conducted regarding when the generated rules should be stored for future use and when they should be discarded.

Our parser, at the moment, does not have the Kleene Closure facility to handle conjunctive/disjunctive sentences. It is a simple matter, however, to add this.

ACKNOWLEDGEMENT

Insightful Suggestions by Vineet Chaitanya and B.N. Patnaik thro--ughout the course of this work are gratefully acknowledged.

REFERENCES

[1] Allwood et.al., Logic in Linguistics, Cambridge University Press, 1977.

[2] Dowty et.al., Introduction to Montague Semantics, D. Reidel, 1981.

[3] Gazdar, G., Klein, E., Pullum, G.K., and Sag, I.A., Generalized Phrase Structure Grammar, basil Blackwell, 1985.

[4] Gazdar, G., Phrase Structure Grammar, in The Nature of Syntactic Representation, P. Jacobson and G.K. Pullum (eds.), D. Reidel, 1982.

[5] Mitra, S. and Budhiraja, N., A Parser for Generalized Phrase Structure Grammar, B.Tech. thesis, Dept. of Computer Sc. and Engg., I.I.T., Kanpur, 1988.

[6] Phillips, J.D., and Thompson, H.S., A Parser for Generalized Phrase Structure Grammar, Res. Paper 289, Dept. of Artificial Intelligence, University of Edinburgh, 1986.

[7] Shieber, S.M., Direct Parsing of ID/LP Grammars, Linguistics and Philosophy, 7,2.

2.5 HANDLING FEATURE RESTRICTIONS

The heart of GPSG is made up of the set of <feature, feature-value> pairs associated with every syntactic category. GPSG introduces some rule and conventions to associate values with these features in required manner.

Some of these restrictions should cause values to be given to features during actual parsing of a sentence, while others should pass up the tree certain feature values which get instantiated at parse time to ensure a valid parse.

Our implementation handles such problems at the compilation stage by considering fully expanded categories, where feature values corresponding to a particular feature which is as yet uninstantiated are bound to a unique variable, and the variable is shared among all instances of the same feature in the rule, which have to be bound together. This approach is similar to the Edinburgh parser.

Later, during the actual parse, if any variable gets bound to a value, then all other instances of the same variable in the rule also get the same value. Any mismatch leads to rejection.

For example, in the rule

A---> B1, B2...Bk

the variable valued features in A get bound to their values as instantiated in B1,B2...Bk. We are assuming that the RHS of a rule is fully instantiated during the parse i.e once a category is added to the CYK table, no more features are added to it. This approach has forced us to use multiple entries in the lexicon.

For example, the entry for 'the' contains two entries, one each for singular and plural respectively.

3 SEMANTICS

The IL formula for the input sentence is built up as the parsing proceeds. Each node in the parse tree being built contains the IL formula of the node. Using the type information and the Semantic Interpretation Schema, the IL formula of the mother is built up from the IL formulae of its children. Finally the node S (the start symbol) contains the IL formula of the input sentence. After parsing finishes, transformations as required by GPSG (e.g. the passive-active transformation, paraphrases etc.) are applied to the IL formula of the root.

4 CONCLUSIONS

The parser described here uses immediate dominance and linear precedence rules together with various feature inheritance conventions. The meta rules in the grammar are not applied

NOTES:

NOTES:

NOTES: