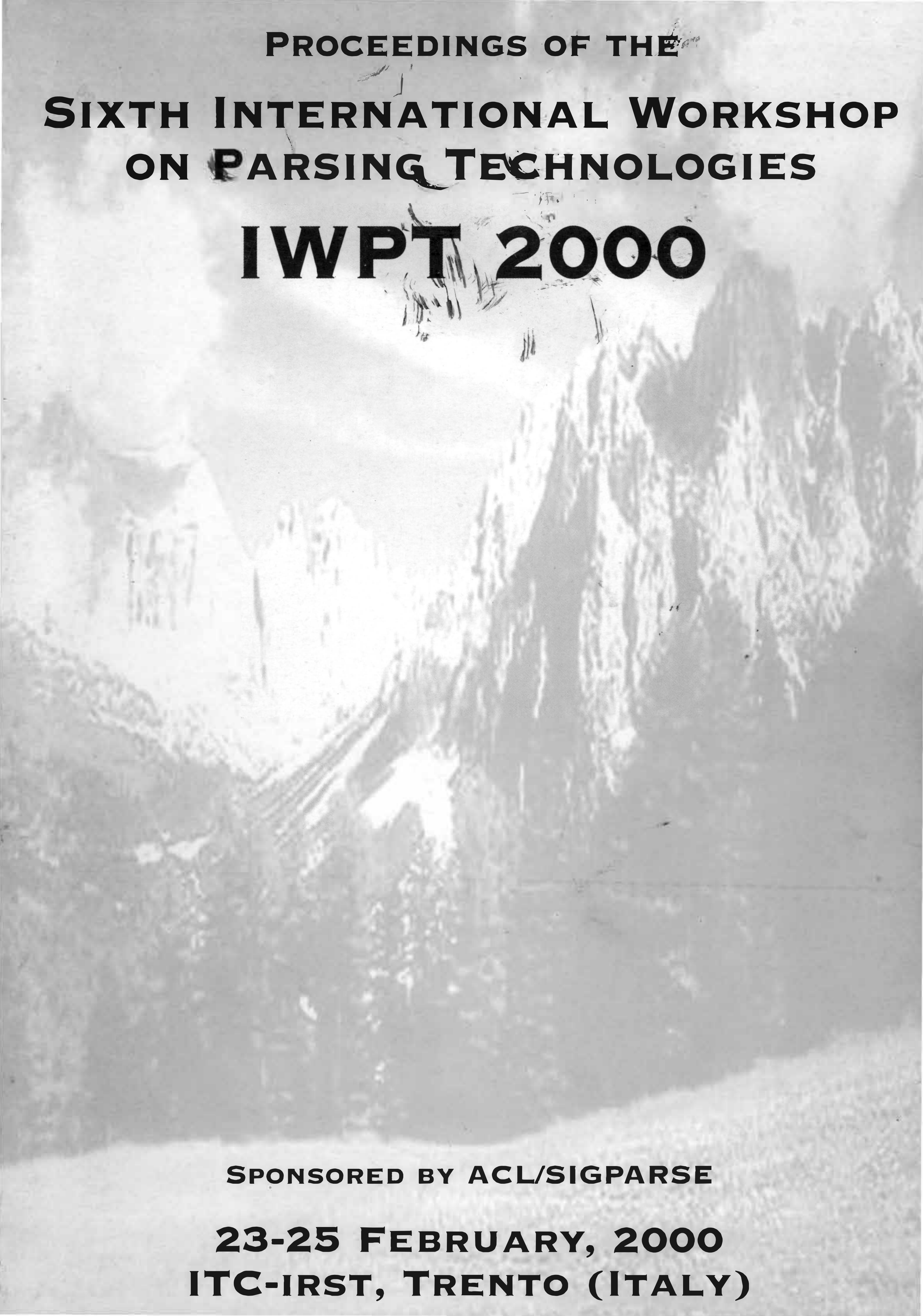
Proceedings of the

Sixth International Workshop on Parsing Technologies

IWPT 2000



Sponsored by ACL/SIGPARSE

23-25 February, 2000

ITC-irst, Trento (Italy)

Proceedings of the

Sixth International Workshop on Parsing Technologies IWPT 2000

Sponsored by ACL/SIGPARSE

23–25 February, 2000

ITC-irst, Trento (Italy)

Cover photo by P. Lattuada

Organisation — IWPT 2000

General Chair
Harry C. Bunt, ITK, Tilburg University, The Netherlands

Programme Chair
John Carroll, COGS, University of Sussex, UK

Local Chair
Alberto Lavelli, ITC-IRST, Trento, Italy

Programme Committee
Robert C. Berwick, MIT, USA
Harry C. Bunt, Tilburg University, The Netherlands
Bob Carpenter, Bell Labs, USA
John Carroll, University of Sussex, UK
Ken Church, AT&T, USA
Mark Johnson, Brown University, USA
Aravind Joshi, University of Pennsylvania, USA
Ronald Kaplan, Xerox, Palo Alto, USA
Martin Kay, Xerox, Palo Alto, USA
Bernard Lang, INRIA, Paris, France
Alon Lavie, Carnegie-Mellon University, USA
Anton Nijholt, University of Twente, The Netherlands
Christer Samuelsson, Xerox, Grenoble, France
Mark Steedman, University of Edinburgh, UK
Oliviero Stock, ITC-IRST, Trento, Italy
Hozumi Tanaka, Tokyo Institute of Technology, Japan
Masaru Tomita, Stanford University, USA
Hans Uszkoreit, DFKI, Saarbrücken, Germany
K. Vijay-Shanker, University of Delaware, USA
David Weir, University of Sussex, UK
Mats Wirén, Telia Research, Sweden

Additional Refereeing
Many thanks to Krzysztof Czuba and Chad Langley (Carnegie-Mellon University), François Barthélemy, Pierre Boullier and Eric de la Clergerie (INRIA, France), Micky Huber, Alberto Lavelli and Carlo Strapparava (ITC-IRST), Anoop Sarkar and Fei Xia (University of Pennsylvania, USA), Rieks op den Akker, Dirk Heylen, Erik Oltmans and Klaas Sikkel (University of Twente, The Netherlands) and Theo Vosse (University of Leiden, The Netherlands).

Preface

IWPT 2000, the Sixth International Workshop on Parsing Technologies, marks the existence of twelve years of parsing workshops, starting in 1989 with the workshop organized by Masaru Tomita in Pittsburgh and Hidden Valley, Pennsylvania. IWPT'89 was followed by four workshops in a biennial rhythm:

IWPT'91 in Cancún (Mexico)

IWPT'93 in Tilburg (The Netherlands) and Durbuy (Belgium)

IWPT'95 in Prague and Karlovy Vary (Czech Republic)

IWPT'97 in Boston/Cambridge (Massachusetts).

The series has achieved a number of successes, becoming the major forum for researchers in parsing technology to meet and discuss advances in the field, attracting a steady number of participants and submitted papers, and resulting in three books:

Current Issues in Parsing Technologies, Masaru Tomita, editor (Kluwer, Boston 1991);

Recent Advances in Parsing Technology, Harry Bunt and Masaru Tomita, editors (Kluwer, Dordrecht 1996);

New Developments in Parsing Technology, Harry Bunt and Anton Nijholt, editors (Kluwer, Dordrecht, in press).

In 1994, when Masaru Tomita left the arena of natural language processing and moved into biogenetic engineering (applying NL parsing techniques to DNA structures), the Special Interest Group on Parsing (SIGPARSE) was set up within ACL with the primary aim of giving continuity to the IWPT series, and has functioned as such.

Of the five previous workshops, two have been in Europe and three in the New World, so it seemed appropriate to return to the Old World for the 2000 workshop. Indeed, the venue of the sixth IWPT in Northern Italy is on historical ground, as the monuments of the city of Trento, and Verona and Venice nearby, so richly testify. I would like to thank the Institute for Scientific and Technological Research ITC-IRST in Trento for hosting IWPT 2000, in particular Alberto Lavelli for his work as local arrangements chair, and ITC-IRST director Oliviero Stock for general support.

Besides the accepted submitted papers and posters, IWPT 2000 features invited talks by Martin Kay, one of the founding fathers of parsing technology, and by Eric Brill and Giorgio Satta, well known for their outstanding contributions to the field. We are very grateful that they have accepted our invitation.

Thanks are due to the members of the Programme Committee for their careful and timely reviewing work, and especially to programme chair John Carroll for organising and supervising the reviewing and acceptance process, for designing the workshop programme, and for preparing the proceedings. Thanks to all these people, IWPT 2000 promises to be a successful continuation of the IWPT tradition into the new millennium.

Harry Bunt
SIGPARSE Officer and IWPT 2000 General Chair

Contents

Author Index

Invited Talks

Automatic Grammar Induction: Combining, Reducing and Doing Nothing

Eric Brill
Microsoft Research
One Microsoft Way
Redmond, Wa. 98052 USA
brill@microsoft.com

John C. Henderson
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730 USA
jhndrsn@mitre.org

Grace Ngai
Department of Computer Science
The Johns Hopkins University
Baltimore, Md. 21218 USA
gyn@cs.jhu.edu

Abstract

This paper surveys three research directions in parsing. First, we look at methods for both automatically generating a set of diverse parsers and combining the outputs of different parsers into a single parse. Next, we will discuss a parsing method known as transformation-based parsing. This method, though less accurate than the best current corpus-derived parsers, is able to parse quite accurately while learning only a small set of easily understood rules, as opposed to the many-megabyte parameter files learned by other techniques. Finally, we review a recent study exploring how people and machines compare at the task of creating a program to automatically annotate noun phrases.

1 Introduction

This paper briefly surveys three research directions we hope will positively impact the field of parsing: parser output combination, automatic rule sequence learning, and manual rule sequence creation.

2 Parser Combination

2.1 Combining Off-the-Shelf Parsers

There has been a great deal of recent research in the machine learning community on combining the outputs of multiple classifiers to improve accuracy. In [4], we demonstrated that taking three off-the-shelf part of speech taggers and combining their outputs results in a significant performance improvement over any single tagger. More recently [8], we have demonstrated that the same is true

	Precision	Recall	(P+R)/2	F-Measure
Best Original Parser	89.6	89.7	89.7	89.7
Constituent Voting	92.4	90.1	91.3	91.3
Parser Switching	90.8	90.7	90.7	90.7

Table 1: Comparison of Single Parser with Combination Techniques

for parsing. We took three statistical parsers that had been trained on a portion of the Penn Treebank [5, 6, 10]. We explored two methods of combination: constituent voting and parser switching.

In constituent voting, each parser posits a set of constituents with weights (votes). We take the union of all posited constituents for a sentence, accumulating their weights, and then discard all constituents with a weight below a threshold. In the simplest version of constituent voting, each parser gives a weight of 1 to any constituent it posits, and then we retain all constuents posited by more than half of the parsers. We can also weight the vote according to the accuracy of the individual parser, or according to an estimate of the accuracy of an individual parser on a particular constituent type in a particular context. It is possible that, for instance, one parser will be very accurate at predicting noun phrases and very inaccurate at predicting prepositional phrases that occur before the main verb. In fact, in studying conditional weights, we were unable to achieve better performance than that achieved using the simplest weighting scheme. This could be either due to our lack of creativity, or an indication that the three parsers we used do not differ significantly in behavior across linguistic types or contexts.

In parser switching, each parser outputs a parse and our algorithm chooses to keep one of these parses. We can define a distance measure over trees, and then given a set T of parser outputs for a particular sentence, we would like to output the centroid tree for that set. In other words, we want to output: $\text{argmin}_{T_i} \sum_{T_j \in T} \text{Dist}(T_i, T_j)$. The motivation for this is that we can think of there being a *true* parse for a sentence, and then each parser makes random errors, independent of the errors made by the other parsers. Since finding the true centroid is intractible, we instead output: $\text{argmin}_{T_i \in T} \sum_{T_j \in T} \text{Dist}(T_i, T_j)$.

In table 1, we show the results obtained using both constituent voting and parser switching. We see that both combination techniques improve upon the best statistical parser, and indeed the results obtained using constuent voting are the highest accuracy numbers reported for the Wall Street Journal to date.

2.2 Generating A Set of Parsers

We have also explored how we can exploit the advantages of classifier combination when we have access to only one trainable parser [7, 9]. One technique for doing so is known as bagging [1]. In bagging, we take a single training set of size M and from it generate a new training set by sampling with replacement M times from the original set. We can do this multiple times to create multiple training sets. Then each derived training set is used to train a separate parser.

We generated 18 bags from a Czech treebank [7] and then used each bag to train a Collins Parser [6]. We then used simple constituent voting to combine the outputs of these 18 parsers on test data. Figure 1 shows that the number of bagged parsers that posit a constituent correlates nicely with the accuracy of that constituent. When only one of the 18 parsers outputs a constituent, that constituent is correct less than 10% of the time. When all 18 parsers output a constituent, that constituent is

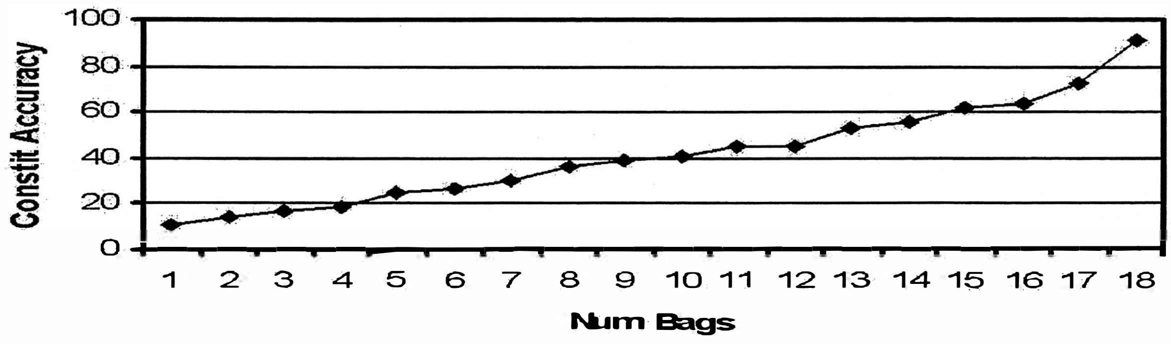


Figure 1: Accuracy versus number of parsers positing a constituent

Parser	Precision	Recall	P+R	F-Measure
Original	79.1	79.1	158.3	79.1
Bagged	79.9	79.9	159.8	79.9

Table 2: Czech Test Set Bagging Results

correct more than 90% of the time. Table 2 shows the results on Czech test data of a single Collins parser trained on the entire training corpus, compared to using bagging to generate 18 parsers and then combining the outputs of these parsers. Similar results have been obtained for English.

3 Transformation-Based Parsing

There have been great advances recently in the accuracy of parsers that are automatically trained from parsed corpora. One disadvantage of these grammar induction methods is that the derived linguistic knowledge is captured in opaque parameter files, typically many megabytes large. This makes it a challenge to capitalize on human intuitions to improve the machine-derived grammars. An alternative to these statistical induction methods is a method called Transformation-Based Parsing (TBP) [2, 11, 12]. In TBP, a grammar consists of an ordered sequence of tree transform rules. To learn the rules, we begin with some initial annotation of the training corpus (for instance, every sentence parsed as a flat structure under an S-node), and then we iteratively search for the transform rule whose application will bring the training set closest to the set of true parses, we append that rule to our rule list and apply it to the training corpus. Transform rules can add, delete or rearrange structure. An example of a learned rule is:

If a sequence begins with a Determiner and ends with a Noun and has a Verb or Modal immediately to the right of it, then bracket that sequence as an NP

Parser	F-Measure
PCFG	73.4
Transformations	83.0
Collins	86.7

Table 3: Transformation-Based Parsing Test Set Results

While there is still much work to be done in improving TBP, we have found that in its current state we achieve significantly better performance than a corpus-derived PCFG, but worse performance than the best statistical methods. In table 3 we show results from training on a 20k-sentence subset of the Penn Treebank WSJ corpus.

While the transformation-based system underperforms the best statistical systems, we believe it is an exciting path worth pursuing for the following reason: the Collins parser needs a trained parameter file about two hundred megabytes in size, while the transformation-based system achieves its accuracy with just a few hundred (mostly) readily understood rules. A human expert can easily study the rule list, and edit or add rules as appropriate.

4 Manual Rule Writing

In parsing, machine-learned rule-based systems achieve accuracy near that obtained by statistical systems. In many other natural language tasks, such as part of speech tagging, word sense disambiguation, and noun phrase chunking, machine-learned rule-based systems achieve performance on par with the best statistical systems, always learning a relatively small sequence of simple rules. This raises the question of whether we are really gaining anything by using machine learning techniques in NLP. While clearly a 200MB file of parameters is not something a person could create manually, creating small rule lists certainly is something a person could do.

In [3], we addressed this question for noun phrase chunking. We built a system to allow people to easily write rule sequences by hand and provided tools for effective manual error analysis. The advantage of a rule sequence, compared to a CFG for instance, is that the human does not have to be concerned about how rules interact. In manually generating a weighted CFG, there are complex interactions between rules, which makes manual grammar adjustment difficult. With rule sequences, at any stage of development the list of rules currently in the sequence can be ignored. In composing the i+1st rule, the person need only study the corpus and try to derive a rule to improve the accuracy of the training corpus in its current state, having been processed by the previous i rules.

We had students in a Natural Language Processing class write rule sequences for NP-bracketing. We found that the best students wrote rules that achieved test-set performance comparable to the best machine-learning system, and were able to achieve this performance after just a few hours of rule writing. We believe this raises a number of interesting issues, such as: Does machine learning really help? Can we effectively combine the (hopefully complementary) abilities of machine learning algorithms and human rule-writers?

5 Summary

We have briefly surveyed three lines of research for parsing: combining the outputs of multiple trained systems, reducing the problem to learning a small set of simple rules, and doing nothing automatically.

Acknowledgements

This work was partly funded by NSF grant IRI-9502312.

References

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[2] E. Brill. Transformation-based error-driven parsing. In *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, The Netherlands, 1993.

[3] Eric Brill and Grace Ngai. Man vs. machine: A case study in base noun phrase learning. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, 1999.

[4] Eric Brill and Jun Wu. Classifier combination for improved lexical disambiguation. In *Proceedings of the 17th International Conference on Computational Linguistics*, 1998.

[5] Eugene Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artifial Intelligence*, 1997.

[6] Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Assocation of Computational Linguistics*, 1997.

[7] Jan Hajič, E. Brill, M. Collins, B. Hladka, D. Jones, C. Kuo, L. Ramshaw, O. Schwartz, C. Tillmann, and D. Zeman. Core natural language processing technology applicable to multiple languages. *Prague Bulletin of Mathematical Linguistics*, 70, 1999.

[8] John Henderson and Eric Brill. Exploiting diversity in natural language processing: Combining parsers. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, 1999.

[9] John Charles Henderson. *Exploiting Diversity for Natural Language Parsing*. PhD thesis, Johns Hopkins University, August 1999.

[10] Adwait Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, 1997.

[11] Giorgio Satta and Eric Brill. Efficient transformation-based parsing. In *Proceedings of ACL*, 1996.

[12] Marc Vilain and David Palmer. Transformation-based bracketing: fast algorithms and experimental results. In *Proceedings of the Workshop on Robust Parsing (at ESSLLI-96)*, 1996.

GUIDES AND ORACLES FOR LINEAR-TIME PARSING

Martin Kay

Stanford University and Xerox Palo Alto Research Center

kay@parc.xerox.com

Abstract

If chart parsing is taken to include the process of reading out solutions one by one, then it has cxponential complexity. The stratagem of separating read-out from chart construction can also be applied to other kinds of parser, in particular, to left-corner parsers that use early composition. When a limit is placed on the size of the stack in such a parser, it becomes context-free equivalent. However, it is not practical to profit directly from this observation because of the large state sets that are involved in otherwise ordinary situations. It may be possible to overcome these problems by means of a *guide* constructed from a weakened version of the initial grammar.

A recognition procedure for a language is a method of determining whether a given string belongs to the language. In the context-free case, it clearly reduces to showing that the string is a phrase of a particular category, the goal category of the grammar. A string α belongs to category C if either, α consists of the single symbol C, or there is a rule $C \longrightarrow c_1 \ldots c_n$ and α is the concatenation of strings that are phrases of categories $c_1 \ldots c_n$, in that order. The proof that a string is a phrase of a given category can be summarized in an ordered tree with nodes named for grammar symbols and this is what we refer to as the structure of the string according to the grammar. The root is named for the grammar's distinguished symbol and the daughters of a node labeled C are labeled, from left to right, $c_1 \ldots c_n$, given that $C \longrightarrow c_1 \ldots c_n$ is a grammar rule.

Concretely, a rule $s \longrightarrow np\ aux\ vp$ can be transcribed directly into Prolog as a *definite-clause grammar* (DCG) somewhat as follows:

```
wof(s, A, D) :-
  wof(np, A, B),
  wof(aux, B, C),
  wof(vp, C, D).
```

There is a word or phrase (wof) of category s stretching from point A to point D in the string if, for some points B and C between A and D, there is a phrase of category np from A to B, of category aux from B to C, and of category vp from C to D. A terminal symbol, say *dog,* is recognized as belonging to category n by virtue of the clause:

```
wof(dog, [dog | X], X).
```

This is based on the convention of using suffixes of the string as names of points in it. In particular, this clause says that a string consists of a noun followed by a string X if it consists of the word *dog* followed by X.

With these, and few more obvious definitions, the Prolog interpreter will be able to prove the proposition

```
wof(s, [the, dog, will, chase, the cat], [])
```

The proof will be carried out in accordance with the so-called *recursive-descent*, or *top-down backtracking* strategy suggested by our initial definitions. In order to show that a string is a word or phrase of category s, the procedure is to first show that it begins with a phrase of category np, and then to show that the remainder of the string consists of the *aux* phrase followed by a vp. Each of these steps consists of a recursive application of the same procedure.

To get the structure of a string, one must arrange to capture the control structure of the recognition process, and this can be done in a variety of ways. To capture all the possible structures of a string, it is necessary to behave on success just as one does on failure, by backing up to the most recent choice point with hitherto unexplored branches.

As an effective recognition or parsing algorithm, the flaws of DCG are well known. The two principal ones are (1) that the assymptotic time complexity is exponential in the length of the string, and (2) that the procedure does not terminate when a phrase of a given category can have an initial substring that must be analyzed as belonging to that same category.

The information that the recognition procedure amasses about the a string can be summarized in the manner exemplified below:

```
oracle(s, [the, dog, will, chase, the, cat], []).
oracle(np, [the, dog], [will, chase, the, cat]).
oracle(det, [the], [dog, will, chase, the, cat]).
oracle(vp, [will, chase, the, cat], []).
...
```

Suppose that the clauses embodying the grammar are augmented as follows.

```
wof(s, A, D) :-
  oracle(np, A, B),
  wof(np, A, B),
  oracle(aux, B, C),
  wof(aux, B, C),
  oracle(vp, C, D)
  wof(vp, C, D).
```

With this augmented grammar and the oracle, the process of recognition is completely trivialized—in fact the first `oracle` clause is all we need for recognition. Since the oracle does not provide structures, however, the control structure of the recursive-decent analysis process must still be recorded if it is required to parse the string. Notice, however, that the existence of such an oracle would eliminate one of the problems with recursive-decent analysis, namely failure to terminate in cases of left recursion, and it alleviates the other by removing from the search space all moves that do not belong to successful paths. It is this last property that motivates the use of the term "oracle".

The interest in recasting top-down syntactic analysis in this way comes from the analogy that can be drawn to chart parsing. The oracle is essentially a chart and the `wof` grammar predicate supplies the read-out procedure. It is usual to include more information in the chart so that the read-out procedure does not have to have information from the grammar rules. In the present formulation, edges contain no information about the members of a phrase, so that polynomial complexity is achieved automatically without having to conflate edges with the same category symbol and string coverage, but different internal structures.

Computational linguists are generally comfortable with the claim that chart parsing with a context-free grammar has polynomial assymptotic time complexity. Since a context-free grammar can assign a number of structures to a string that increases as an exponential function of its length, we assume that there is a tacit agreement not to count the read-out procedure, but only the process of building the chart that will serve as an oracle for the read-out procedure. How this is justified in detail is not clear. Intuitively, however, dividing the parsing process into a first stage in which a data base of grammatical information about the string is constructed and a second stage in which individual analyses are read out has the advantage of allowing different parsing algorithms to be seen as idffering in complexity on the basis of the first, and intuitively more interesting part of the process.

In the balance of this paper, we outline a parsing algorithm that is very different from chart parsing in its details, but similar in that it proceeds in two stages, one in which a data base is constructed at quite attractive cost in complexity, and one in which individual analyses are read out by a simple, oracle-driven, backtracking parser. It will turn out, however, that the approach can form the basis of a practical parser only if the influence of grammar size on the overall process can be brought under control. For this purpose, we introduce the notion of a *guide*, which is a weak form of an oracle. If an oracle is available at a particular branch in a process, it can be counted on to eliminate all choices that do not lead to a successful outcome. A guide will, in general, not eliminate all unproductive choices, but it can be counted on not to eliminate any choices that do could lead to a successful outcome. As an example of a guide, consider the weakened form of chart represented in the following clauses:

```
guide([the, dog, will, chase, the, cat], []).
guide([the, dog], [will, chase, the, cat]).
guide([the], [dog, will, chase, the, cat]).
guide([will, chase, the, cat], []).
  ...
```

and a read-out procedure based on clauses like the following:

```
wof(s, A, D) :-
  guide(A, B),
  wof(np, A, B),
  guide(B, C),
  wof(aux, B, C),
  guide(C, D)
   wof(vp, C, D).
```

This chart shows where there are phrases in the string, but does not give their grammatical category. It is sufficient, however, to eliminate problems arising from left-recursive grammars.

The scheme we will outline provides analyses of strings in linear time with a context-free grammar. There is good reason to believe that this is not possible if all the structures allowed by the grammar are to be recovered, and our scheme will indeed ignore certain structures. However, there is also good reason to believe that the structures that we shall ignore are also not accessible to humans and, if this is the case, then nothing but good can come from leaving them out of account.

Abney and Johnson (1989) have shown that a left-corner parser with early composition uses stack space in proportion to the amount of center embedding in the structure. Such a parser is clearly also equivalent to a finite-state automaton which can recognize a string in linear time. One problem with this is that a finite-state automaton can serve only as a recognizer, and not as a parser. However, a recognizer and can serve as an oracle for parser[1]. The idea is simply to scan the string to be parsed from right to left, using a finite-state automaton that recognizes the reverse of the depth-limited version of the context-free language and to associate with the space between each pair of words the state of the machines. When read from left to right, this sequence of states serves as an oracle for the left-corner parser with the early composition and a finite stack.

Unfortunately, even for modest sized grammars, and an early limit on stack size, the numbers states in the automaton is unmanageably large so that it cannot be represented explicitly and therefore cannot be made deterministic. But, while undoubtedly a setback, this does not entirely upset the plan. Recognition with a nondeterministic automaton is possible in linear time if an appropriate strategy is employed for caching the states reached following a given substring. This follows from the fact that the number of alternative states that the automaton can be in after a given sub-string is limited by properties of the automaton and not by the length of string. However, even this is not enough to bring the cost of the computation within reasonable bounds because the number of stacks configurations that are possible following even a fairly short string can also be unmanageably large.

The idea of providing an oracle to control the construction of the sequence of state sets that will, in its turn, serve as an oracle in reading out the structures of the string suggests itself, but it is difficult to see how such an oracle would differ from the structure it is intended to help assemble. The intuition is that a useful oracle must contain only a part of the information in the structure whose assembly it controls. However, the possibility of a guide may be more promising. The idea will be to construct a weaker version of the context free grammar, which assigns to any given string a superset to of the structures that are signed by the original grammar, but which gives rises to an automaton with a smaller set of states. These states map in a systematic way onto those of the original automaton and, when this is applied to the string, the only states that will be considered at a given point will be those corresponding to states through which the smaller automaton has passed.

A simple way to construct a weakend version of a grammar is to construct a partition that symbols and to map each class in the partition onto a single symbol. The rules of the weekend grammar are simply the images are under this mapping of the rules in the original grammar. The new grammar will be weeker to the extent that it derives from a smaller number of a larger classes . Since the total number of symbols in the weakened grammar is smaller than that in the original grammar, so is the number of possible stack configurations.

The picture we now have is of a parser that proceeds in three phases. First, it scans the string from left to right using a left corner recognizer based on the weakened grammar, annotating the spaces between the words with the sets of states that the automaton is in at that point. Next, it scans the string from right to left using the left-corner recognizer based on the full grammar and allowing states to be entered only if they map onto members of the list of states associated with that point in the string in the preceding phase. The reason for the reversal of direction is simply to ensure that the states on each list that is encountered are reachable from the other end of the string, thus providing a guide for the present scan that is, to the extent possible, predictive. Following the practice in chart parsing, we declare these first two phases to constitute a parser and

1. This is reminiscent of the way the first member of a bimachine (Schützenberger, 1961) is used to control the operation of the second member.

declare its assymptotic in complexity to be linear . The third phase reads out structures using the original context free grammar and the left-corner parser with the early composition whose stack states must be chosen from those associated with the string in the second phase.

One apparently minor matter remains, namely how to construct a weakened version of a particular grammar that will serve as an effective guide, in this process. Surprisingly, this proves to be the sticking point. One possibility would be simply to construct the partition of the grammar symbols in a random fashion. Another would be to eliminate the distinctions made by X-bar theory, collapsing, for example, N, N-bar, and NP onto the same symbol. Yet another would be to eliminate "minor" grammatical matters , such as agreement from the first scan. The disturbing fact is that none of these things can be counted on to give a weakened grammar with desirable properties. Either the grammar remains essentially unchanged, or it reduces to one that accepts almost everything. Minor changes can easily cause it to move, almost chaotically, from one of these conditions to the other. I offer this as a challenge to the parsing community.

References

Abney, S. P. and M. Johnson (1989). "Memory requirements and local ambiguities of parsing strategies." *Journal of Psycholinguistic Research* 18(1): 129-144.

Schützenberger, Marcel Paul. 1961. A remark on finite transducers. *Information and Control*, 4. pp. 185-187.

PARSING TECHNIQUES FOR LEXICALIZED CONTEXT-FREE GRAMMARS

Giorgio Satta
Dip. di Elettronica e Informatica
Università di Padova
via Gradenigo 6/A
35131 Padova, Italy
satta@dei.unipd.it

1 Introduction

In recent years, much of the parsing literature has focused on so-called lexicalized grammars, that is grammars in which each individual rule is specialized for one or more lexical items. Formalisms of this sort include dependency grammar [13], lexicalized tree-adjoining grammar [17], link grammar [20], head-driven phrase-structure grammar [14], tree insertion grammar [18], combinatorial categorial grammar [21] and bilexical grammar [7]. Probabilistic lexicalized grammars have also been exploited in state-of-the-art, real-world parsers, as reported in [11], [1], [6], [2], [4], and [9]. Other parsers or language models for speech recognition that do not directly exploit a generative grammar, still are heavily based on lexicalization, as for instance the systems presented in [10], [12], [15] and [3].

The wide diffusion of lexicalized grammars is mainly due to the capability of these formalisms to control syntactic acceptability, when this is sensitive to individual words in the language, and word selection, accounting for genuinely lexical factors as well as semantic and world knowledge conditions. More precisely, lexicalized grammars can select the complements and modifiers that a constituent can take, on the basis of special words playing a particularly informative role within the given constituent and the given complements and modifiers. These special words are typically identified with the lexical head and co-heads of a constituent, where with the term co-head of a constituent A we denote a lexical head of any of the subconstituents of A. To give a simple example (from [8]), the word *convene* requires an NP object to form a VP, but some NPs are more lexically or semantically appropriate than others, and the appropriateness depends largely on the NP's head, e.g., the word *meeting* vs. the word *party*. In this way, the grammar is able to make stipulations on the acceptability of input sentences like *Nora convened the meeting* and *Nora convened the party*. This was not satisfactorily captured by earlier formalisms that did not make use of lexicalization mechanisms. See [5] for further discussion.

Within the parsing community, and in the speech community as well, a considerable research effort has been devoted to the important problem of defining statistical parameters associated with lexicalized grammars, and to the problem of the specification of algorithms for the statistical estimation of these parameters. In contrast, not much has been done with respect to the sentence processing problem. Most of the above mentioned systems parse input strings using naïve adaptations of existing algorithms developed for the unlexicalized version of the adopted formalism, possibly in combination with heuristics specially tailored to cut down the parsing search space. However, the internal structure and the large size of formalisms derived by means of some lexicalization mechanism render these systems unsuitable to be processed with traditional parsing techniques. In the talk we address these problems and present novel parsing algorithms for lexicalized grammars that overcome the computational inefficiencies that arise when standard algorithms are used. We will focus on lexicalized context-free grammars (LCFGs), a class of grammars originally introduced in [8]. LCFGs are a convenient abstraction that allows us to study important computational and generative properties that are common to several of the above mentioned lexicalized grammars. A similar abstraction, called probabilistic feature grammars (PFG), has been presented in [9], motivated by parameter estimation problems. In contrast to PFG, features with lexical values have a special status within LCFG: this facilitates analysis of several complexity measures. In what follows, we report a formal definition of LCFGs and give a brief outline of the results that will be more carefully presented in the talk.

2 Lexicalized context-free grammars

We can think of a lexicalized context-free grammar as a particular kind of CFG obtained by applying a lexicalization procedure to some underlying CFG.[1] Before presenting the formal definition of LCFG, we briefly discuss an example. Consider the sample phrase *dumped sacks into a bin.* In order to be able to capture the lexical and syntactic relations holding among the words in this phrase, we pair each of the standard nonterminals NP, VP, etc., with a tuple of words from the phrase itself. The nonterminals of the resulting grammar are therefore of the following kind: V[dump], NP[sack], VP[dump][sack], etc. Using these new lexicalized nonterminals we can write new context-free productions of the form VP[dump][sack] $\rightarrow$ V[dump] NP[sack]. The main idea here is that lexical items appearing in the right-hand side nonterminals can be inherited by the left-hand side nonterminal. A possible derivation of the above phrase is depicted in Figure 1.

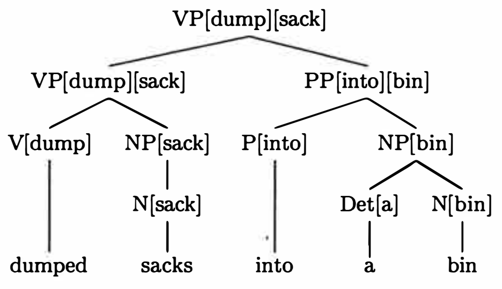


Figure 1: A sample derivation in an LCFG.

We now give a formal definition of LCFG. A **lexicalized context-free grammar** is a CFG $G = (V_N, V_T, P, S[\$])$, with V_N and V_T the set of nonterminal and terminal symbols, respectively, and with $S[\$] \in V_N$ a special start symbol. The following conditions are all satisfied by G:

[1]The particular way we lexicalize CFGs differs from the proposal in [18], where a CFG is transformed into a tree substitution grammar. Also, we note here in passing that an LCFG could be alternatively defined as a very restricted kind of attribute grammar with only synthesized attributes [16].

(i) there exists a set V_D, called the set of **delexicalized nonterminals**, such that

$$V_\mathrm{N} \subseteq \{A[a_1][a_2]\cdots[a_r] \mid A \in V_\mathrm{D},\ r \geq 1,\ a_j \in V_\mathrm{T},\ 1 \leq j \leq r\};$$

(ii) every production in P has one of the following two forms:

(a) $A_0[a_{0,1}]\cdots[a_{0,r_0}] \to A_1[a_{1,1}]\cdots[a_{1,r_1}]\ A_2[a_{2,1}]\cdots[a_{2,r_2}]\ \cdots\ A_q[a_{q,1}]\cdots[a_{q,r_q}]$,
where $q \geq 1$ and multiset $\{a_{0,1},\ldots,a_{0,r_0}\}$ is included in multiset $\{a_{1,1},\ldots,a_{1,r_1},a_{2,1},\ldots,a_{q,r_q}\}$;

(b) $A[a] \to a$.

The multiset condition in (ii)a states that the lexical items in the left-hand side nonterminal are always inherited from the nonterminals in the right-hand side. Note also that the start symbol $S[\$]$ is a lexicalized nonterminal. As a convention, we assume that \$ is a dummy lexical item that does not appear anywhere else in G, and disregard it in the definition of $L(G)$.

In current practice, the set of delexicalized nonterminals V_D is kept separate from set V_T. Set V_D usually encodes word sense information and other features as number, tense, etc., structural information as bar-level and subcategorization requirements, and any additional information that does not explicitly refer to individual lexical items, as for instance contextual constraints for parent node category [2], or constraints on the constituent's yield, expressed through finite information about distribution of some lexical category [4].

Let G be some LCFG and let p be some production in P. If p has the form in (ii)a above, let $k_p = \sum_{j=1}^{q} r_j$; otherwise, let $k_p = 0$. The **degree of lexicalization** of an LCFG G is defined as $k_G = \max_{p\in P} k_p$. We also say that G is a k_G-lexical CFG. Note that from condition (ii)a it directly follows that G has productions with right-hand side length bounded by k_G. When the set of delexicalized nonterminals is fixed, the degree of lexicalization induces a proper hierarchy on the class LCFG. More precisely, it can be shown that, for any non-empty set V_D and any $k \geq 3$, there exists a k-lexical CFG G defined on V_D such that $L(G)$ cannot be generated by any k'-lexical CFG defined on V_D and with $k' < k$.

The class of 2-lexical CFG, also called bilexical CFG, turns out to be of major importance in current parsing practice. In fact, the probabilistic formalisms adopted in [1], [6], [2] and [4] are strongly equivalent to bilexical grammars, in the following sense. For each of the above formalisms, we can effectively construct a corresponding probabilistic bilexical grammar with the following properties. There is a one-to-one mapping between derivations in the source formalism and derivations in the target bilexical grammar. This map preserves the associated probabilities and can be computed through a homomorphism (homomorphisms can be implemented as real-time, memoryless processes). Most important here, when computational problems related to parsing must be investigated, bilexical grammars are a useful abstraction of the above mentioned formalisms, and parsing algorithms developed for bilexical grammars can directly be adapted to these formalisms. In Section 3, we will mainly focus on bilexical grammars.[2]

3 LCFG Parsing

The cost of the expressiveness of an LCFG is a very large production set. In the simplest case of bilexical CFGs, the size of set P usually grows with the square of the size of V_T, and thus can be very

[2]The adoption of bilexical formalisms in state-of-the-art, real-world parsers is related to the fact that currently available natural language annotated corpora are still limited in their size, so that the estimation of probabilities for lexical relations of arity greater than two is still quite problematic.

large. Standard context-free parsing algorithms, which run in time linear with the size of the input grammar, are inefficient in such cases. Therefore, a first goal in the design of a parsing algorithm for LCFGs is the one of achieving running time sublinear with respect to the grammar size. As a first result, we show that algorithms satisfying the so-called correct-prefix property [19] are unlikely to achieve this goal, even in case the grammar is precompiled in an amount of time polynomial in its size.

In order to achieve the sublinear time goal, a usual practice is to use standard CFG parsing algorithms and to select only those rules in the grammar that are lexically grounded in the input sentence. However, this in practice adds a factor of n^2 for an input string of length n, resulting in $O(n^5)$ running time for bilexical CFGs and related formalisms. We show how dynamic programming can be exploited to achieve an $O(n^4)$ result. Also, we specify an $O(n^3)$ time algorithm for a restricted kind of bilexical CFGs. We argue that the proposed restriction leaves enough power to the formalism to capture most common natural language constructions. We discuss how these results generalize to LCFG with higher degree of lexicalization and to lexicalized tree-adjoining grammars [17] as well.

The above algorithms exploit bottom-up strategies, which are the most common in parsing lexicalized grammars. We show how top-down strategies can be applied in parsing bilexical CFGs, still retaining the desired condition on sublinear running time with respect to the input grammar size. This is done by exploiting generalized look-ahead techniques. We argue that the resulting strategies are even more selective than standard top-down strategies satisfying the correct-prefix property.

Acknowledgements

This abstract presents ongoing research in collaboration with Jason Eisner, University of Rochester, and Mark-Jan Nederhof, DFKI. The author is supported by MURST under project *PRIN: BioInformatica e Ricerca Genomica* and by University of Padua, under project *Sviluppo di Sistemi ad Addestramento Automatico per l'Analisi del Linguaggio Naturale.*

References

[1] H. Alshawi. Head automata and bilingual tiling: Translation with minimal representations. In *Proc. of the 34th ACL*, pages 167–176, Santa Cruz, CA, 1996.

[2] E. Charniak. Statistical parsing with a context-free grammar and word statistics. In *Proc. of AAAI-97*, Menlo Park, CA, 1997.

[3] C. Chelba and F. Jelinek. Exploiting syntactic structure for language modeling. In *Proc. of the 36th ACL*, Montreal, Canada, 1998.

[4] M. Collins. Three generative, lexicalised models for statistical parsing. In *Proc. of the 35th ACL*, Madrid, Spain, 1997.

[5] M. Collins. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA, 1999.

[6] J. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proc. of the 16th COLING*, pages 340–345, Copenhagen, Denmark, 1990.

[7] J. Eisner. Bilexical grammars and a cubic-time probabilistic parser. In *Proceedings of the 4th Int. Workshop on Parsing Technologies*, MIT, Cambridge, MA, September 1997.

[8] J. Eisner and G. Satta. Efficient parsing for bilexical context-free grammars and head automaton grammars. In *Proc. of the 37th ACL*, pages 457–464, College Park, Maryland, 1999.

[9] J. Goodman. Probabilistic feature grammars. In *Proceedings of the 4th Int. Workshop on Parsing Technologies*, MIT, Cambridge, MA, September 1997.

[10] F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, A. Ratnaparkhi, and S. Roukos. Decision tree parsing using a hidden derivation model. In *Proceedings of the 1994 Human Language Technology Workshop*, pages 272–277, 1994.

[11] J. Lafferty, D. Sleator, and D. Temperley. Grammatical trigrams: A probabilistic model of link grammar. In *Proc. of the AAAI Conf. on Probabilistic Approaches to Nat. Lang.*, October 1992.

[12] D. Magerman. Statistical decision-tree models for parsing. In *Proc. of the 33rd ACL*, pages 276–283, Cambridge, MA, 1995.

[13] I. Mel'čuk. *Dependency Syntax: Theory and Practice.* State University of New York Press., 1988.

[14] C. Pollard and I. Sag. *Head-Driven Phrase Structure Grammar.* University of Chicago Press., 1994.

[15] A. Ratnaparkhi. A linear observed time statistical parser based on maximum entropy models. In *Second Conference on Empirical Methods in Natural Language Processing*, Brown University, Providence, Rhode Island, 1997.

[16] G. Rozenberg and A. Salomaa. *Handbook of Formal Languages*, volume 1. Springer-Verlag, Berlin, Germany, 1997.

[17] Y. Schabes, A. Abeille, and A. K. Joshi. Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proc. of the 12th COLING*, pages 578–583, Budapest, Hungary, 1988.

[18] Y. Schabes and R. C. Waters. Tree insertion grammar: A cubic-time parsable formalism that lexicalizes context-free grammar without changing the trees produced. *Computational Linguistics*, 21(4):479–515, 1995.

[19] S. Sippu and E. Soisalon-Soininen. *Parsing Theory: LR(k) and LL(k) Parsing*, volume 2. Springer-Verlag, Berlin, Germany, 1990.

[20] D. Sleator and D. Temperley. Parsing english with a link grammar. In *Proceedings of the 3trd Int. Workshop on Parsing Technologies*, Tilburg, Durbuy, Germany, August 1993.

[21] M. Steedman. *Surface Structure and Interpretation.* The MIT Press, Cambridge, MA, 1996.

Papers

A BOOTSTRAPPING APPROACH TO PARSER DEVELOPMENT

Izaskun Aldezabal, Koldo Gojenola, Kepa Sarasola

Department of Computer Languages and Systems
Informatika Fakultatea, 649 P. K., Euskal Herriko Unibertsitatea,
20080 Donostia (Euskal Herria)

{jibalroi, jipgogak, jipsagak}@si.ehu.es

Abstract

This paper presents a robust parsing system for unrestricted Basque texts. It analyzes a sentence in two stages: a unification-based parser builds basic syntactic units such as NPs, PPs, and sentential complements, while a finite-state parser performs syntactic disambiguation and filtering of the results. The system has been applied to the acquisition of verbal subcategorization information, obtaining 66% recall and 87% precision in the determination of verb subcategorization instances. This information will be later incorporated to the parser, in order to improve its performance.

1 Introduction

As NLP-based applications are growing, there is a stronger need for wide-coverage parsing systems. At the moment, comprehensive grammars are available for some languages, like English [Briscoe and Carroll 1993], or parallel LFG German, French and English grammars [Butt et al., 1999], developed after a considerable effort. Moreover, the huge size of the now available corpora demands successive extensions of the grammars, to include corpus-specific information or to augment the basic syntactic grammars with lexical information, like subcategorization frames or selectional restrictions [Briscoe and Carroll 1997, Carroll and Rooth 1998]. However, the situation is different for most other languages, due to several reasons:

- Limited number of language users. This fact implies a reduced number of researchers/developers of computational linguistic tools.
- Limited number of language resources, in the form of computational lexicons, grammars, corpora, annotated treebanks or dictionaries.

Although there are current efforts for the development of parsing systems for other languages [Oflazer 1999, Hajic and Hladká 1998], there will always be the problem of reaching the complexity and performance of the parsers for the most studied languages. This is in spite of the efforts to make publicly available language resources (ELRA) that could at most alleviate the problem. Therefore, methods must be devised which obtain results automatically, minimizing development costs.

This work presents both the development of a parsing system for unrestricted Basque texts and the first results obtained using it in the process of acquiring subcategorization information. The system is applied to Basque, which has as its main characteristics being agglutinative and having basically constituent-free order. These characteristics involve some complexities for syntactic analysis.

As a first step, a basic parsing system has been developed. It consists of two modules, applied sequentially: an unification based chart-parser and a finite-state parser. This combined system covers the syntactic core of the language; however, although it is useful for several non-trivial applications like the detection of syntactic errors, is still incomplete, lacking important aspects like subcategorization information. For this reason, we have applied this basic parser to text corpora, with the aim of obtaining subcategorization information that will be used to enrich the lexical database. This way, we plan to develop a parsing system in a bootstrapping fashion, with incremental improvements.

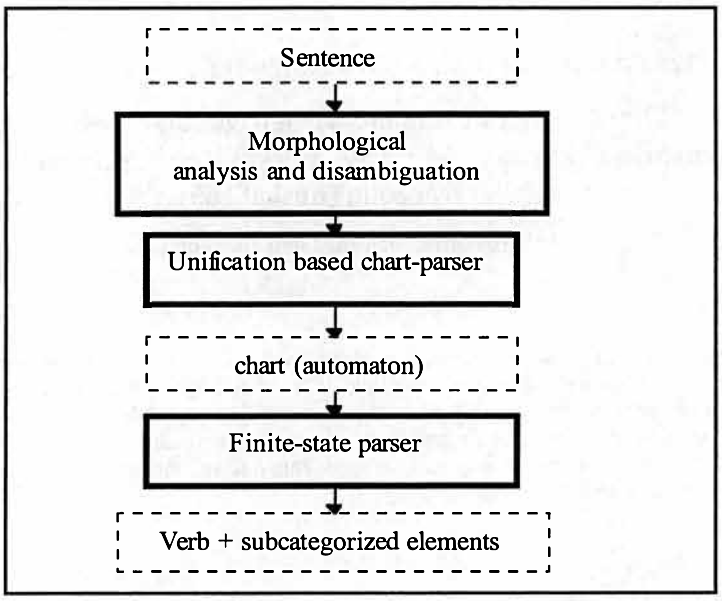

Sentence
Morphological analysis and disambiguation
Unification based chart-parser
chart (automaton)
Finite-state parser
Verb + subcategorized elements

Figure 1. Overview of the system.

The rest of the paper is organized as follows. Section 2 presents the basic parsing system we have implemented, detailing its main components and justifying its sequential architecture. It also examines the application of the system to the extraction of subcategorization information. Section 3 gives the results of its evaluation against a set of manually tagged 500 sentences, while section 4 reviews the literature on parsing systems and automatic acquisition of subcategorization information.

2 The Parser

We have developed a parsing system divided in two main modules: a unification based parser and a finite-state parser (see figure 1). Prior to parsing, there is another step concerned with morphological analysis and disambiguation, using the basic tools for Basque (http://ixa.si.ehu.es) that have been developed in previous projects:

- The lexical database. It is a large repository of lexical information, with about 70.000 entries (including lemmas and declension/derivational morphemes), each one with its associated linguistic features, like category, subcategory, case and number, contained in a commercial database management system.
- Morphological segmentation. Inflectional morphology of Basque was completely described in [Alegria *et al.* 1996]. This system applies Two-Level Morphology for the morphological description and obtains for each word its segmentation(s) into component morphemes, where each morpheme is associated with its corresponding features in the lexicon. The segmentation module has full coverage of free-running texts in Basque, capable of treating unknown words and non-standard forms (dialectal variants and typical errors).

- Morphological disambiguation. A disambiguation system was implemented for the assignment of the correct lemma and part-of-speech to each token in a corpus [Ezeiza *et al.* 1998] taking the context into account, by means of statistical (Hidden Markov Models) and hand-crafted rules (Constraint Grammar (CG) formalism [Samuelsson and Voutilainen 1997]). This tool reduces the high word-level ambiguity from 2.65 to 1.19 interpretations, still leaving a number of interpretations per word.

2.1 The Unification Based Parser

After morphological analysis and disambiguation, each word is assigned one or more readings, each of them as a list of its components (lemma and morphemes) with their associated morphosyntactic information. Some facts concerning syntactic analysis must be taken into account:

- The morpheme is the basic unit of syntactic analysis, following the most extended syntactic descriptions for Basque [Abaitua 1988]. In figure 2, dashed lines represent lemmas and morphemes, that is, units smaller than the word that will form the input to the syntactic analyzer. This kind of analysis has been adopted by other systems for agglutinative languages like Hungarian [Prószéky 1996] and Turkish [Oflazer 1999].
- Regarding the syntactic structure of Basque, it has been considered as a language with free order of constituents. However, this is only true for main sentence constituents with respect to the verb (such as noun phrases, prepositional phrases and sentential complements), because inside those constituents the order of elements is fixed or quite limited. Moreover, the syntactic relationships inside these fixed order components require the testing of complex agreement and the building of non-trivial syntactic structures, not definable by finite-state techniques [Beesley 1998]. These facts led us to describe the syntax of these components by means of feature structures, using a unification based formalism.
- There are more problems if we want to go beyond the level of the main sentential constituents (verbs, NPs, PPs and sentential complements). As their relative order is almost free, their analysis would suppose the proliferation of a high number of unsolvable attachment ambiguities. Example 1 shows the presence of two elements (PP and NP) that could be attached to either of the two surrounding verbs (giving three different interpretations). Although this kind of ambiguity can be resolved in some cases (the auxiliary verb, when present, can indicate information about the case, number and person of subject, object and indirect object), a general solution will need at least the use of subcategorization information, unavailable at the moment. As a result, the effort devoted to the design of such a grammar would be of little final value at the moment, due to unsolved ambiguity. Furthermore, its development would also be a costly enterprise. For that reason, we decided to postpone the development of that part of the grammar until the relevant information is available.

... it was seen necessary to create an institution at this side of the Pyrennees ...

... beharrezko	*ikusi zen*	*Pirineotako bazter honetan*	*erakunde bat*	*sortzea ...*
Adjective	Verb	PP	NP	Verb
(necessary)	(was seen)	(at this side of the Pyrennees)	(an institution)	(to create)

Example 1. The attachment of elements between the two verbs needs (at least) subcategorization information.

Hence, a partial unification grammar has been developed that gives a complete coverage of the main elements of the sentence (NPs, PPs and sentential complements). At the moment the grammar contains 120 rules written in the PATR-II formalism. We chose this formalism because there has not been a broad description for Basque using more elaborated theories like LFG [Abaitua 1988] and HPSG, and also because the theories are based on information not available at the moment, such as verb subcategorization. PATR-II is more flexible at the cost of extra writing, as it is defined at a lower level. There is an average number of 15 equations per rule, some of them for testing conditions like agreement, and others for structure building. The main phenomena covered are:

- Noun phrases and prepositional phrases. Agreement among the component elements is verified, added to the proper use of determiners.
- Subordinate sentences, such as sentential complements (completive clauses, indirect questions, ...) and relative clauses.
- Simple sentences using all the previous elements. The rich agreement between the verb and the main sentence constituents (subject, object and second object) in case, number and person is verified. As we explained before, sentence analysis is performed up to the level of phenomena that can be described using only syntactic information now included in the lexicon.

Example 2 shows the (simplified) rule that combines a noun-group (noun + adjectives + determiners + noun modifiers) with a case mark (simple or composed), forming an indefinite NP or PP. Although we will not explain the rule in detail, the example shows the relative complexity of the rules as they must test for several kinds of agreement on number, definiteness and case. As a consequence, the linguistically relevant morphosyntactic information is very rich compared to most chunking systems. This will have the effect of increased flexibility, as different applications will typically use only a subset of the information.

```
rule NP_1_def
X0 ---> X1 X2
        X1/category                       <=>   noun-group
        X2/category                       <=>   case-morpheme
        X0/category                       <=>   NP
        X1/sint/agr/def                   <=>   X2/sint/agr/def
        X1/sint/agr/num                   <=>   X2/sint/agr/num
        X1/head/plu                       <=>   minus
        X2/sint/agr/case                  not   [genitive]
        X2/sint/agr/def                   <=>   indefinite
        or[X1/sint/det/head/subcategory   in    [definite, indefinite, interrogative]
           X1/head/subcategory            in    [cpronoun, interrogative-pronoun]
           X2/sint/agr/case               in    [partitive, prolative, inessive]
           X1/head/subcategory            in    [place-name, proper-name]
                                        ...
```

Example 2. Rule that combines a noun-group with a case-morpheme.

This system can be seen as a shallow parser [Abney 1997, Giguet and Vergne 1997] that can be used for subsequent processing, following "... *the basic assumption that it is possible to define an interesting intermediate level between words and sentences*", as [Basili *et al.* 1998] point out. The parser is applied bottom-up to each sentence, giving a chart as a result. The output for each sentence still contains both morphological and syntactic ambiguities, giving a huge number of different potential readings per sentence. Figure 2 shows an example where dashed lines are used to indicate lexical elements (lemmas or morphemes), while plain lines define syntactic ones. The bold circles represent word-boundaries, and the plain ones delimit morpheme-boundaries. Although the figure has been simplified, each arc is actually represented by its morphological and syntactic information, in the form of a sequence of feature-value pairs.

2.2 The Finite-State Parser

As we showed in the previous section, the unification based parser obtains the decomposition of a sentence into its main syntactic components. However, this result is not directly useful due to several reasons:

- Ambiguity. There are multiple readings for each sentence, as a result of both morphological ambiguity (1.19 interpretations per word-form) and syntactic ambiguities introduced by the unification based parser.
- Different output profiles. Linguistic information is defined at different levels, each of which will be useful depending on a particular application. For example, in the acquisition of subcategorization information all NPs, PPs and sentential complements will be necessary, but for term identification only NPs and PPs are needed (this means that sentential complements, which may include NPs and PPs, must be eliminated from the output).

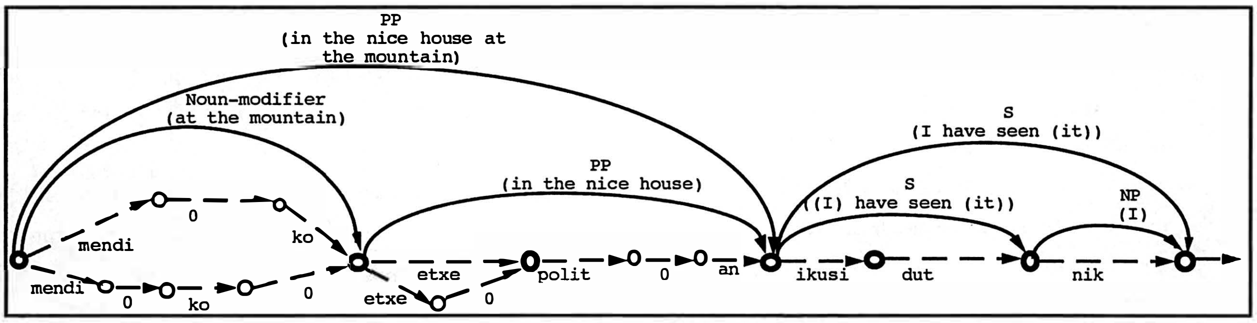


Figure 2. State of the chart after the analysis of *Mendiko etxe politean ikusi dut nik* ('I have seen (it) in the nice house at the mountain').

As a consequence, a tool is needed that will allow the definition of complex linguistic patterns for disambiguation and filtering. In recent years, several parsing systems based on finite-state technology have been developed, based on automata and transducers [Roche and Schabes 1997]. We decided to treat the resulting chart (see figure 2) as an automaton to which finite-state disambiguation constraints and filters can be applied, encoded in the form of regular expressions and relations. This way, finite-state rules provide a modular, declarative and flexible workbench to deal with the resulting chart. Currently we use the Xerox Finite State Tool (XFST, http://www.rxrc.xerox.com/research/mltt/fst/home.html), which has as its main characteristic a rich set of operations, like the replacement operator [Karttunen *et al.* 1997], defined in terms of simpler regular expressions (or relations) so that the combined expressions always belong to the finite-state calculus and can, therefore, be implemented using a finite-state automaton (transducer).

Among the finite-state operators used we apply composition, intersection and union of automata and transducers. We use both ordinary composition and the recently defined *lenient composition* [Karttunen 1998]. This operator allows the application of different eliminating constraints to a sentence, always with the certainty that when some constraint eliminates all the interpretations, then the constraint is not applied at all, that is, the interpretations are 'rescued'. The operator was first proposed to formalize Optimality Theory constraints in phonology. As Karttunen points out, it also provides a flexible way to enforce linguistic or empirical constraints in syntactic disambiguation.

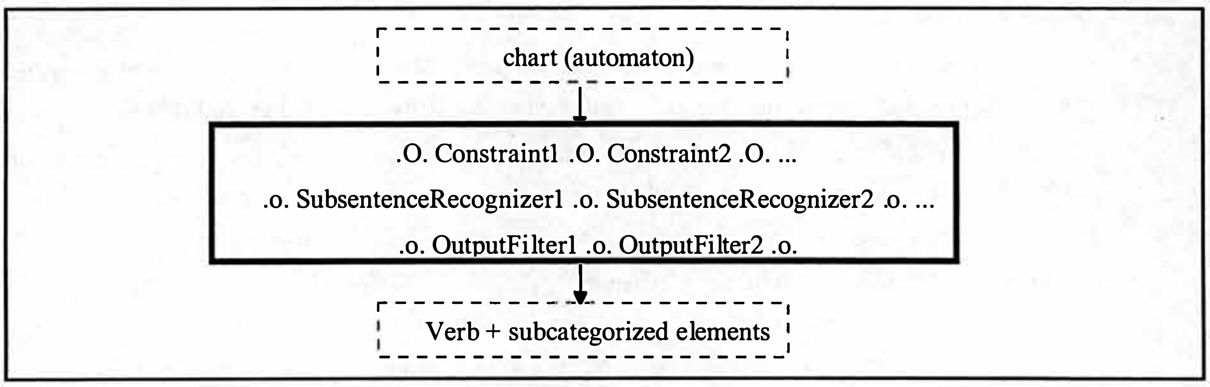


chart (automaton)
.O. Constraint1 .O. Constraint2 .O. ...
.o. SubsentenceRecognizer1 .o. SubsentenceRecognizer2 .o. ...
.o. OutputFilter1 .o. OutputFilter2 .o.
Verb + subcategorized elements

Figure 3. The finite-state parser.

The design of the finite-state rules is a non-trivial task when dealing with real texts, including proper names, syntactic/spelling errors, unknown/foreign words and a wide variety of syntactic constructions. So we had to define 388 finite-state definitions and constraints for the acquisition of verb subcategorization information (actually most of them reflect linguistic facts that can be directly used in other applications). They range from simple local constraints (305 automata with less than 100 states) to most complex patterns (there are a few automata with more than 15,000 states and 300,000 arcs).

As constraints and filters are defined by means of automata and transducers, they could theoretically be merged into a single final automaton, hence improving performance. However, as patterns are more complex, the size of the resulting automaton grows prohibitively large, so we had to arrange it by sequencing the automata. Although this slows down parsing time, it makes the compilation viable [Tapanainen 1997]. The interaction of different automata is a matter that requires further investigation.

As a first evaluation of the system, we chose the problem of acquiring verbal subcategorization information, that is, given a sentence and a verb, extracting the verb's corresponding subcategorized elements. This application has the advantage of a well defined environment to test the performance of the parser, and also that the resulting subcategorization frames may be fed back to the parser, to improve its coverage and precision [Briscoe and Carroll 1997].

These are the main operations performed by the finite-state parser (see figure 3):

- Disambiguation. As whole syntactic units can be used, this process is similar to that of Constraint Grammar disambiguation, with the advantage of being able to reference syntactic units wider than the word, which must be defined in a roundabout manner in the word-based CG formalism. As figure 3 shows, the disambiguation constraints are applied using the lenient composition operator (.O.), so that no constraint will discard all the readings of a sentence, making the system robust.
- Extracting parts of a sentence. The global ambiguity of a sentence is considerably reduced if only part of it is considered (see example 3). For instance, in the case of extracting verb subcategorization information, some rules examine the context of the target verb and define the scope of the subsentence to which the disambiguation operations will be applied (these filters use the ordinary composition operator .o.).
- Filtering. Sometimes not all the available information is relevant. For example, the *noun/adjective* ambiguity present in *zuriekin* ('with the whites' (*zuri* as a noun) / 'with the white ones' (adjective)) can be ignored when acquiring verb subcategorization information, as we are interested in the syntactic category and the grammatical case (prepositional phrase and commitative, respectively), the same in both alternatives.

Example 3 shows the application to a sentence containing the wordform *doa* (goes), in the context of analyzing the verb *joan* (to go). The result is simplified, as both the input and output are presented as text, rather than as an automaton containing feature-value pairs that represent syntactic components (the translation of the output subsentence is given later in example 4).

Eta lepo honetatik Narbajara bidegarbia doa Negurako pagaditik
Larraingoitiko hareharrizko harrobitik zehar igaroaz

$\downarrow$

(lepo honetatik) (Narbajara) (bide garbia) (doa)

Example 3. Simplified input and output of the parser.

3 Evaluation

We took a corpus consisting of 500 sentences corresponding to 5 verbs, that is, 100 sentences per verb. In order to test different corpora, half of the sentences were taken from a general corpus of Basque, while the other half came from newspaper texts. We manually marked for each sentence the occurrence of each target verb and its associated subcategorized elements, and then compared it with the output of the parser. 350 sentences were used for the refinement of both the unification based parser and the finite-state parser, while the remaining set of 150 sentences (30 for each verb) was only examined for the final test.

Regarding the tested sentences, we did not select them by lexical or syntactic coverage of the parser, i. e., we took the first set of 500 sentences containing the target verbs from the two corpora, so that we could measure the actual performance of the parser with unrestricted texts. There are several extra difficulties added to the problem of ambiguity:

- Sentence length. Each sentence contains an instance of the target verb together with other main or subordinate subsentences (the average sentence length is 22 words). Delimiting the exact boundaries of the subsentence corresponding to the target verb is a difficult task.
- Multiword lexical units. Although we plan to include their treatment in the morphological analysis phase, it is not implemented yet. Its main consequence will be an increase in the number of errors (false positives), as some non-compositional elements will be interpreted compositionally by the general unification based grammar.
- Unknown words, proper names and spelling errors. Although the morphological analyzer recognizes a subset of them, the rest is problematic because each of them will give a number of hypothetical interpretations, therefore increasing ambiguity and consequently the error rate. Increasing lexical coverage will have a positive impact in future developments.

To evaluate the correct analysis of a sentence, we have developed a simple coding scheme inspired on [Carroll *et al.* 1999], who define a hierarchy of grammatical relations. Instead of marking syntactic functions, we annotate the declension case, lemma and number of NPs and PPs [Oesterle and Maier-Meyer 1998], and the subordination type for sentential complements (see example 4). We have postponed the assignment of syntactic functions until relevant data is available.

Input: ... *lepo honetatik Narbajara bide garbia doa* ... (... a clear path goes from this hill to Narbaja ...)				
Output:	*lepo honetatik*	*Narbajara*	*bide garbia*	*doa*
	ablative(lepo, doa)	*alative(Narbaja, doa)*	*absolutive(bide, doa)*	*doa*
	from this hill	to Narbaja	a clear path	goes
	ablative(hill, goes)	alative(Narbaja, goes)	absolutive(path, goes)	goes

Example 4. Coding scheme based on the grammatical case of subcategorized elements.

For evaluation we measured precision (the number of correctly selected elements / all the elements returned by the parser) and recall (the number of correctly selected elements / all the elements present in the sentence). Table 1 shows the results as the mean over all sentences. Although there is always a balance between recall and precision, we tried to maximize the latter, sometimes at the cost of lowering recall. As we could inspect the development corpus during the refinement of the parser, the results in the second and third columns can be understood as an upper limit of the parser in its current state, approximately 92% precision and 71% recall. As we will explain next, these results can be improved refining the lexicon and the grammars.

	Development corpus (350 sentences)		**Test corpus (150 sentences)**	
	Precision	**Recall**	**Precision**	**Recall**
agertu (to appear)	95%	69%	87%	62%
atera (to go)	91%	65%	92%	64%
erabili (to use)	92%	70%	86%	55%
ikusi (to see)	91%	76%	87%	78%
joan (to go)	93%	74%	83%	70%
Total	92%	71%	87%	66%

Table 1. Evaluation results.

We examined manually the causes of the errors (68 errors were identified in the test corpus causing problems in precision or recall), which can be classified into several types[1]:

- Errors due to multiword units (5), unknown words, proper names (9) and spelling errors (8). Their treatment corresponds naturally to morphological analysis and is mainly linked to future extensions of the lexicon.
- Errors due to incorrect disambiguation. They can be subdivided into two main types. When the morphological disambiguator chooses an incorrect reading, it has the effect of causing a false positive, i.e., decreasing precision (9 errors). On the other hand, sometimes more than one alternative is left (including the correct one). Its main effect will be increased ambiguity that will show as lower recall (5 errors).
- Errors due to the lack of syntactic coverage of the grammars (32 errors). This kind of errors define the limits of the partial parsing approach. Although more than half of these errors are due to the incompleteness of the grammar, and they can be solved simply by extending it to cover the

[1] We did not examine the relationships among different errors, as many times one kind of error has the effect of causing other error types.

corresponding phenomena, there are other errors that would need qualitative changes, like the inclusion of subcategorization information. Finally, a third set of errors are due to the characteristics of unrestricted corpora, such as syntactically odd constructions, that we doubt a parser could analyze even after solving the two other problems.

As the results show, more than half of the errors could be solved by improvements on the lexicon, the morphological analyzer and morphological disambiguation (totaling 36 errors). Although morphological disambiguation is relatively difficult to extend and modify, further careful work extending the treatment of proper names, spelling errors and multiword units would imply a noticeable increase in both recall and precision. In a similar way, work must be done extending the basic syntactic grammar, which we estimate could reduce the syntactic errors to about a half of the present ones. Consequently, we consider the results satisfactory, with 87% precision and 66% recall, as the results for new sentences are near the expected best results (those obtained for the development corpus, with 92% precision and 71% recall), showing that the system behaves correctly with unseen sentences.

After obtaining instances of putative subcategorization frames, there is still work to be done. In configurational languages like English, subcategorized elements appear at fixed places around the verb, while in nonconfigurational ones they can appear at several different positions (hypothetically all the permutations are possible). Basque being mainly a nonconfigurational and pro-drop language with respect to phrases in ergative, absolutive and dative cases, there is not a direct correspondence between subcategorization instances and frames, as one subcategorization frame may correspond to several kinds of instances. As an experiment, we applied the parser to 1,000 sentences (22,000 words) corresponding to the verb *ikusi* ('to see'), and classified the results according to the different sets of subcategorized elements, without taking their relative order in consideration. The results are presented in Table 2. Results were obtained for 826 sentences, after discarding those having more than one interpretation. For example, the patterns 'instrumental' and 'absolutive instrumental' correspond to the same subcategorization frame, due to pro-drop phenomena with the phrase in the absolutive case. The possibility of automatically classifying subcategorization patterns into frames deserves further work.

Subcategorization pattern	**# of occurrences**
absolutive	206
inessive	59
inessive absolutive	42
ergative	36
instrumental	14
ergative absolutive	11
absolutive instrumental	6
absolutive inessive ergative	4

Table 2. Different patterns found in the corpus.

4 Related Work

[Abney 1997, Giguet and Vergne 1997, Basili *et al.* 1998] show the benefits of a stratified approach to parsing, where one or more intermediate levels can be defined between the basic level of words and the analysis of a full sentence. Our work differs from theirs in that we apply two different kinds of analyzers (unification based and finite-state), rather than defining the different levels using the same formalism.

[Ritchie *et al.* 1992] present a system that performs morphological analysis, divided in a segmentation phase (using finite state networks) and the application of a unification grammar for the combination of morphemes. The results of the segmentation are interpreted as a chart that serves as input to a unification based chart parser. Our system shares the idea of dividing work between different kinds of formalism. However, our approach differs in that we first apply a unification grammar, indispensable for the treatment of complex syntactic phenomena, and then a finite state grammar is used for disambiguation and filtering.

Regarding the problem of syntactic disambiguation, most grammar based systems [Briscoe and Carroll 1997] have adopted a statistical approach. For morphological disambiguation, however, there are both statistical and rule based systems, with better results for the second approach [Samuelsson and Voutilainen 1997]. Our system adopts a rule formalism based on regular expressions, using syntactic elements instead of words as the basic disambiguation unit. We justify this election on both the unavailability of syntactically annotated treebanks and the better performance of systems based on hand-coded rules.

Concerning the acquisition of verb subcategorization information, there are proposals ranging from manual examination of corpora [Grishman *et al.* 1994] to fully automatic approaches. [Briscoe and Carroll 1997] describe a grammar based experiment for the extraction of subcategorization frames with their associated relative frequencies, obtaining 76.6% precision and 43.4% recall. Our results are not directly comparable, as we only estimate precision and recall on subcategorization instances, not frames.

[Kuhn *et al.* 1998] compare two approaches for the acquisition of subcategorization information: a corpus query pattern based approach (no grammar, using regular expressions on morphologically analyzed wordforms) and a grammar based approach (in a way similar to [Briscoe and Carroll 1997]). Both are applied to the problem of acquiring subcategorization instances of 3 subcategorization frames, showing that the grammar based approach improves results specially in recall, due mainly to the higher-level knowledge encoded in the grammar. Comparing with our work, we think that our system is situated between the two approaches, as we use patterns on partially parsed sentences. Our objective is more ambitious in the sense that we try to find all the subcategorization instances, rather than distinguishing among 3 previously selected frames.

The above mentioned studies depend on a set of manually annotated analyses. [Carroll and Rooth 1998] present a learning technique for subcategorization frames based on a probabilistic lexicalized grammar and the *Expectation Maximization* algorithm using unmarked corpora. The results are promising, although the method is still computationally expensive and requires big corpora (50M).

5 Conclusion

This work presents the development of a robust parser for unrestricted Basque texts. As the linguistic resources are limited, the lexicon lacks important aspects such as verbal subcategorization information. We have implemented a basic syntactic parser using the information now available in the lexicon. The system has been divided in two sequential modules:

- A unification based grammar that covers the main sentence components of the sentence. It gives a description of well-formed linguistic phenomena. Due to the agglutinative nature of the language, feature structures are necessary to treat the wealth of information contained in words/morphemes.
- Finite-state rules that provide a modular, declarative and flexible workbench to deal with the resulting chart of syntactic elements. It establishes the application of empirical, corpus-oriented facts, versus the more general facts on linguistic well-formedness encoded in the unification grammar.

The unification based grammar and the finite-state one are complementary. The unification grammar is necessary to treat aspects like complex agreement and constituent order variations, currently unsolvable using finite-state networks, due to the exponential growth in size of the resulting automata [Beesley 1998]. The limits of this grammar are mainly defined by the unavailability of important information, like subcategorization frames. On the other hand, regular expressions and relations, in the form of automata and transducers, are indispensable to cope with morphological/syntactic ambiguity (by means of hand-coded rules or constraints) and filtering of the information relevant to each application, thus adding to the flexibility of the resulting tool.

The parser is being used in the process of acquiring verb subcategorization instances, obtaining 87% precision and 66% recall over a corpus of previously unseen 150 sentences. In future work, we plan to integrate the resulting subcategorization information into the grammar, so that it will be extended by successive bootstrapping cycles.

Acknowledgements

This research is supported by the Basque Goverment, the University of the Basque Country and the Interministerial Commision for Science and Technology (CICYT). Thanks to Gorka Elordieta for his help writing the final version of the paper.

Bibliography

[Abaitua 1988] Abaitua, J. 1988. Complex predicates in Basque: from lexical forms to functional structures. PhD thesis, University of Manchester.

[Abney 1997] Abney S. P. 1997. Part-of-Speech Tagging and Partial Parsing. in Corpus-Based Methods in Language and Speech Processing, Kluwer, Dordrecht.

[Aldezabal *et al.* 1999] Aldezabal I., Gojenola K., Oronoz M. 1999. Combining Chart-Parsing and Finite State Parsing. Proceedings of the ESSLLI'99 Student Session, Utrecht.

[Alegria *et al.* 1996] Alegria I., Artola X., Sarasola K., Urkia M. 1996. Automatic morphological analysis of Basque. Literary and Linguistic Computing 11 (4).

[Basili *et al.* 1998] Basili, R., Pazienza M.T., Zanzotto F.M. 1998. Efficient Parsing for Information Extraction. Proceedings of the 13th European Conference on Artificial Intelligence, John Wiley & Sons Ltd.

[Beesley 1998] Beesley K. 1998. Constraining Separate Morphotactic Dependencies in Finite-State Grammars. Proceedings of the International Workshop on Finite State Methods in Natural Language Processing, Ankara.

[Briscoe and Carroll 1993] Briscoe T., Carroll J. 1993. Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. Computational Linguistics, vol. 19(1).

[Briscoe and Carroll 1997] Briscoe T., Carroll J. 1997. Automatic Extraction of Subcategorization from Corpora. ANLP'97, Washington.

[Butt *et al.* 1999] Butt M., King T.H., Nino M.E., Segond F. 1999 A Grammar Writer's Cookbook. Stanford, CA: CSLI Lecture Notes, CSLI Publications, 1999.

[Carroll and Rooth 1998] Carroll G., Rooth M. 1998. Valence Induction with a Head-Lexicalized PCFG. Proceedings of the Conference on Empirical Methods in Natural Language Processing, Granada.

[Carroll *et al.* 1999] Carroll J, Minnen G., Briscoe T. 1999. Corpus Annotation for Parser Evaluation. Proceedings of Workshop on Linguistically Interpreted Corpora, EACL'99, Bergen.

[Ezeiza *et al.* 1998] Ezeiza N., Alegria I., Arriola J.M., Urizar R., Aduriz I., 1998. Combining Stochastic and Rule-Based Methods for Disambiguation in Agglutinative Languages. COLING-ACL'98, Montreal.

[Giguet and Vergne 1997] Giguet E., Vergne J. 1997. From Part of Speech Tagging to Memory-based Deep Syntactic Analysis. Fifth International Workshop on Parsing Technologies, Boston.

[Grishman *et al.* 1994] Grishman R., Macleod C., Meyers A. 1994. Comlex Syntax: Building a Computational Lexicon. COLING'94, Japan.

[Hajic and Hladká 1998] Hajic J., Hladká B. 1998. Tagging Inflective Languages: Prediction of Morphological Categories for a Rich, Structured Tagset. COLING-ACL'98, Montreal.

[Karttunen *et al.* 1997] Karttunen L., Chanod J-P., Grefenstette G., Schiller A. 1997. Regular Expressions For Language Engineering. Natural Language Engineering.

[Karttunen 1998] Karttunen L. 1998. The Proper Treatment of Optimality in Computational Phonology. Proceedings of the International Workshop on Finite State Methods in Natural Language Processing, Ankara.

[Kuhn *et al.* 1998] Kuhn J., Eckle-Kohler J., Rohrer. C. 1998. Lexicon Acquisition with and for Symbolic NLP-Systems -- a Bootstrapping Approach. Int. Conference on Language Resources and Evaluation (LREC98), Granada.

[Oesterle and Maier-Meyer 1998] Oesterle J., Maier-Meyer P. 1998. The GNoP (German Noun Phrase) Treebank. LREC98, Granada.

[Oflazer 1999] Oflazer K. 1999. Dependency Parsing with an Extended Finite State Approach. ACL'99, Maryland.

[Prószéky 1996] Prószéky G. 1996. Morphological Analyzer as Syntactic Parser. COLING'96, Copenhagen.

[Ritchie *et al.* 1992] Ritchie G., Russel G. J., Black A., W., Pullman S. G. 1992. Computational Morphology: Practical Mechanisms for the English Lexicon. The MIT Press.

[Roche and Schabes 1997] Roche R., Schabes Y. 1997. Finite-State Language Processing. MIT Press.

[Samuelsson and Voutilainen 1997] Samuelsson C., Voutilainen A. 1997. Comparing a Linguistic and a Stochastic Tagger. ACL-EACL'97, Madrid.

[Tapanainen 1997] Tapanainen P. 1997. Applying a Finite-State Intersection Grammar. Finite-State Language Processing, MIT Press.

NEW TABULAR ALGORITHMS FOR LIG PARSING

Miguel A. Alonso
Jorge Graña
Manuel Vilares
Departamento de Computación
Universidad de La Coruña
Campus de Elviña s/n
15071 La Coruña (Spain)
{alonso,grana,vilares}@dc.fi.udc.es

Eric de la Clergerie
INRIA
Domaine de Voluceau
Rocquecourt, B.P. 105
78153 Le Chesnay (France)
Eric.De_La_Clergerie@inria.fr

Abstract

We develop a set of new tabular parsing algorithms for Linear Indexed Grammars, including bottom-up algorithms and Earley-like algorithms with and without the valid prefix property, creating a continuum in which one algorithm can in turn be derived from another. The output of these algorithms is a shared forest in the form of a context-free grammar that encodes all possible derivations for a given input string.

1 Introduction

Tree Adjoining Grammars (TAG) [8] and Linear Indexed Grammars (LIG) [7] are extensions of Context Free Grammars (CFG). Tree adjoining grammars use trees instead of productions as primary representing structure and seems to be adequate to describe syntactic phenomena occurring in natural language, due to their extended domain of locality and to their ability for factoring recursion from the domain of dependencies. Linear indexed grammars associate a stack of indices with each non-terminal symbol, with the restriction that the indices stack of the head non-terminal of each production (the *father*) can be inherited by at most one body non-terminal (the *dependent child*) while the other stacks must have a bounded stack size.

Several parsing algorithms have been proposed for TAG, ranging from simple bottom-up algorithms, like CYK [17], to sophisticated extensions of the Earley's algorithm [9]. In order to improve efficiency, it is usual to translate the source tree adjoining grammar into a linear indexed grammar [16, 12, 13, 17]. However, in some cases is not possible to translate the parsing strategy from TAG to LIG, as there are parsing strategies for TAG which are not incorporated in any parsing algorithm for LIG. To eliminate this drawback, we present in this paper several parsing algorithms for LIG which mimic the most popular parsing strategies for TAG [1].

1.1 Linear Indexed Grammars

A linear indexed grammar is a tuple (V_T, V_N, V_I, P, S), where V_T is a finite set of terminals, V_N a finite set of non-terminals, V_I is a finite set of indices, $S \in V_N$ is the start symbol and P is a finite set of productions. Following [7] we consider productions in which at most one element can be pushed on or popped from a stack of indices:

$$A_0[\circ\circ\gamma] \rightarrow A_1[\,]\ldots A_{i-1}[\,]\; A_i[\circ\circ\gamma']\; A_{i-1}[\,]\;\ldots A_m[\,]$$

$$A_0[\,] \rightarrow a$$

where m is the length of the production, $A_j \in V_N$ for each $0 \leq j \leq m$, A_i is the dependent child, $\circ\circ$ is the part of the indices stack transmitted from the father to the dependent child, $\gamma, \gamma' \in V_I \cup \{\epsilon\}$ and for each production either γ or γ' or both must be ϵ and $a \in V_T \cup \{\epsilon\}$.

The derivation relation $\Rightarrow$ is defined for LIG as $\Upsilon \Rightarrow \Upsilon'$

- if $\Upsilon = \Upsilon_1 A[\alpha\gamma]\; \Upsilon_4$ and there exists a production $A[\circ\circ\gamma] \rightarrow \Upsilon_2 A'[\circ\circ\gamma']\; \Upsilon_3$ such that $\Upsilon' = \Upsilon_1\Upsilon_2 A'[\alpha\gamma']\; \Upsilon_3\Upsilon_4$
- or else if $\Upsilon = \Upsilon_1 A[\,]\; \Upsilon_4$ and there exists a production $A[\,] \rightarrow a$ such that $\Upsilon' = \Upsilon_1\; a\; \Upsilon_4$

where $A \in V_N$, $\alpha \in V_I^*$ and $\gamma, \gamma' \in V_I \cup \{\epsilon\}$. The reflexive and transitive closure of $\Rightarrow$ is denoted by $\overset{*}{\Rightarrow}$. The language defined by a LIG is the set of strings $w \in V_T^*$ such that $S[\,] \overset{*}{\Rightarrow} w$.

To parse this type of grammars, tabulation techniques with polynomial complexity can be designed based on a property defined in [17], that we call *context-freeness property of LIG*, establishing that if $A[\gamma] \overset{*}{\Rightarrow} uB[\,]w$ where $u, w \in V_T^*$, $A, B \in V_N$, $\gamma \in V_I \cup \{\epsilon\}$ and $B[\,]$ is a dependent descendant of $A[\gamma]$, then for each $\Upsilon_1, \Upsilon_2 \in (V_N[V_I^*] \cup V_T)^*$and $\beta \in V_I^*$ we have $\Upsilon_1 A[\beta\gamma]\Upsilon_2 \overset{*}{\Rightarrow} \Upsilon_1 uB[\beta]w\Upsilon_2$. Also, if $B[\gamma]$ is a dependent descendant of $A[\,]$ and $A[\,] \overset{*}{\Rightarrow} uB[\gamma]w$ then $\Upsilon_1 A[\beta]\Upsilon_2 \overset{*}{\Rightarrow} \Upsilon_1 uB[\beta\gamma]w\Upsilon_2$.

1.2 Parsing Schemata

We will describe parsing algorithms using *Parsing Schemata*, a framework for high-level description of parsing algorithms [15]. An interesting application of this framework is the analysis of the relations between different parsing algorithms by studying the formal relations between their underlying parsing schemata.

A *parsing system* for a grammar G and string $a_1 \ldots a_n$ is a triple $\langle \mathcal{I}, \mathcal{H}, \mathcal{D} \rangle$, with $\mathcal{I}$ a set of *items* which represent intermediate parse results, $\mathcal{H}$ an initial set of items called *hypothesis* that encodes the sentence to be parsed, and $\mathcal{D}$ a set of *deduction steps* that allow new items to be derived from already known items. Deduction steps are of the form $\frac{\eta_1, \ldots, \eta_k}{\xi}\; cond$, meaning that if all antecedents η_i of a deduction step are present and the conditions *cond* are satisfied, then the consequent ξ should be generated by the parser. A set $\mathcal{F} \subseteq \mathcal{I}$ of *final items* represent the recognition of a sentence. A *parsing schema* is a parsing system parameterized by a grammar and a sentence.

Parsing schemata are closely related to *grammatical deduction systems* [14], where items are called *formula schemata*, deduction steps are *inference rules*, hypothesis are *axioms* and final items are *goal formulas*.

2 A CYK-like Algorithm

We have chosen the CYK-like algorithm for LIG described in [16] as our starting point. Due to the intrinsic limitations of this pure bottom-up algorithm, the grammars it can deal with are restricted to those having two elements, or one element which must be a terminal, in the right-hand side of each production. This restriction could be considered as the transposition of the Chomsky normal form to linear indexed grammars.

The algorithm works by recognizing in a bottom-up way the part of the input string spanned by each grammar element. The items used in the tabular interpretation of this algorithm are of the form $[A, \gamma, i, j \mid B, p, q]$ and represent one of the following types of derivation:

- $A[\gamma] \overset{*}{\Rightarrow} a_{i+1} \dots a_p \; B[\,] \; a_{q+1} \dots a_j$ if and only if $(B,p,q) \neq (-,-,-)$, $B[\,]$ is a dependent descendent of $A[\gamma]$ and $(p,q) \leq (i,j)$.

- $A[\,] \overset{*}{\Rightarrow} a_{i+1} \dots a_j$ if and only if $\gamma = -$ and $(B,p,q) = (-,-,-)$.

where $-$ means that the value of a component is not bound and $(p,q) \leq (i,j)$ is used to represent that $i \leq p \leq q \leq j$ when p and q are bound.

These items are like those proposed for the tabulation of linear indexed automata [10] and for the tabulation of bottom-up 2–stack automata [6]. They are slightly different from the items of the form $[A, \gamma, i, j \mid B, \eta, p, q]$ proposed by Vijay-Shanker and Weir in [16] for their CYK-like algorithm, where the element $\eta \in V_I$ is redundant: due to the context-freeness property of LIG we have that if $A[\gamma] \overset{*}{\Rightarrow} a_{i+1} \dots a_p \; B[\,] \; a_{q+1} \dots a_j$ then for any β we have that $A[\beta\gamma] \overset{*}{\Rightarrow} a_{i+1} \dots a_p \; B[\beta] \; a_{q+1} \dots a_j$.

Schema 1 *The parsing system $\mathbb{P}_{\text{CYK}}$ corresponding to the CYK-like parsing algorithm for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \dots a_n$ is defined as follows:*

$$\mathcal{I}_{\text{CYK}} = \{ \; [A, \gamma, i, j \mid B, p, q] \mid A, B \in V_N, \; \gamma \in V_I, \; 0 \leq i \leq j \,, \; (p,q) \leq (i,j) \; \}$$

$$\mathcal{H}_{\text{CYK}} = \{ \; [a, i-1, i] \mid a = a_i, \; 1 \leq i \leq n \; \}$$

$$\mathcal{D}^{\text{Scan}}_{\text{CYK}} = \frac{[a, j, j+1]}{[A, -, j, j+1 \mid -, -, -]} \quad A[\,] \to a \in P$$

$$\mathcal{D}^{[\circ\circ\gamma][\,][\circ\circ]}_{\text{CYK}} = \frac{\begin{array}{c}[B, -, i, k \mid -, -, -], \\ [C, \eta, k, j \mid D, p, q]\end{array}}{[A, \gamma, i, j \mid C, k, j]} \quad A[\circ\circ\gamma] \to B[\,] \; C[\circ\circ] \in P$$

$$\mathcal{D}^{[\circ\circ\gamma][\circ\circ][\,]}_{\text{CYK}} = \frac{\begin{array}{c}[B, \eta, i, k \mid D, p, q], \\ [C, -, k, j \mid -, -, -]\end{array}}{[A, \gamma, i, j \mid B, i, k]} \quad A[\circ\circ\gamma] \to B[\circ\circ] \; C[\,] \in P$$

$$\mathcal{D}^{[\circ\circ][\,][\circ\circ]}_{\text{CYK}} = \frac{\begin{array}{c}[B, -, i, k \mid -, -, -], \\ [C, \eta, k, j \mid D, p, q]\end{array}}{[A, \eta, i, j \mid D, p, q]} \quad A[\circ\circ] \to B[\,] \; C[\circ\circ] \in P$$

$$\mathcal{D}^{[\circ\circ][\circ\circ][\,]}_{\text{CYK}} = \frac{\begin{array}{c}[B, \eta, i, k \mid D, p, q], \\ [C, -, k, j \mid -, -, -]\end{array}}{[A, \eta, i, j \mid D, p, q]} \quad A[\circ\circ] \to B[\circ\circ] \; C[\,] \in P$$

$$\mathcal{D}^{[\circ\circ][\,][\circ\circ\gamma]}_{\text{CYK}} = \frac{\begin{array}{c}[B, -, i, k \mid -, -, -], \\ [C, \gamma, k, j \mid D, p, q], \\ [D, \eta, p, q \mid E, r, s]\end{array}}{[A, \eta, i, j \mid E, r, s]} \quad A[\circ\circ] \to B[\,] \; C[\circ\circ\gamma] \in P$$

$$\mathcal{D}^{[\circ\circ][\circ\circ\gamma][\,]}_{\text{CYK}} = \frac{\begin{array}{c}[B, \gamma, i, k \mid D, p, q], \\ [C, -, k, j \mid -, -, -], \\ [D, \eta, p, q \mid E, r, s]\end{array}}{[A, \eta, i, j \mid E, r, s]} \quad A[\circ\circ] \to B[\circ\circ\gamma] \; C[\,] \in P$$

$$\mathcal{D}_{\text{CYK}} = \mathcal{D}^{\text{Scan}}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ\gamma][\,][\circ\circ]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ\gamma][\circ\circ][\,]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\,][\circ\circ]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\circ\circ][\,]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\,][\circ\circ\gamma]}_{\text{CYK}} \cup \mathcal{D}^{[\circ\circ][\circ\circ\gamma][\,]}_{\text{CYK}}$$

$$\mathcal{F}_{\text{CYK}} = \{ \; [S, -, 0, n \mid -, -, -] \; \}$$

The hypotheses defined for this parsing system are the standard ones and therefore they will be omitted in the remaining parsing systems described in this paper.

Steps in the set $\mathcal{D}_{\text{CYK}}^{\text{Scan}}$ are in charge of starting the parsing process. The other steps are in charge of combining the items corresponding to the elements in the right-hand side of a production in order to generate the item corresponding to the left-hand side element, propagating bottom-up the information about the indices stack.

The space complexity of the algorithm with respect to the length n of the input string is $\mathcal{O}(n^4)$, as each item stores four positions of the input string. The time complexity is $\mathcal{O}(n^6)$ and it is given by the deduction steps in $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma]\text{right}}$ and $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma]\text{left}}$. Although these steps involve 7 positions of the input string, by partial application each step can be decomposed in a set of deduction steps involving at most 6 positions.

A CYK-like algorithm generalized for linear indexed grammar with productions manipulating more than one symbol at the top of the indices stacks is described in [17]. The same kind of generalization could be incorporated into the algorithm presented here.

3 A Bottom-up Earley-like Algorithm

The CYK-like algorithm has an important limitation: it can only be applied to linear indexed grammars having at most two children in the right-hand side. To overcome this limitation we use dotted production into items instead single nodes. Thus, we can distinguish the part of a production already processed from the part not yet processed. With respect to notation, we use $\boldsymbol{A}$ to denote a grammar element having the non-terminal A when the associated indices stack is not relevant in that context.

The items of the new parsing system $\mathbb{P}_{\text{buE}}$ are of the form $[\boldsymbol{A} \to \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B, p, q]$ and can be obtained by refining the items in $\mathbb{P}_{\text{CYK}}$. They represent one of the following two cases:

- $A[\gamma] \Rightarrow \Upsilon_1\Upsilon_2 \overset{*}{\Rightarrow} a_{i+1} \ldots a_p \; B[\,] \; a_{q+1} \ldots a_j \; \Upsilon_2$ if and only if $(B,p,q) \neq (-,-,-)$, $B[\,]$ is a dependent descendant of $A[\gamma]$ and $(p,q) \leq (i,j)$.
- $\Upsilon_1 \overset{*}{\Rightarrow} a_{i+1} \ldots a_j$ if and only if $\gamma = -$ and $(B,p,q) = (-,-,-)$. If the dependent child is in Υ_1 then the indices stack associated to $\boldsymbol{A}$ and to the dependent child must be empty.

The set of deduction steps of $\mathbb{P}_{\text{buE}}$ is obtained by refining the steps in $\mathbb{P}_{\text{CYK}}$: steps $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma][\,][\circ\circ]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ\gamma][\circ\circ][\,]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\,][\circ\circ]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\circ\circ][\,]}$, $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\,][\circ\circ\gamma]}$ and $\mathcal{D}_{\text{CYK}}^{[\circ\circ][\circ\circ\gamma][\,]}$ are separated into different steps Init and Comp. Finally, the domain is extended to deal with linear indexed grammars having productions of arbitrary length.

Schema 2 *The parsing system $\mathbb{P}_{\text{buE}}$ corresponding to the bottom-up Earley-like parsing algorithm for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$ is defined as follows:*

$$\mathcal{I}_{\text{buE}} = \left\{ \begin{array}{ll} [\boldsymbol{A} \to \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B, p, q] \mid & \boldsymbol{A} \to \Upsilon_1\Upsilon_2 \in P,\ B \in V_N,\ \gamma \in V_I, \\ & 0 \leq i \leq j\ ,\ \ (p,q) \leq (i,j) \end{array} \right\}$$

$$\mathcal{D}_{\text{buE}}^{\text{Init}} = \frac{}{[\boldsymbol{A} \to \bullet\Upsilon, -, i, i \mid -,-,-]}$$

$$\mathcal{D}_{\text{buE}}^{\text{Scan}} = \frac{\begin{array}{c}[A[\,] \to \bullet a, -, j, j \mid -,-,-], \\ [a, j, j+1]\end{array}}{[A[\,] \to a\bullet, -, j, j+1 \mid -,-,-]}$$

$$\mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\,]}} = \frac{\begin{array}{l}[A \to \Upsilon_1 \bullet B[\,]\,\Upsilon_2, \gamma, i, k \mid C, p, q],\\ [B \to \Upsilon_3 \bullet, -, k, j \mid -, -, -]\end{array}}{[A \to \Upsilon_1\, B[\,] \bullet \Upsilon_2, \gamma, i, j \mid C, p, q]}$$

$$\mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ\gamma][\circ\circ]}} = \frac{\begin{array}{l}[A[\circ\circ\gamma] \to \Upsilon_1 \bullet B[\circ\circ]\,\Upsilon_2, -, i, k \mid -, -, -],\\ [B \to \Upsilon_3 \bullet, \eta, k, j \mid C, p, q]\end{array}}{[A[\circ\circ\gamma] \to \Upsilon_1\, B[\circ\circ] \bullet \Upsilon_2, \gamma, i, j \mid B, k, j]}$$

$$\mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ][\circ\circ]}} = \frac{\begin{array}{l}[A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ]\,\Upsilon_2, -, i, k \mid -, -, -],\\ [B \to \Upsilon_3 \bullet, \eta, k, j \mid C, p, q]\end{array}}{[A[\circ\circ] \to \Upsilon_1\, B[\circ\circ] \bullet \Upsilon_2, \eta, i, j \mid C, p, q]}$$

$$\mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ][\circ\circ\gamma]}} = \frac{\begin{array}{l}[A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ\gamma]\,\Upsilon_2\,, -, i, k \mid -, -, -],\\ [B \to \Upsilon_3 \bullet\,, \gamma, k, j \mid C, p, q],\\ [C \to \Upsilon_4 \bullet, \eta, p, q \mid D, r, s]\end{array}}{[A[\circ\circ] \to \Upsilon_1\, B[\circ\circ\gamma] \bullet \Upsilon_2, \eta, i, j \mid D, r, s]}$$

$$\mathcal{D}_{\mathrm{buE}} = \mathcal{D}_{\mathrm{buE}}^{\mathrm{Init}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Scan}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\,]}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ\gamma][\circ\circ]}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ][\circ\circ]}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ][\circ\circ\gamma]}}$$

$$\mathcal{F}_{\mathrm{buE}} = \{\ [S \to \Upsilon \bullet\,, -, 0, n \mid -, -, -]\ \}$$

The space complexity with respect to the input string is $\mathcal{O}(n^4)$ because each item stores four positions. The time complexity with respect to the input string is $\mathcal{O}(n^6)$ and it is given by deduction steps in $\mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ][\circ\circ\gamma]}}$.

4 An Earley-like Algorithm

The algorithm described by the parsing system $\mathbb{P}_{\mathrm{buE}}$ does not take into account whether the part of the input string recognized by each grammar element can be derived from S, the axiom of the grammar. Earley-like algorithms limit the number of deduction steps that can be applied in each moment by predicting productions which are candidates to be part of a derivation having as its starting point the axiom of the grammar.

As a first approach, we consider prediction is performed taking into account the context-free skeleton only. The parsing system so obtained is denoted $\mathbb{P}_{\mathrm{E}}$ and can be derived from $\mathbb{P}_{\mathrm{buE}}$ by applying the following filters:

- Init deduction steps only consider productions with S as left-hand side.
- Instead of generating items of the form $[A \to \bullet\Upsilon, -, i, i \mid -, -, -]$ for each possible production $A \to \Upsilon \in P$ and positions i and j, a set of Pred deduction steps generate only those items involving productions with a relevant context-free skeleton.

Schema 3 *The parsing system $\mathbb{P}_{\mathrm{E}}$ corresponding to the Earley-like parsing algorithm for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \dots a_n$ is defined as follows:*

$$\mathcal{I}_{\mathrm{E}} = \mathcal{I}_{\mathrm{buE}}$$

$$\mathcal{D}_{\mathrm{E}}^{\mathrm{Init}} = \frac{}{[S \to \bullet\Upsilon, -, 0, 0 \mid -, -, -]}$$

$$\mathcal{D}_{\mathrm{E}}^{\mathrm{Pred}} = \frac{[A \to \Upsilon_1 \bullet B\, \Upsilon_2, \gamma, i, j \mid C, p, q]}{[B \to \bullet\Upsilon_3, -, j, j \mid -, -, -]}$$

$$\mathcal{D}_{\mathrm{E}} = \mathcal{D}_{\mathrm{E}}^{\mathrm{Init}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Scan}} \cup \mathcal{D}_{\mathrm{E}}^{\mathrm{Pred}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\,]}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ\gamma][\circ\circ]}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ][\circ\circ]}} \cup \mathcal{D}_{\mathrm{buE}}^{\mathrm{Comp[\circ\circ][\circ\circ\gamma]}}$$

$$\mathcal{F}_{\mathrm{E}} = \mathcal{F}_{\mathrm{buE}}$$

This algorithm, which has a space complexity $\mathcal{O}(n^4)$ and a time complexity $\mathcal{O}(n^6)$, is very close to the Earley-like algorithm described by Schabes and Shieber in [13] although the latter can only be applied to a specific class of linear indexed grammars obtained from tree adjoining grammars. However, both algorithms share an important feature: they are weakly predictive as they do not consider the contents of the indices stacks when predictive steps are applied. In appearance, the algorithm proposed by Schabes and Shieber in [13] consults the element on the top of the indices stack at prediction, but a deeper study of the behavior of the algorithm makes it clear that this is not really true, as the authors store the context-free skeleton of the elementary trees of a TAG into the indices stacks, reducing the non-terminal set of the resulting LIG to $\{t, b\}$. Indeed, a production $b[\circ\circ\eta] \to t[\eta_1] \ldots t[\circ\circ\eta_s] \ldots t[\eta_n]$ is equivalent to $\eta^b[\circ\circ] \to \eta_1^t[\,] \ldots \eta_s^t[\circ\circ] \ldots \eta_n^t[\,]$ and a production $b[\circ\circ\eta] \to t[\eta_1] \ldots t[\eta_n]$ is equivalent to $\eta^b[\,] \to \eta_1^t[\,] \ldots \eta_n^t[\,]$.

5 An Earley-like Algorithm Preserving the VPP

Parsers satisfying the *valid prefix property* (VPP) guarantee that, as they read the input string from left to right, the substrings read so far are valid prefixes of the language defined by the grammar. More formally, a parser satisfies the valid prefix property if, for any substring $a_1 \ldots a_k$ read from the input string $a_1 \ldots a_k a_{k+1} \ldots a_n$, it guarantees that there is a string of tokens $b_1 \ldots b_m$, where b_i need not be part of the input string, such that $a_1 \ldots a_k b_1 \ldots b_m$ is a valid string of the language.

In the case of LIG, preserving the valid prefix property requires checking if each predicted production $A[\circ\circ\gamma] \to \bullet\Upsilon$ satisfies $S[\,] \overset{*}{\Rightarrow} w\; A[\alpha\gamma]\; \Upsilon$ where $\gamma \in V_I \cup \{\epsilon\}$. Therefore, to obtain an Earley-like parsing algorithm for LIG preserving this property we need to modify the Pred steps of the parsing system $\mathbb{P}_{\mathrm{E}}$ in order to predict information about the indices stacks. As a consequence, items must be also modified, introducing a new element that allows us to track the contents of the predicted indices stacks. The items are now of the form $[E, h \mid A \to \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B, p, q]$ and they represent one of the following types of derivation:

- $S[\,] \overset{*}{\Rightarrow} a_1 \ldots a_h\, E[\alpha]\, \Upsilon_4 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i\, A[\alpha\gamma]\, \Upsilon_3 \Upsilon_4 \overset{*}{\Rightarrow} a_1 \ldots a_h \ldots a_i \ldots a_p\, B[\alpha]\, a_{q+1} \ldots a_j\, \Upsilon_2 \Upsilon_3 \Upsilon_4$ if and only if $(B, p, q) \neq (-, -, -)$, $A[\alpha\gamma]$ is a dependent descendent of $E[\alpha]$ and $B[\alpha]$ is a dependent descendent of $A[\alpha\gamma]$. This type of derivation corresponds to the completer of the dependent child of a production having the non-terminal A as left-hand side. The indices stack associated to that non-terminal is not empty.

- $S[\,] \overset{*}{\Rightarrow} a_1 \dots a_h\, E[\alpha]\, \Upsilon_4 \overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i\; A[\alpha\gamma]\; \Upsilon_3 \Upsilon_4 \overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i \dots a_j\; \Upsilon_2 \Upsilon_3 \Upsilon_4$ if and only if $(E,h) \neq (-,-)$, $(B,p,q) = (-,-,-)$, $A[\alpha\gamma]$ is a dependent descendant of $E[\alpha]$, Υ_1 does not contain the descendent child of $A[\alpha\gamma]$ and $(p,q) \leq (i,j)$. This type of derivation refers to a prediction of the non-terminal A with a non-empty indices stack.

- $S[\,] \overset{*}{\Rightarrow} a_1 \dots a_i A[\,] \Upsilon_4 \overset{*}{\Rightarrow} a_1 \dots a_i \dots a_j\; \Upsilon_2 \Upsilon_4$ if and only if $(E,h) = (-,-)$, $\gamma = -$ and $(B,p,q) = (-,-,-)$. If Υ_1 includes the dependent child of $A[\,]$ then the indices stack associated to that dependent child is empty. This type of derivation refers to the prediction or completer of a non-terminal A with an empty indices stack.

The new set of items so defined is a refinement of the items in the parsing system $\mathbb{P}_{\mathrm{E}}$: the element γ is used to store the top of the predicted indices stacks (in the parsing system $\mathbb{P}_{\mathrm{E}}$, $\gamma = -$ for items resulting of a prediction) and the pair (E,h) allows us to track the item involved in the prediction.

With respect to the deduction steps, the completer steps must be adapted to the new form of the items in order to manipulate the new components E and h and the predicted steps must be refined taking into account the different types of productions.

Schema 4 *The parsing system $\mathbb{P}_{\mathrm{Earley}_1}$ corresponding to the Earley-like parsing algorithm preserving the valid-prefix property for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \dots a_n$ is defined as follows:*

$$\mathcal{I}_{\mathrm{Earley}_1} = \left\{ \begin{array}{ll} [E,h \mid A \to \Upsilon_1 \bullet \Upsilon_2, \gamma, i, j \mid B,p,q] \mid & A \to \Upsilon_1 \Upsilon_2 \in P,\ B,C \in V_N,\ \gamma \in V_I, \\ & 0 \leq h \leq i \leq j\ ,\ \ (p,q) \leq (i,j) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{Earley}_1} = \frac{}{[-,- \mid S \to \bullet\Upsilon, -, 0, 0 \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Scan}}_{\mathrm{Earley}_1} = \frac{\begin{array}{l}[-,- \mid A[\,] \to \bullet a, -, j, j \mid -,-,-], \\ {[a, j, j+1]}\end{array}}{[-,- \mid A[\,] \to a\bullet, -, j, j+1 \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\,]}_{\mathrm{Earley}_1} = \frac{[E,h \mid A \to \Upsilon_1 \bullet B[\,]\, \Upsilon_2, \gamma, i, j \mid C,p,q]}{[-,- \mid B \to \bullet\Upsilon_3, -, j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\circ\circ\gamma][\circ\circ]}_{\mathrm{Earley}_1} = \frac{\begin{array}{l}[E,h \mid A[\circ\circ\gamma] \to \Upsilon_1 \bullet B[\circ\circ]\, \Upsilon_2, \gamma, i, j \mid -,-,-], \\ {[M,m \mid E \to \bullet\Upsilon_3, \gamma', h, h \mid -,-,-]}\end{array}}{[M,m \mid B \to \bullet\Upsilon_4, \gamma', j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\circ\circ][\circ\circ]}_{\mathrm{Earley}_1} = \frac{[E,h \mid A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ]\, \Upsilon_2, \gamma, i, j \mid -,-,-]}{[E,h \mid B \to \bullet\Upsilon_3, \gamma, j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Pred}[\circ\circ][\circ\circ\gamma]}_{\mathrm{Earley}_1} = \frac{[E,h \mid A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ\gamma]\, \Upsilon_2, \gamma', i, j \mid -,-,-]}{[A,i \mid B \to \bullet\Upsilon_3, \gamma, j, j \mid -,-,-]}$$

$$\mathcal{D}^{\mathrm{Comp}[\,]}_{\mathrm{Earley}_1} = \frac{\begin{array}{l}[E,h \mid A \to \Upsilon_1 \bullet B[\,]\, \Upsilon_2, \gamma, i, j \mid C,p,q], \\ {[-,- \mid B \to \Upsilon_3\bullet, -, j, k \mid -,-,-]}\end{array}}{[E,h \mid A \to \Upsilon_1\, B[\,] \bullet \Upsilon_2, \gamma, i, k \mid C,p,q]}$$

$$\mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ\gamma][\circ\circ]} = \frac{\begin{array}{l}[E,h \mid A[\circ\circ\gamma] \to \Upsilon_1 \bullet B[\circ\circ]\,\Upsilon_2,\gamma,i,j \mid -,-,-],\\ [M,m \mid \boldsymbol{E} \to \bullet\Upsilon_3,\gamma',h,h \mid -,-,-],\\ [M,m \mid \boldsymbol{B} \to \Upsilon_4\bullet,\gamma',j,k \mid C,p,q]\end{array}}{[E,h \mid A[\circ\circ\gamma] \to \Upsilon_1\, B[\circ\circ] \bullet \Upsilon_2,\gamma,i,k \mid B,j,k]}$$

$$\mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ][\circ\circ]} = \frac{\begin{array}{l}[E,h \mid A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ]\,\Upsilon_2,\gamma,i,j \mid -,-,-],\\ [E,h \mid \boldsymbol{B} \to \Upsilon_3\bullet,\gamma,j,k \mid C,p,q]\end{array}}{[E,h \mid A[\circ\circ] \to \Upsilon_1\, B[\circ\circ] \bullet \Upsilon_2,\gamma,i,k \mid C,p,q]}$$

$$\mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ][\circ\circ\gamma]} = \frac{\begin{array}{l}[E,h \mid A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ\gamma]\,\Upsilon_2\,,\gamma',i,j \mid -,-,-],\\ [A,i \mid \boldsymbol{B} \to \Upsilon_3\bullet\,,\gamma,j,k \mid C,p,q],\\ [E,h \mid \boldsymbol{C} \to \Upsilon_4\bullet,\gamma',p,q \mid D,r,s]\end{array}}{[E,h \mid A[\circ\circ] \to \Upsilon_1\, B[\circ\circ\gamma] \bullet \Upsilon_2,\gamma',i,k \mid D,r,s]}$$

$$\begin{aligned}\mathcal{D}_{\text{Earley}_1} = {} & \mathcal{D}_{\text{Earley}_1}^{\text{Init}} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Scan}} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Pred}[\,]} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Pred}[\circ\circ\gamma][\circ\circ]} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Pred}[\circ\circ][\circ\circ]} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Pred}[\circ\circ][\circ\circ\gamma]} \cup \\ & \mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\,]} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ\gamma][\circ\circ]} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ][\circ\circ]} \cup \mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}\end{aligned}$$

$$\mathcal{F}_{\text{Earley}_1} = \{\ [-,- \mid \boldsymbol{S} \to \Upsilon\bullet\,,-,0,n \mid -,-,-]\ \}$$

The space complexity of the algorithm with respect to the length n of the input string is $\mathcal{O}(n^5)$, due to the five positions of the input string stored in each item. The time complexity is $\mathcal{O}(n^7)$ due to deduction steps in the set $\mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}$. To reduce the time complexity we will use a technique similar to that used in [5, 2] to reduce the complexity of the tabular interpretations of automata for tree adjoining languages. In this case, we split each deduction step in $\mathcal{D}_{\text{Earley}_1}^{\text{Comp}[\circ\circ][\circ\circ\gamma]}$ into two different steps such that their final complexity is at most $\mathcal{O}(n^6)$. The resulting parsing schema is defined by the following parsing system.

Schema 5 *The parsing system $\mathbb{P}_{\text{Earley}}$ corresponding to the Earley-like parsing algorithm preserving the valid-prefix property working with a time complexity $\mathcal{O}(n^6)$ for a linear indexed grammar $\mathcal{G}$ and a input string $a_1 \ldots a_n$ is defined as follows:*

$$\mathcal{I}_{\text{Earley}^{(1)}} = \left\{ \begin{array}{ll}[E,h \mid \boldsymbol{A} \to \Upsilon_1 \bullet \Upsilon_2,\gamma,i,j \mid B,p,q] \mid & \boldsymbol{A} \to \Upsilon_1\Upsilon_2 \in P,\ B,C \in V_N,\ \gamma \in V_I,\\ & 0 \le h \le i \le j\ ,\ \ (p,q) \le (i,j)\end{array}\right\}$$

$$\mathcal{I}_{\text{Earley}^{(2)}} = \left\{ \begin{array}{ll}[[\boldsymbol{A} \to \Upsilon\bullet,\gamma,i,j \mid B,p,q]] \mid & \boldsymbol{A} \to \Upsilon \in P,\ B \in V_N,\\ & \gamma \in V_I,\ i \le j\ ,\ \ (p,q) \le (i,j)\end{array}\right\}$$

$$\mathcal{I}_{\text{Earley}} = \mathcal{I}_{\text{Earley}^{(1)}} \cup \mathcal{I}_{\text{Earley}^{(2)}}$$

$$\mathcal{D}_{\text{Earley}}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^0} = \frac{\begin{array}{l}[A,i \mid \boldsymbol{B} \to \Upsilon_3\bullet\,,\gamma,j,k \mid C,p,q],\\ [E,h \mid \boldsymbol{C} \to \Upsilon_4\bullet,\gamma',p,q \mid D,r,s]\end{array}}{[[\boldsymbol{B} \to \Upsilon_3\bullet\,,\gamma,j,k \mid D,r,s]]}$$

$$\mathcal{D}_{\text{Earley}}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^1} = \frac{\begin{array}{l}[[\boldsymbol{B} \to \Upsilon_3\bullet\,,\gamma,j,k \mid D,r,s]],\\ [E,h \mid A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ\gamma]\,\Upsilon_2\,,\gamma',i,j \mid -,-,-],\\ [E,h \mid \boldsymbol{C} \to \Upsilon_4\bullet,\gamma',p,q \mid D,r,s]\end{array}}{[E,h \mid A[\circ\circ] \to \Upsilon_1\, B[\circ\circ\gamma] \bullet \Upsilon_2,\gamma',i,k \mid D,r,s]}$$

$$\mathcal{D}_{\text{Earley}} = \mathcal{D}^{\text{Init}}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Scan}}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\,]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ\gamma][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Pred}[\circ\circ][\circ\circ\gamma]}_{\text{Earley}_1} \cup$$

$$\mathcal{D}^{\text{Comp}[\,]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ\gamma][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ]}_{\text{Earley}_1} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^0}_{\text{Earley}} \cup \mathcal{D}^{\text{Comp}[\circ\circ][\circ\circ\gamma]^1}_{\text{Earley}}$$

$$\mathcal{F}_{\text{Earley}} = \mathcal{F}_{\text{Earley}_1}$$

Deduction steps Comp[$\circ\circ$][$\circ\circ\gamma$]^0 generate an intermediate item of the form $[[B \to \Upsilon_3 \bullet\ , \gamma, j, k \mid D, r, s]]$ that will be taken as antecedent for steps Comp[$\circ\circ$][$\circ\circ\gamma$]^1 and that represents a derivation

$$B[\gamma'\gamma] \overset{*}{\Rightarrow} a_{j+1} \dots a_p\, C[\gamma']\, a_{s+1} \dots a_q \overset{*}{\Rightarrow} a_{j+1} \dots a_p \dots a_r\, D[\,]\, a_{s+1} \dots a_q \dots a_k$$

for some γ', p and q. Deduction steps Comp[$\circ\circ$][$\circ\circ\gamma$]^1 combine this pseudo-item with an item $[E, h \mid A[\circ\circ] \to \Upsilon_1 \bullet B[\circ\circ\gamma]\, \Upsilon_2\ , \gamma', i, j \mid -, -, -]$ that represents a derivation

$$S[\,] \overset{*}{\Rightarrow} a_1 \dots a_h\, E[\alpha]\, \Upsilon_5 \overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i\, A[\alpha\gamma']\, \Upsilon_3 \Upsilon_5 \overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i \dots a_j\, B[\alpha\gamma'\gamma]\, \Upsilon_2 \Upsilon_3 \Upsilon_5$$

and with an item $[E, h \mid C \to \Upsilon_4 \bullet, \gamma', p, q \mid D, r, s]$ representing a derivation

$$S[\,] \overset{*}{\Rightarrow} a_1 \dots a_h\, E[\alpha]\, \Upsilon_5 \overset{*}{\Rightarrow} a_1 \dots a_h \dots a_p\, C[\alpha\gamma']\, \Upsilon_4 \Upsilon_5 \overset{*}{\Rightarrow} a_1 \dots a_h \dots a_p \dots a_r\, D[\alpha]\, a_{s+1} \dots a_q\, \Upsilon_4 \Upsilon_5$$

and a new item of the form $[E, h \mid A[\circ\circ] \to \Upsilon_1\ B[\circ\circ\gamma] \bullet \Upsilon_2, \gamma', i, k \mid D, r, s]$ is generated. This last item represent the existence of a derivation

$$\begin{aligned}
S[\,] &\overset{*}{\Rightarrow} a_1 \dots a_h\, E[\alpha]\, \Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i\, A[\alpha\gamma']\, \Upsilon_3 \Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i \dots a_j\, B[\alpha\gamma'\gamma]\, \Upsilon_2 \Upsilon_3 \Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i \dots a_j \dots a_p\, C[\alpha\gamma']\, a_{q+1} \dots a_k\, \Upsilon_2 \Upsilon_3 \Upsilon_5 \\
&\overset{*}{\Rightarrow} a_1 \dots a_h \dots a_i \dots a_j \dots a_p \dots a_r\, D[\alpha]\, a_{s+1} \dots a_{q+1} \dots a_k\, \Upsilon_2 \Upsilon_3 \Upsilon_5
\end{aligned}$$

6 The Shared Forest

The algorithms described so far are just recognizers. They do not build a representation of the derived trees. However, we can modify them to build these trees as a shared forest satisfying the following properties: it must store in a compact form all parse trees and it must be possible to retrieve every individual parse tree in linear time with respect to the size of the forest.

Billot and Lang [3] define the shared forest for a context-free grammar $\mathcal{G} = (V_T, V_N, P, S)$ and an input string $a_1 \dots a_n$ as a context free grammar in which the non-terminals are of the form $\langle A, i, j\rangle$, where $A \in V_N$ and $i, j \in 0..n$, and productions are of the form

$$\langle A_0, j_0, j_m\rangle \to w_0\ \langle A_1, j_0, j_1\rangle\ w_1\ \langle A_2, j_1, j_2\rangle\ \cdots\ w_{m-1}\ \langle A_m, j_{m-1}, j_m\rangle\ w_m$$

where $A_0 \to w_0 A_1 w_1 A_2 \dots w_{m-1} A_m w_m \in P$ and $w_i \in V_T^*$, meaning that A_0 recognizes the part $a_{j_0+1} \dots a_{j_m}$ of the input string by applying the production $A_0 \to w_0 A_1 w_1 A_2 \dots w_{m-1} A_m w_m$ such that A_i recognizes the part $a_{j_{i-1}+1} \dots a_{j_i}$ of the input string.

We can extend the concept of shared forest for CFG to define the concept of *LIGed forest* [18]. Given the shared forest of the context-free skeleton of a LIG, when a LIG production $A_0[\circ\circ\gamma] \to A_1[\,] \dots A_d[\circ\circ\gamma'] \dots A_m[\,]$ is involved in a derivation, a production

$$\langle A_0, j_0, j_m\rangle[\circ\circ\gamma] \to \langle A_1, j_0, j_1\rangle\ \cdots\ \langle A_d, j_{d-1}, j_d\rangle[\circ\circ\gamma']\ \cdots\ \langle A_m, j_{m-1}, j_m\rangle$$

is added to the LIGed forest meaning that A_0 recognizes the part $a_{j_0+1} \dots a_{j_m}$ of the input string by applying the production $A_0[\circ\circ\gamma] \to A_1 \dots A_d[\circ\circ\gamma'] \dots A_m$ such that A_i recognizes the part $a_{j_{i-1}+1} \dots a_{j_i}$ of the input string and the indices stack is passed from A_0 to A_d replacing the top index γ for γ'. The LIG so generated does not satisfy our definition of shared forest because single parse trees can not be extracted in linear time. Vijay-Shanker and Weir [18] try to solve this problem by defining a non-deterministic finite state automaton that determines if a given LIGed forest symbol $\langle A, i, j\rangle[\alpha]$ derives a string of terminals. A similar finite-state automata is also defined by Nederhof in [11].

As an alternative approach, Boullier [4] defines the shared forest for a LIG $\mathcal{G} = (V_T, V_N, V_I, P, S)$ and an input string w by means of a *linear derivation grammar*, a context-free grammar recognizing the language defined by the sequences of LIG productions of $\mathcal{G}$ that could be used to derive w. Previously to the construction of the linear derivation grammar, we must compute the transitive closure for a set of relations on $V_N \times V_N$.

To avoid the use of additional data structures, such as finite automata or precomputed relations, we have been inspired by the use of context-free grammars to represent the parse forest of tree adjoining grammars [18] in order to capture the context-freeness of production application in the case of LIG. Given a linear indexed grammar $\mathcal{G} = (V_T, V_N, V_I, P, S)$ and an input string $w = a_1 \dots a_n$, the shared forest for $\mathcal{G}$ and w is a context-free grammar $\mathcal{G}^w = (V_T, V_N^w, P^w, S^w)$. Elements in V_N^w have the form $\langle A, \gamma, i, j, B, p, q\rangle$, where $A, B \in V_N$, $\gamma \in V_I$ and $i, j, p, q \in 0 \dots n$. The axiom S^w is the non-terminal $\langle S, -, 0, n, -, -, -,\rangle$. Productions in P^w are of the form:

Case 1a: If $A[\,] \Rightarrow a_j$ then add the production $\langle A, -, j-1, j, -, -, -\rangle \to a_j$

Case 1b: If $A[\,] \Rightarrow \epsilon$ then add the production $\langle A, -, j, j, -, -, -\rangle \to \epsilon$

Case 2a: If $A[\circ\circ\gamma] \to B[\,]\ C[\circ\circ] \in P$, $B[\,] \overset{*}{\Rightarrow} a_{i+1} \dots a_k$ and $C[\alpha\eta] \overset{*}{\Rightarrow} a_{k+1} \dots a_p\, D[\alpha]\, a_{q+1} \dots a_j$ then add the production $\langle A, \gamma, i, j, C, k, j\rangle \to \langle B, -, i, k, -, -, -\rangle\ \langle C, \eta, k, j, D, p, q\rangle$

Case 2b: If $A[\circ\circ\gamma] \to B[\circ\circ]\ C[\,] \in P$, $B[\alpha\eta] \overset{*}{\Rightarrow} a_{i+1} \dots a_p\, D[\alpha]\, a_{q+1} \dots a_k$ and $C[\,] \overset{*}{\Rightarrow} a_{k+1} \dots a_j$ then add the production $\langle A, \gamma, i, j, B, i, k\rangle \to \langle B, \eta, i, k, D, p, q\rangle\ \langle C, -, k, j, -, -, -\rangle$

Case 3a: If $A[\circ\circ] \to B[\,]\, C[\circ\circ] \in P$, $B[\,] \overset{*}{\Rightarrow} a_{i+1} \dots a_k$ and $C[\alpha\eta] \overset{*}{\Rightarrow} a_{k+1} \dots a_p\, D[\alpha]\, a_{q+1} \dots a_j$ then add the production $\langle A, \eta, i, j, D, p, q\rangle \to \langle B, -, i, k, -, -, -\rangle\ \langle C, \eta, k, j, D, p, q\rangle$

Case 3b: If $A[\circ\circ] \to B[\circ\circ]\ C[\,] \in P$, $B[\alpha\eta] \overset{*}{\Rightarrow} a_{i+1} \dots a_p\, D[\alpha]\, a_{q+1} \dots a_k$ and $C[\,] \overset{*}{\Rightarrow} a_{k+1} \dots a_j$ then add the production $\langle A, \eta, i, j, D, p, q\rangle \to \langle B, \eta, i, k, D, p, q\rangle\ \langle C, -, k, j, -, -, -\rangle$

Case 4a: If $A[\circ\circ] \to B[\,]\ C[\circ\circ\gamma] \in P$, $B[\,] \overset{*}{\Rightarrow} a_{i+1} \dots a_k$ and $C[\alpha\eta\gamma] \overset{*}{\Rightarrow} a_{k+1} \dots a_p\, D[\alpha\eta]\, a_{q+1} \dots a_j$ and $D[\alpha\eta] \overset{*}{\Rightarrow} a_{p+1} \dots a_r\, E[\alpha]\, a_{s+1} \dots a_q$ then add the production $\langle A, \eta, i, j, E, r, s\rangle \to \langle B, -, i, k, -, -, -\rangle\ \langle C, \gamma, k, j, D, p, q\rangle\ \langle D, \eta, p, q, E, r, s\rangle$

Case 4b: If $A[\circ\circ] \to B[\circ\circ\gamma]\ C[\,] \in P$, $B[\alpha\eta\gamma] \overset{*}{\Rightarrow} a_{i+1} \dots a_p\, D[\alpha\eta]\, a_{q+1} \dots a_k$ and $C[\,] \overset{*}{\Rightarrow} a_{k+1} \dots a_j$ and $D[\alpha\eta] \overset{*}{\Rightarrow} a_{p+1} \dots a_r\, E[\alpha]\, a_{s+1} \dots a_q$ then add the production $\langle A, \eta, i, j, E, r, s\rangle \to \langle B, \gamma, i, k, D, p, q\rangle\ \langle C, -, k, j, -, -, -\rangle\ \langle D, \eta, p, q, E, r, s\rangle$

Case 5: If $A[\circ\circ\gamma] \to B[\circ\circ] \in P$ and $B[\alpha\eta] \overset{*}{\Rightarrow} a_{i+1} \dots a_p\, D[\alpha]\, a_{q+1} \dots a_j$ then add the production $\langle A, \gamma, i, j, B, i, j\rangle \to \langle B, \eta, i, j, D, p, q\rangle$

Case 6: If $A[\circ\circ] \to B[\circ\circ] \in P$ and $B[\alpha\gamma] \overset{*}{\Rightarrow} a_{i+1} \dots a_p\, D[\alpha]\, a_{q+1} \dots a_j$ then add the production $\langle A, \gamma, i, j, D, p, q\rangle \to \langle B, \gamma, i, j, D, p, q\rangle$

Case 7: If $A[\circ\circ] \rightarrow B[\circ\circ\gamma] \in P$ and $B[\alpha\eta\gamma] \overset{*}{\Rightarrow} a_{i+1} \dots a_p \, D[\alpha\eta] \, a_{q+1} \dots a_j$ and $D[\alpha\eta] \overset{*}{\Rightarrow} a_{p+1} \dots a_r \, E[\alpha] \, a_{s+1} \dots a_q$ then add the production $\langle A,\eta,i,j,E,r,s\rangle \rightarrow \langle B,\gamma,i,j,D,p,q\rangle \; \langle D,\eta,p,q,E,r,s\rangle$

In cases 4a, 4b and 7, derivations starting at $\langle D,\eta,p,q,E,r,s\rangle$ allow us to retrieve the rest of the indices stack corresponding to A. Note that we are assuming a grammar with productions having at most two children. Any production $A_0[\circ\circ\gamma] \rightarrow A_1[\,] \dots A_d[\circ\circ\gamma'] \dots A_m[\,]$ can be translated into

$$\begin{array}{ll}
\nabla_0[\,] \rightarrow \epsilon & \nabla_{d+1}[\circ\circ] \rightarrow \nabla_d[\circ\circ] \; A_{d+1}[\,] \\
\nabla_1[\circ\circ] \rightarrow \nabla_0[\circ\circ] \; A_1[\,] & \vdots \\
\vdots & \nabla_m[\circ\circ] \rightarrow \nabla_{m-1}[\circ\circ] \; A_m[\,] \\
\nabla_{d-1}[\circ\circ] \rightarrow \nabla_{d-2}[\circ\circ] \; A_{d-1}[\,] & A_0[\circ\circ] \rightarrow \nabla_m[\circ\circ] \\
\nabla_d[\circ\circ\gamma] \rightarrow \nabla_{d-1}[\,] \; A_d[\circ\circ\gamma'] &
\end{array}$$

where the ∇_i are fresh symbols that represent partial recognition of the original production. In fact, a ∇_i symbol is equivalent to a dotted production with the dot just before the non-terminal A_{i+1} or with the dot at the end of the right-hand side in the case of ∇_m.

It is interesting to remark that the set of non-terminals is a subset of the set of items for CYK-like and bottom-up Earley-like algorithms, and Earley-like algorithms without the VPP. The case of the Earley-like algorithm preserving the valid prefix property is slightly different, as a non-terminal $\langle A,\gamma,i,j,B,p,q\rangle$ represent the class of items $[E,h \mid A,\gamma,i,j \mid D,p,q]$ for any value of E and h.

Like context-free grammars used as shared forest in the case of TAG [18], the derivations in $\mathcal{G}^w$ encode derivations of the string w by $\mathcal{G}$ but the specific set of terminal strings that is generated by $\mathcal{G}^w$ is not important. We do however have the language generated by $\mathcal{G}^w$ is not empty if and only if w belongs to the language generated by $\mathcal{G}$. We can prune $\mathcal{G}^w$ by retaining only production with useful symbols to guarantee that every non-terminal can derive a terminal string. In this case, derivations of w in the original grammar can be read off by simple reading off of derivations in $\mathcal{G}^w$.

The number of possible productions in $\mathcal{G}^w$ is $\mathcal{O}(n^7)$. The complexity can be reduced to $\mathcal{O}(n^6)$ by transforming productions of the form $A[\circ\circ] \rightarrow B[\,] \; C[\circ\circ\gamma]$ into two productions $A[\circ\circ] \rightarrow B[\,] \; X[\circ\circ]$ and $X[\circ\circ] \rightarrow C[\circ\circ\gamma]$ where X is a fresh non-terminal. A similar transformation must be applied to productions $A[\circ\circ] \rightarrow B[\circ\circ\gamma] \; C[\,]$.

7 Conclusion

We have described a set of algorithms for LIG parsing, creating a continuum which has the CYK-like parsing algorithm by Vijay-Shanker and Weir [16] as its starting point and a new Earley-like algorithm which preserves the valid prefix property as its goal. In the middle, a new bottom-up Earley-like algorithm and a new Earley-like algorithm have been described. The time complexity for all these algorithms with respect to the length of the input string is $\mathcal{O}(n^6)$. Other algorithms could also have been included in the continuum, but for reasons of space we have chosen to show only the algorithms we consider milestones in the development of parsing algorithms for LIG.

Acknowledgements

We would like to thank Pierre Boullier, Patrice Lopez and Mark-Jan Nederhof for fruitful discussions. This work was partially supported by the FEDER of EU (project 1FD97-0047-C04-02) and Xunta de

Galicia (projects PGIDT99XI10502B and XUGA20402B97).

References

[1] Alonso, M. A., D. Cabrero, E. de la Clergerie, and M. Vilares. 1999 Tabular algorithms for TAG parsing. *Proc. of EACL'99*, pages 150–157, Bergen, Norway.

[2] Alonso, M. A., E. de la Clergerie, and D. Cabrero. 1999. Tabulation of automata for tree adjoining languages. *Proc. of MOL-6*, pages 127–141, Orlando, Florida.

[3] Billot, S. and B. Lang. 1989. The structure of shared forest in ambiguous parsing. *Proc. of ACL'89*, pages 143–151, Vancouver, British Columbia, Canada.

[4] Boullier, P. 1996. Another facet of LIG parsing. *Proc. of ACL'96*, Santa Cruz, CA.

[5] De la Clergerie, E. and M. A. Alonso. 1998. A tabular interpretation of a class of 2-Stack Automata. *Proc. of COLING-ACL'98*, volume II, pages 1333–1339, Montreal, Canada.

[6] De la Clergerie, E., M. A. Alonso, and D. Cabrero. 1998. A tabular interpretation of bottom-up automata for TAG. *Proc. of TAG+4*, pages 42–45, Philadelphia.

[7] Gazdar, G. 1987. Applicability of indexed grammars to natural languages. In U. Reyle and C. Rohrer, editors, *Natural Language Parsing and Linguistic Theories*, pages 69–94. D. Reidel Publishing Company.

[8] Joshi, A. K. and Y. Schabes. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages. Vol 3: Beyond Words*, chapter 2, pages 69–123. Springer-Verlag, Berlin/Heidelberg/New York, 1997.

[9] Nederhof, M-J. 1997 Solving the correct-prefix property for TAGs. In T. Becker and H.-V. Krieger, editors, *Proc. of MOL-5*, pages 124–130, Schloss Dagstuhl, Saarbruecken, Germany.

[10] Nederhof, M-J. 1998. Linear indexed automata and tabulation of TAG parsing. *Proc. of TAPD'98*, pages 1–9, Paris, France.

[11] Nederhof, M-J. 1999. Models of tabulation for TAG parsing. *Proc. of MOL-6*, pages 143–158, Orlando, Florida.

[12] Schabes, Y. 1992. Stochastic lexicalized tree-adjoining grammars. *Proc. of COLING'92*, pages 426–432, Nantes, France.

[13] Schabes, Y. and S. M. Shieber. 1994. An alternative conception of tree-adjoining derivation. *Computational Linguistics*, 20(1):91–124.

[14] Shieber, S. M., Y. Schabes, and F. C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1&2):3–36.

[15] Sikkel, K. 1997. *Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms.* Springer-Verlag, Berlin/Heidelberg/New York.

[16] Vijay-Shanker, K. and D. J. Weir. 1991. Polynomial parsing of extensions of context-free grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, chapter 13, pages 191–206. Kluwer Academic Publishers, Norwell, MA.

[17] Vijay-Shanker, K. and D. J. Weir. 1993. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636.

[18] Vijay-Shanker, K. and D. J. Weir. 1993. The use of shared forest in tree adjoining grammar parsing. *Proc. of EACL'93*, pages 384–393.

Customizable Modular Lexicalized Parsing

R. Basili, M.T. Pazienza, F.M. Zanzotto
Dipartimento di Informatica, Sistemi e Produzione,
Universita' di Roma Tor Vergata (ITALY)
{basili,pazienza,zanzotto}@info.uniroma2.it

Abstract

Different NLP applications have different efficiency constraints (i.e. quality of the results and throughput) that reflect on each core linguistic component. Syntactic processors are basic modules in some NLP application. A customization that permits the performance control of these components enables their reuse in different application scenarios. Throughput has been commonly improved using partial syntactic processors. On the other hand, specialized lexicons are generally employed to improve the quality of the syntactic material produced by specific parsing (sub)process (e.g. verb argument detection or PP-attachment disambiguation). Building upon the idea of grammar stratification, in this paper a method to push modularity and lexical sensitivity, in parsing, in view of customizable syntactic analysers is presented. A framework for modular parser design is proposed and its main properties are discussed. Parsers (i.e. different parsing module chains) are then presented and their performances are analyzed in an application-driven scenarios.

1 Introduction

NLP applications require efficient NLP core components both in terms of linguistic quality and throughput. Different NLP applications have different efficiency constraints and this reflects on each component. Several text processing applications include syntactic parsers as core components. Customizing parsing processors enable the reuse of these components in different application scenarios.

Let us consider as an example a real time application like a front-end question-answering for on-line services. Here fast are preferred to accurate parsing processors. Target sentences are rather simple and structures are recurrent. For example, booking train tickets is often expressed by sentences like: *At what time is the next train from Rome to Paris?*. A parsing processor able to produce partial structures like *At what time* and *from Rome to Paris* is sufficient to support a deductive machinery that answers the question. Complex analysis, e.g. clause boundary recognition, is not relevant as very short sentences (with high expectation about the discourse domain) are always used.

On the other hand, an event recognition (ER) task in an Information Extraction (IE) [20, 21] scenario asks for accurate syntactic material over complex sentences. As an example, let us consider the Penn Tree-bank [19] sentence #1692(9):

(*wsj*_1692(9)) *As part of the agreement, Mr. Gaubert contributed real estate valued at $ 25 million to the assets of Independent American.*

The focus here is on the extraction of the event mainly suggested by syntactic relations established by the verb *to contribute*. Clause embedding, i.e. *valued at $ 25 million*, plays here an important role. As a consequence, deeper parsing, relying on a more expressive grammar, is mandatory.

It is also worth noticing that applications may differ in the *type of necessary syntactic relations.* In order to limit the time complexity, underlying grammars should thus be designed to efficiently cover specific phenomena of interest. *All and only* the information necessary to cover the specific target phenomena with the suitable quality should be used.

A key issue is that limited coverage (i.e. low time complexity) and high confidence are conflicting requirements for the kind of grammatical competence available to the parser. Application developers search for technologies for the largest coverage and confidence of specific phenomena. *Shallow parsing* [1, 2, 9, 4], introduced in the perspective of improving time performances, is, alone, inherently weak and often lexical sensitivity has been suggested as successful approach. In order to increase accuracy, syntactic parsing processors usually exploit lexical information; lexicalized grammar formalisms have been widely proposed (e.g. HPSG [22], LTAG [16], LFG [12]) at this scope, although in frameworks (i.e. linguistic theories) targeted to full syntactic analysis [13]. On the contrary, applications require effective methods for specific phenomena. Flexibility is thus the crucial factor for the success of parsing technologies in applications. It is our opinion that the integration of lexicalized approaches in frameworks for shallow parsing [7] is a relevant area of research.

Building upon the idea of grammar stratification [1], we propose a method to push modularity and lexical sensitivity in parsing in view of customizable syntactic analysers.

Supporting modularity within the parsing process requires:

- a formal and homogeneous definition/representation for the partial parsing results able to support information sharing among subcomponents;
- principles for coherent composition of parsing subcomponents able to ease the design of application specific parsers;
- methods for the systematic control of ambiguity within as well as among the components(i.e. throughout a chain of interactions);
- the detection of specific language phenomena where lexical information is relevant to the control of ambiguity, so that specific lexicalized components can be designed to reflect it.

In this paper a framework for modular parsing is presented. The principles for the stratification of the grammatical analysis and their implications on modularity are defined in the next section. In Section 3 the notion of syntactic module is introduced and a classification according to basic grammatical properties of the different modules is given. In the same section an annotation scheme useful for information exchange among modules is defined as a combination of a dependency and constituency based formalism. Implications on grammatical properties and parsing architectures are then discussed. Finally, section 4 discusses the evaluation of some parsing architectures within a typical application scenario.

2 Fitting parsing performance through stratification and modularization

The interest of NLP application developers is in customizing a parser in order to meet application quality and time constraints. Final performances depend on the adopted trade-off between the two.

It is widely accepted that computational lexicons increase the quality of the syntactic information produced by a parser [8]: this improvement is tightly dependent on the specific language level to which lexical information refers. The stratification of a grammar resulting from a modular decomposition of the parser should facilitates the use of lexical information specific to each level.

Let us again consider the (*wsj*_1692(9)) example and suppose that verb *subcategorization* information is available. The verb *to contribute* would be associated to a `direct object` and to a `recipient` (or `beneficiary`) argument as well. This would result in a frame like `contribute-NP-PP(to)`[1]. The other verb in the sentence, *to value*, would be associated to its `object` (i.e. the evaluated entity) and to a prepositional phrase expressing the "degree/amount" (usually ruled by the preposition *at*), i.e. `value-NP-PP(at)`.

A strategy using a combination of clause boundary recognition and a verb argument detection algorithms could decide that: (i) *valued* is linked to *at \$ 25 million*; (ii) *contributed* is linked to *to the assets*. At the level of PP-attachment, most of the ambiguities in the sample sentence disappear since they are resolved by lexical information. Firstly, links derived on lexical basis (i.e. attachment of verb argumental modifiers) have important effects on the remaining ambiguities: other potential attachment sites of argumental PPs like *at \$ 25 million* and *to the assets* are discarded. Secondly, persistent ambiguity is reduced. The $(\textit{of Independent American})_{PP}$ structure is no longer allowed to attach to nouns like *real estate* or *million* as illegal bracket crossing of the clause related to *contribute* would be generated: as a result the only allowed attachments are those with the verb *contribute* itself or with the noun *assets*.

The search space of the parser during the above lexicalized process depends on the number of sentence words. If an early parsing phase, i.e. *chunking* [1], is applied, later parsing steps (e.g. the detection of verb modifiers) deal with a much lower amount of ambiguity. *Chunking* is widely adopted to recognize sentence fragments whose boundaries are independent from the verb grammatical projections. In the example sentence (*wsj*_1692(9)), noun phrases (e.g. *Mr. Gaubert*, *real estate*) and modifiers (e.g. *to the assets*, *at \$ 25 million*) are simple examples of these segments. The detection of verb modifiers is disburdened since it has to deal only with the representative elements of the recognized structures.

The above example is a simple instance of a phenomenon (i.e. verb subcategorization) that plays a relevant role in the control of the ambiguity propagation throughout the search space of the parser. The level (i.e. after chunking) in which this algorithm is applied and the used lexical knowledge are crucial for optimizing the derived advantage:

- the use of chunks provide an optimal representation as the search for verb arguments is limited to chunk heads;
- the adopted lexical knowledge (i.e. subcat frames) in this specific process is a well focused components of a lexical KBs;
- the verb argument detection suggested by the example strongly interact with other parsing activities (e.g. detection of non-argumental and nominal modifiers), with positive side-effects on the reduction of ambiguity.

The above properties are not specific to this kind of modular decomposition (i.e. chunking + verb_phrase_parsing) but can be generalized to a variety of other potential decompositions. The

[1]Note that the `subject` is missing as mandatory in syntax, although it can be omitted.

effects of lexical information within each component increase the accuracy with respect to each target specific (sub)problem. Modularity thus optimizes the lexical effects on the control of ambiguity throughout chains of specific parsing steps.

The adoption of a modular view in parsing supports a more flexible design (via composition of simpler subcomponents in different parsing architectures) and the throughput control is explicit.

First throughput constraints can be met via simplification (i.e. removing not crucial subcomponents) of the overall architecture. If a modular design is adopted, functionalities of modules and functional dependencies are well-defined. Eliciting processing capabilities consists in removing modules from the parsing architecture.

Moreover, modularity again helps in the control of losses in accuracy over the target phenomenon due to the removal of modules. As an example, let us consider a parser aiming to determine NP boundaries in order to detect candidate terms within a Terminology Extraction process. The removal of a verb argument recognition module would increase the parser throughput, by reducing also the resulting precision. In the example ($wsj\_1692(9)$), the lack of verb subcategorization information provides, as a potential NP, the wrong excerpt *\$ 25 million to the assets of Independent American.* It is only by means of a well-defined notion of verb argument detection component that a systematic measure of the trade-off between accuracy and throughput can be controlled and employed as a design principle.

In the next section, a method for designing modular parsing systems is introduced able to support principles of lexicalization and decomposition.

3 A modular approach to parsing

A syntactic processor *SP*, according to the classification given in [3], is a *linguistic processing module.* It is a function $SP(S, K)$ that, exploiting the syntactic knowledge *K*, produces a syntactic representation of the input sentence *S*.

The stratification of the grammar induces modularization of the syntactic processor. The general module component P_i takes the sentence at given state of analysis S_i and augments this information in S_{i+1} exploiting the knowledge K_i. The parser SP is thus a cascade of this modules.

It is crucial to define how the syntactic information produced and processed is represented. The stratification of the parsing activity requires that the representation scheme adopted satisfies some requirements. In fact, on the one hand, stratified parsing techniques require the handling of partially parsed structures (cf. Sec. 2). On the other, lexicalized approaches require that the heads of some types of phrases are accessible during the analysis. In sec. 3.1, classical representations are discussed from the point of view of a modular perspective. Then, we propose, in sec. 3.2, an annotation scheme that satisfies the two requirements, some properties of the annotation scheme are discussed and some restrictions, i.e. *planarity constraints*, proposed. Finally, a classification of the modules is given in Sec. 3.3 according to the kind of information K used and to the typical actions they perform in augmenting the syntactic knowledge gathered for the input sentence.

3.1 Modularity vs. annotation scheme

Modularization and lexicalization impose strict requirements on the annotation scheme used to describe the syntactic information that the processors gather for a target sentence.

In a modularized approach, a stable representation of partially analyzed structures is crucial. In particular, it is required to handle the representation of long-distance dependencies. For instance, considering the example (*wsj*_1692(9)), at a given state of the analysis could be necessary to express that *contributed* is linked to *to the assets.* In a constituency-based framework [11], it is quite hard to express the above relation without specifying the role of the excerpt *real estate valued at $ 25 million.* Furthermore, in the same framework, the relation between contiguous constituents can not be expressed if the constituent captured is not completely formed. In the excerpt of the example sentence (*wsj*_1692(9)) *contributed real estate valued at $ 25 million*, the relation between *contributed* and *real estate* can be expressed only if the constituent *real estate valued at $ 25 million* has been fully recognized. Extensions of constituency-based theories such as TAG [17] and D-Trees [23] allow to express discontinuous links and partial trees. From this point of view, a dependency-based syntactic [24] representation is preferable, since constituency-based approaches in the annotation are not naturally conceived for the representation of distant dependencies without specifying the role of inner structures. On the other hand, a fully dependency-based syntactic approach generally considers the words of a sentence as basic constituents. Thus, each analyzing step has to deal with the same simple constituents: no packing of information is allowed. However, packing is important in a modular approach. A processor using verb subcategorization frames as suggested in section 2, would be enhanced by looking at the candidate complements as single structures. For instance (*wsj*_1692), the analysis of the complements of the verb `contribute-NP-PP(to)` is disburdened if the candidate excerpt of the sentence were factorized in its chunks *[real estate][valued][at $ 25 million][to the assets][of Independent American].* In fact, the argument *PP(to)* can be easily filled with the chunk *[to the assets].*

In a lexicalized approach, it is crucial to determine the *potential governor* [14] of a given structure that is its semantic head [22] and activates lexicalized rules. For instance, given the structure *[has widely contributed]*, the annotation scheme should allow to express that the lexical item governing its behavior is *contribute.*

3.2 Extended dependency graph

To satisfy the requirements imposed by the modularization and the lexicalization, the adopted annotation scheme is a combination of the constituency-based and the dependency-based formalisms. Basically, the syntactic information associated to a given sentence is gathered in a graph, i.e. $g = (n, a)$. The typed nodes (i.e. elements of n) of the graph g are the basic constituents of the sentence, while the typed and oriented arcs (i.e. elements of a) express dependencies between constituents (an head and a modifier). Since the order of constituents is important, the set n is an ordered set. For the purposes of the syntactic parsing, nodes can represent sequences of words, i.e. constituents, that can degenerate in a single word. To satisfy the constraint arisen by the lexicalization, a function h that spot the head of each constituent has been introduced. The representation should allow to express the type of each constituent and each arc. The possible types, elements, respectively, of the sets $NTAG$ and $ATAG$, depend on the underlying grammar model. In the following, we refer to those representation graphs as eXtended Dependency Graph (XDG) that is defined as follows:

Def. 1
An XDG is a tuple $XDG =< n, a, Ntag, Atag, h >$ where n are the nodes, a are the arcs, $Ntag$ is the function that relates n with the set of $NTAG$, $Atag$ is the function that relates a with the set of $ATAG$, and h is the function that elects for each node a representing head.

For sake of simplicity, we introduce a compact version $G = (N, A)$ of the XDG. The compact version is a transcription of the XDG defined as follows:

Def. 2
$G = (N, A)$ related to XDG is such that $N = \{(node, tag, head) | node \in n, tag = Ntag(n), head = h(n)\}$ and $A = \{(arc, tag) | arc \in a, tag = Atag(arc)\}$.

The proposed XDG allows to model the grammatical information, i.e. the detected relation and persisting ambiguity, in an efficient way. In an XDG alternative interpretations coexist. In general, more than one interpretations projected by the same nodes are expressed by the same representation graph that, by itself, do not allow multiple interpretations of the nodes. The ambiguity at this level can be modeled with an inherent proliferation of the interpretation graphs. This limitation is an inheritance of the dependency-based theory. Generally, in these theories, words in an interpretation representation belongs to exactly a single word class (cf. [10]).

The XDG represents a single syntactic interpretation only if it is a *dependency tree* (defined in [10]). In term of constraints on the XDG, the requirement translates in the property that forbids multi-headed nodes [24]:

Prop. 1: *Single headed nodes*
if $\exists(a, b) \in A$ then $\forall a' \in N$ then $\nexists(a', b) \in A$.

For instance, in the example ($wsj\_1692(9)$), an interpretation willing to be a single unambiguous syntactic representation of the sentence can not include both the relations *([valued],[at $ 25 million])* and *([contributed],[at $ 25 million])*.

In order to preserve the compatibility in the proposed representation with the constituency based approach, the property (*Prop. 1*) is not enough. Not enabling *crossing links* may be required. *Crossing links* are defined as follows:

Def. 3: *Crossing links*
Two links, $(w_h, w_k), (w_m, w_n) \in A$ where $min\{h, k\} < min\{m, n\}$, are *crossing* **iff** $min\{m, n\} < max\{h, k\} < max\{m, n\}$.

The *planarity* property [15] can, thus, be introduced:

Prop. 2: *Planarity*
$\forall l_1, l_2 \in A. l_1, l_2$ are not crossing.

The two properties, *Prop. 1* and *Prop. 2*, are called *planarity constraints* and make a XDG that satisfies them a *planar graph*. An XDG satisfying *planarity constraints* is a single (partial) syntactic interpretation.

Consequently, since a viable single interpretation of the sentence must be a *planar graph*, an interpretation in which crossing links coexist is ambiguous. In the example, if both the relations *([valued],[of Independent America])* and *([contributed],[to the assets])* coexist, the interpretation is ambiguous.

3.3 Parsing modules

A component P of the modular syntactic parser is a processor that, using a specific set of rules R, adds syntactic information to the intermediate representation of the sentence. Formally, a processor P is a function $P(R, G)$ where R the knowledge expressed in a specific set of rule, and G the input graph. The result $P(R, G) = G'$ is still an XDG.

The syntactic parser modules are classified according to the actions they perform on the sentence, and to the information they use to perform these actions.

The actions that modules perform on the input XDG can be *conservative* or *not-conservative*. In the case of *conservative modules*, all the choices contained in the input graph are preserved in the output. The property is not true for the *not conservative modules.* A *conservative* module results in a *monotonic* function of the module. A *not-conservative* module is a *not-monotonic* function. A syntactic processor is a cascade of processing modules. Note that the composition of modules preserve, where it exists, the monotonicity.

Furthermore, since the representation of the syntactic information is an *XDG*, the ability of the modules refers to: (i) **constituent** gathering; (ii) **dependency** gathering. Under this distinction, processors are:

- **constituent processors**, P_c, that are purely constituent gatherer;
- **dependency processors**, P_d, that are purely dependency gatherer;
- **hybrid processors**, P_h, that perform both dependency and constituent gathering.

Starting by a model of the process as previously described, i.e. $P(R,G) = G'$, where $G = (N, A)$ and $G' = (N', A')$, and by the distinctions introduced, a description of the typology of processing modules used in the whole parsing processor will be provided. The description is in term of the action they perform on the syntactic graph.

The main characteristic of the processors of the typology P_c is that, in the changing of the constituents (i.e. nodes of the representation) the arcs between constituents are coherently translated, i.e. for each arc in A, there is the correspondent arc in A' if it connects different nodes. For this typology of modules, a *monotonic* processor P_c^M preserves the property of not crossing-brackets between the input and the output, i.e. N' is a partition of N or vice-versa. A *not-monotonic* processor P_c^{NM} does not satisfy this property. In the *monotonic* processors, we distinguish:

[$P_{c.1}^M$] N' is a partition of N and $A' = \{(a,b)|a \neq b, a = (a_1, \dots, a_n), b = (b_1, \dots, b_m), (a_j, b_i) \in A\}$

[$P_{c.2}^M$] N is a partition of N' and $A' = \{(a', b')|a' = h(a), b' = h(b), (a,b) \in A\}$

We now analyze how, according to this taxonomy, a tokenizer T and a chunker [1] can be classified. The aim of a tokenizer is to split a sentence $S = c_1 c_2 \dots c_m$ represented by a stream of characters in its composing words $S' = w_1 w_2 \dots w_n$. It is a $P_{c.1}^M$ module. In fact, the input is a graph whose set of node represents the stream of characters, i.e. $G = (\{(c_1, char, c_1), \dots, (c_m, char, c_m)\}, \emptyset)$, while the output G' models the words, i.e. $G' = (\{(w_1, token, w_1), \dots, (w_n, token, w_n)\}, \emptyset)$. The relation between A and A' satisfies the constraint of the module typology. A chunker [1] falls in the typology $P_{c.1}^M$. As, a *Chunker* is a rewriting device of input sentences, according to the available *chunk prototype*(CP) [6]. The objective of the chunker function is to build the chunk representation $cs = ch_1 \dots ch_m$ corresponding to each input sentence $ws = w_1 \dots w_n$. Each chunk ch_i is the instance of a chunk prototype in CP and is a sequences of words that does not overlaps other chunk of the sentence. Then, in the proposed framework, the chunker transforms $G = (\{(w_1, m_1, w_1), \dots, (w_n, m_n, w_n), A)\}$ in $G' = (\{(ch_1, cht_1, h_1), \dots, (ch_n, cht_n, h_n), A)\}, A')$, where m_i is the pos-tag of the word w_i, cht_i and h_i are respectively the type and the head of the chunk.

The main characteristic of the processor of the type P_d is that in the processing the property $N = N'$ is met. For the *not-monotonic* processors P_d^{NM} of this type no additional property is required. *Monotonic* processor we adopt in the architecture are defined as follows:

[$P_{d.1}^M$] $A \subseteq A'$, G and G' meet planarity constrains (G and G' represent a single interpretation of the sentence)

[$P^M_{d.2}$] $A \subseteq A'$, where for each $a \in A'$ the graph $G'' = (N, A \cup \{a\})$ meets planarity constrains, this means that in general a module of this type introduces ambiguity

According to these definitions, a $P^M_{d.1}$ is a $P^M_{d.2}$. Generally, $P^M_{d.1}$ processors gather unambiguous information and are used to trigger $P^M_{d.2}$ processors. These latter are thought to complete the partial information given by $P^M_{d.1}$ processors. Under this taxonomy, a *link parser* [15] is of the $P^M_{d.1}$. In fact, starting from a representation $G = (ts, \emptyset)$ where ts is the set of ordered tokens representing the target sentence, produces a $G' = (ts, A')$ that is a *planar graph*, i.e. meets planarity constraints.

Another classification may be done considering the knowledge that a processor uses to produce modification in the syntactic graph. Here the lexicalization of the grammatical rules plays a crucial role. In this classification, processors are: (i) **lexicon-driven** processors; (2) **grammar-driven** processors. A mildly lexicalized approach is also possible when grammars are only adopted if lexical information is not available. A lexicalized approach usually depends on the availability of accurate information, and it is usually domain dependent. Examples of the lexicalized modules will be given in the next section.

4 Chaos: a modular lexicalized syntactic parser

The major result of the proposed parsing methodology is the possibility of customization given both by the modular and by the lexicalized approaches. Given a set of syntactic processing modules, this results in a range of possible parsers that differ in term of produced syntactic material and performance. In the following sections, we will introduce *Chaos*, (*Chunk Analysis Oriented System*), a customizable parser based on a pool of four modules: the *Chunker*, the *Verb Shallow Analyzer*, the *Shallow Analyzer*, and the *Projector*. We will discuss its adaptability to different applications through the analysis of performance obtained on the standard Penn Treebank [19].

4.1 Linguistic modules and parsing architectures

The stratification of *Chaos* and its parsing processor modules (Fig. 1), reflect the idea that verbs are crucial in controlling the ambiguity at the level of PP-attachment and are important for applications. Thus, the *Chunker* is especially conceived for packing the ambiguity not relevant at the level of PP-attachment. This rely on syntactic categories and on the relative position between words. It processes a POS tagged sentence $ams = (ws, \emptyset)$ and produces a chunked sentence $chunks = (cs, \emptyset)$ using as rules the *chunks prototypes*(see Fig. 1.(1)). According to the classification given in Sec. 3.3, this processor is a *grammar-based constituent gatherer* ($P^M_{c.1}$).

The verb subcategorization structures that play a disambiguating role are exploited by the verb-driven analysis processor *VSP*(Fig. 1.(3)). It is conceived to efficiently extract dependencies that involve verbs as heads ($V - icds$, i.e. verb inter-chunk dependencies). This processor is a *dependency lexicalized processor*, that can work at different level of lexicalization, of the class $P^M_{d.1}$. *VSP* demands a syntactic graph whose node are chunks, and it works correctly if those chunks are conceived to pack the ambiguity not controlled by verb connections. The module architecture exploits a clause hierarchy approximation (H) via the loop Clause Boundary Recognition (*CBR*) and Verb Shallow Analyzer (*VSA*).

The module of shallow analysis *SP*(Fig. 1.(2)) is designed to express all the syntactic links that are compliant with a particular configuration of the input. It is a *grammar-based dependency module* of

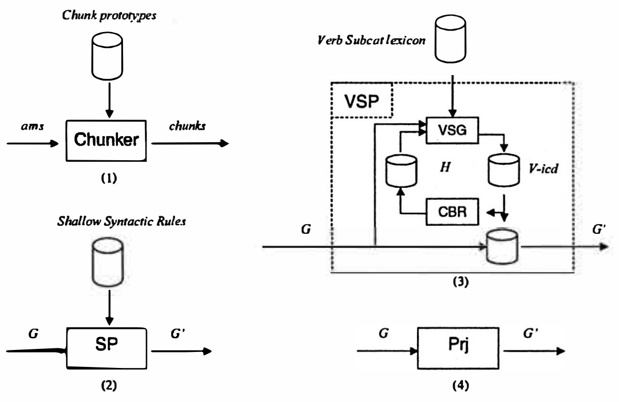


Figure 1: The pool of Chaos processors

the class $P_{d.2}^{M}$.

The module of unambiguous projection *Prj*(Fig. 1.(4)) aims to project a given XDG on the unambiguous subgraph removing colliding arcs. This is a *grammar-based dependency module* of the type P_d^{NM}.

To meet the requirements of an application, different chains of analysis can be arranged. Note that in the present configuration, the *Verb Shallow Analyzer* and *Shallow Analyzer* modules work at the higher level of performance if the specific chunker is used.

A chain *Chunker-Verb Shallow Processor* can be sufficient for an IE application devoted to extract events from sentences if the events prototypes are well described by the verb subcategorization frames. On the other hand, for Lexical Acquisition applications such as verb subcategorization frames acquisition that requires a high coverage of the phenomena [5], a parser composed by the *Chunker* and the *Shallow Analyzer* is sufficient. For an application as Terminology Extraction focussed on Noun Phrase boundary recognition, from the point of view of typology of the phenomena covered a chain composed by the *Chunker* and the *Shallow Processor* is enough, but the performance are not sufficient for the task. Thus, a chain *Chunker-Verb Shallow Processor-Shallow Processor* is required to augment the performance.

4.2 Task oriented parser design

We here analyze how to choose parsing chains for given application scenarios through the investigation of their performances. The examined applications are event recognition in an IE context, and candidate term boundary detection in a Terminology Extraction framework. Performances in term of quality of the syntactic material are evaluated through the metrics of Recall, Precision and F-measure. Given a grammatical relation τ (e.g. $NP - PP$), metrics defined as follows:

$$(a) \quad R^{\tau} = \frac{card((A_o^{\tau} \cap A_s^{\tau}))}{card(A_o^{\tau})} \quad (b) \quad P^{\tau} = \frac{card((A_o^{\tau} \cap A_s^{\tau}))}{card(A_s^{\tau})} \quad (c) \quad F^{\tau}(\alpha) = \frac{1}{(\alpha \frac{1}{P^{\tau}} + (1-\alpha)\frac{1}{R^{\tau}})} \quad (1)$$

A_o^{τ} are the correct syntactic relations of type τ for the sentence, and A_s^{τ} are the syntactic relations of type τ extracted by the system. The *oracle* used is obtained via a translation from the Penn Treebank [19]. The translation of the PTB constituency-based to the dependency-based annotation scheme, compliant with the evaluation requirements, is a crucial problem. Translation algorithms have been settled in previous works [18, 6]. In the present work the adopted translation algorithm left untranslated about 10% of the *oracle trees*(i.e. reference corpus trees). The resulting evaluation test-set consists of nearly 44,000 sentences.

For the event recognition, three parsing chains have been tested: two light and one lexicalized. The first composes the chunker, the shallow analyzer and the disambiguator, i.e. *Chunker-SP-Prj*, the second remove the disambiguator, i.e. *Chunker-SA*, and the third introduces the lexicalized verb shallow analyzer, i.e. *Chunker-VSP-SP-Prj*. The interest here is in extracting relations whose verb is the head (V-Sub, V-Obj, and V-PP).

Parsing chain	**Link Type**	R	P	$F(\alpha = 0.5)$
Chunker-SP	V-Sub	0.75	0.89	0.82
	V-Obj	0.90	0.65	0.75
	V-PP	0.82	0.58	0.68
Chunker-SP-Prj	V-Sub	0.75	0.89	0.82
	V-Obj	0.90	0.66	0.76
	V-PP	0.58	0.94	0.72
Chunker-VSP-SP-Prj	V-Sub	0.76	0.89	0.82
	V-Obj	0.90	0.69	0.78
	V-PP	0.70	0.86	0.77

Table 1: verb arguments

Analyzing the table 1, from the point of view of the coverage of the phenomena, a better architecture appears to be *Chunker-SP*, but it guarantees a low level of precision compared to the other two. In case the interest of event extraction is in populating a database of facts, the most suitable process is the chain that guarantees the higher precision degree: the chain *Chunker-VSP-SP-Prj*. While, if the developer will feed an information retrieval system, the chain *Chunker-SP-Prj* is more appropriate.

In the case of NP recognition that Terminology Extraction (TE) requires, the application is interested in the relation typed NP-PP. Experimental evidence shows that the coverage of the phenomena is assured by a chain *Chunker-SP*, but the quality of the syntactic material is improved through the use of triggers provided by verb subcategorization lexicon in the chain *Chunker-VSP-SP*. The trade off between the cost of the system in term of subcategorization lexicon production and the performance required is another factor to be considered. The table 2 shows experimental results.

Parsing chain	**Link Type**	R	P	$F(\alpha = 0.5)$
Chunker-SP	NP-PP	0.85	0.65	0.73
Chunker-VSP-SP	NP-PP	0.82	0.75	0.78

Table 2: noun phrases-prepositional phrases attachment

In a TE chain where the filtering is based upon statistical methods, the chain *Chunker-SP* is light and assures an higher coverage of the phenomena. While in a TE chain where the filtering is done manually, an high degree of precision disburden the work of the terminologists. The improvement with respect to the precision from *Chunker-SP* to the chain *Chunker-VSP-SP*, even if there is a loss in the recall, may justify the cost in term of time complexity of choosing the *Chunker-VSP-SP* instead of the *Chunker-SP*.

5 Conclusions

A framework for modularization of the parsing process that eases their customization to the applications has been here described. The notion of syntactic module has been introduced and a classification according to basic grammatical properties of the different modules has been provided. Particular attention has been given to the syntactic annotation scheme. A useful syntactic information "holder" for the exchange among modules has been defined as a combination of a dependency and constituency based formalisms. An application of the given framework has been proposed. It has been shown and measured how different NLP applications may select an appropriate parsing chain according to their requirements.

References

[1] Steven Abney. Part-of-speech tagging and partial parsing. In G.Bloothooft K.Church, S.Young, editor, *Corpus-based methods in language and speech.* Kluwer academic publishers, Dordrecht, 1996.

[2] Salah Aït-Mokhtar and Jean-Pierre Chanod. Incremental finite-state parsing. In *Proceedings of ANLP97*, Washington, 1997.

[3] Roberto Basili, Massimo Di Nanni, and Maria Teresa Pazienza. Engineering of ie systems: An object-oriented approach. In Maria Teresa Pazienza, editor, *Information Extraction. Towards Scalable, Adaptabe Systems*, number 1714 in LNAI. Springer-Verlag, Heidelberg, Germany, 1999.

[4] Roberto Basili, Maria Teresa Pazienza, and Paola Velardi. A shallow syntactic analyser to extract word association from corpora. *Literary and linguistic computing*, 7:114–124, 1992.

[5] Roberto Basili, Maria Teresa Pazienza, and Michele Vindigni. Corpus-driven unsupervised learning of verb subcategorization frames. Number 1321 in LNAI, Heidelberg, Germany, 1997. Springer-Verlag.

[6] Roberto Basili, Maria Teresa Pazienza, and Fabio Massimo Zanzotto. Evaluating a robust parser for italian language. In *Proc. of the Workshop on the Evaluation of Parsing Systems, held jointly with 1st International Conference on Language Resources and Evaluation*, Granada, Spain, 1998.

[7] Roberto Basili, Maria Teresa Pazienza, and Fabio Massimo Zanzotto. Lexicalizing a shallow parser. In *Proc. of the TALN99*, Cargese, FR, 1999.

[8] Branimir Boguraev and James Pustejovsky, editors. *Corpus Processing for Lexical Acquisition.* The MIT Press, Cambridge, Massachusetts, US, 1996.

[9] Eric Brill. A simple rule-based tagger. In *Proc. of 3rd Applied Natural Language Processing Conference*, Trento, IT, 1992.

[10] Norbert Broker. A projection architecture for dependency grammar and how it compares to lfg. In *Proc. of LFG98 Conference*, Brisbane, US, 1998.

[11] Naom Chomsky. *Aspect of Syntax Theory.* MIT Press, Cambridge, Massachussetts, 1957.

[12] Mary Darlymple, Ronald M. Kaplan, John T. Maxwell III, and Annie Zeanen, editors. *Formal Issues in Lexical-Functional Grammar.* CSLI Publications, US, 1995.

[13] Christy Doran, Dania Egedi, Beth Ann Hockey, B. Srinivas, and Martin Zaidel. Xtag system - a wide coverage grammar for english. In *Proc. of 15th International Conference on Computational Linguistic, COLING'94*, Kyoto, Japan, 1994.

[14] S. Federici, S. Montemagni, and V. Pirrelli. Shallow parsing and text parsing: a view in underspecification in syntax. In *Proc. of Workshop on robust parsing ESSLLI*, Prague, 1996.

[15] D. Grinberg, J. Lafferty, and D. Sleator. A robust parsing algorithm for link grammar. In *Proc. of 4th International workshop on parsing tecnologies*, Prague, 1996.

[16] A. Joshi and Y. Shabes. Tree-adjoining grammars and lexicalized grammars. In M. Nivat and A. Podelsky, editors, *Definability and Recognizability of Sets of Trees.* Elsevier, 1991.

[17] A.J. Joshi, L. Levy, and M. Takahashi. Tree adjunct grammars. *Journal of Computer and System Science*, 1975.

[18] D. Lin. A dependency-based method for evaluating broad-coverage parsers. In *Proc. of the 14th IJCAI*, pages 1420–1425, Montreal, Canada, 1995.

[19] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19:313–330, 1993.

[20] Maria Teresa Pazienza, editor. *Information Extraction. A Multidisciplinary Approach to an Emerging Information Technology.* Number 1299 in LNAI. Springer-Verlag, Heidelberg, Germany, 1997.

[21] Maria Teresa Pazienza, editor. *Information Extraction. Towards Scalable, Adaptabe Systems.* Number 1714 in LNAI. Springer-Verlag, Heidelberg, Germany, 1999.

[22] C. Pollard and I.A. Sag. *Head-driven Phrase Structured Grammar.* Chicago CSLI, Stanford, 1994.

[23] Owen Rambow, J. Vijay-Shanker, and David Weir. D-tree grammars. In *Proc. of ACL'95*, 1995.

[24] L. Tesniere. *Elements de syntaxe structural.* Klincksiek, Paris, France, 1959.

RANGE CONCATENATION GRAMMARS

Pierre Boullier
INRIA-Rocquencourt
BP 105
78153 Le Chesnay Cedex, France
Pierre.Boullier@inria.fr

Abstract

In this paper we present Range Concatenation Grammars, a syntactic formalism which possesses many attractive features among which we underline here, power and closure properties. For example, Range Concatenation Grammars are more powerful than Linear Context-Free Rewriting Systems though this power is not reached to the detriment of efficiency since its sentences can always be parsed in polynomial time. Range Concatenation Languages are closed both under intersection and complementation and these closure properties may allow to consider novel ways to describe some linguistic processings. We also present a parsing algorithm which is the basis of our current prototype implementation.

1 Introduction

The great number of syntactic formalisms upon which natural language (NL) processing is based may be interpreted in two ways: on one hand this shows that this research field is very active and on the other hand it shows that, at the evidence, there is no consensus for a single formalism and that the one with the *right* properties is still to be discovered. What properties should have such an ideal formalism? Of course, it must allow the description of features that have been identified so far in various NLs, while staying computationally tractable. We know that, due to their lack of expressiveness, context-free grammars (CFGs) cannot play this role (See [Shieber, 1985]). Yet, context sensitive grammars are powerful enough although they are too greedy in computer time. A first answer is given by the notion of mild context-sensitivity. This notion is an attempt to express the formal power needed to define NLs (see [Joshi, 1985] and [Weir, 1988]). However, there exist some phenomena such as large Chinese numbers or word scrambling that are outside the power of mildly context-sensitive (MCS) formalisms. In this paper, we present a convincing alternative: the range concatenation grammars (RCGs). RCG is a syntactic formalism which is a variant of the simple version of literal movement grammar (LMG), described in [Groenink, 1997], and which is also related to the framework of LFP developed in [Rounds, 1988]. In fact it may be considered to lie halfway between their respective *string* and *integer* versions; RCGs retain from the string version of LMGs or LFPs the notion of concatenation, applying it to ranges (couples of integers which denote occurrences of substrings in a source text) rather than strings, and from their integer version the ability to handle only (part of) the source text (this later feature is the key to tractability). The rewriting rules of RCGs, called *clauses*, apply to composite objects named *predicates* which can be seen as nonterminal symbols with arguments. We have shown that the positive version of RCGs, as simple LMGs or integer indexing LFPs, exactly covers the class *PTIME* of languages recognizable in deterministic polynomial time. Since the composition operations of RCGs are not restricted to be linear and non-erasing, its languages (RCLs) are not semi-linear. Therefore, RCGs are *not* MCS and they are more powerful than linear context-free rewriting systems (LCFRS) [Vijay-Shanker, Weir, and Joshi, 1987],[1] while staying computationally tractable: its sentences can be parsed in polynomial time. However, this formalism shares with LCFRS the fact that its derivations are CF (i.e., the choice of the operation performed at each step only depends on the object to be derived from). As in the CF case, its derived trees can be packed into polynomial sized parse forests. Besides its power and efficiency, this formalism possesses many other attractive properties. Let us emphasize in this introduction the fact that RCLs are both closed under intersection and complementation, and, like CFGs, RCGs can act as syntactic backbones upon which can be grafted decorations from other domains (probabilities, logical terms, feature structures).

This paper studies the full class of RCGs and presents a polynomial parse time algorithm.

2 Positive Range Concatenation Grammars

Definition 1 *A positive range concatenation grammar (PRCG) $G = (N, T, V, P, S)$ is a 5-tuple where N is a finite set of predicate names, T and V are finite, disjoint sets of terminal symbols and variable symbols respectively, $S \in N$ is the start predicate name, and P is a finite set of clauses*

$$\psi_0 \to \psi_1 \ldots \psi_m$$

where $m \geq 0$ and each of $\psi_0, \psi_1, \ldots, \psi_m$ is a predicate of the form

$$A(\alpha_1, \ldots, \alpha_i, \ldots, \alpha_p)$$

where $p \geq 1$ is its arity, $A \in N$ and each of $\alpha_i \in (T \cup V)^*$, $1 \leq i \leq p$, is an argument.

Each occurrence of a predicate in the RHS of a clause is a predicate *call*, it is a predicate *definition* if it occurs in its LHS. Clauses which define predicate A are called A-clauses. This definition assigns a fixed arity to each predicate name $A \in N$ whose value is *arity*(A). By definition *arity*(S), the arity of the start predicate name, is one. The *arity* k of a grammar (we have a k-PRCG), is the maximum arity of its predicates.

2.1 Ranges & Bindings

If we consider a derivation in a CFG, headed at the start symbol and leading to some sentence, we know that each nonterminal occurrence is responsible for the generation of the substring laying between two indexes say i and j. For a given input string $w = a_1 \ldots a_n$, such a couple (i, j) is called a *range*. We know that in CF theory, ranges play a central role. For example the (unbounded number of) parse trees, associated with some input string w can be represented by a CFG, called *shared forest* [Lang, 1994]. Its nonterminal symbols have the form (A, i, j) where A is a nonterminal of the initial grammar and (i, j) is a range in w. In an analogous way, ranges are the core of our formalism.

In the sequel terminal symbol in T are denoted by early occurring lower case letters such as $a, b, c, \ldots$, while variables in V are denoted by late occurring upper case letters such as X, Y, Z. If $w \in T^*$, $|w| = n$, its set of ranges is $\mathcal{R}_w = \{\rho \mid \rho = (i, j), 0 \leq i \leq j \leq n\}$. We shall sometimes use vectors to denote elements of cartesian products. For example elements in $\mathcal{R}_w^k$ (i.e., tuple of ranges of length k) may be denoted by $\vec{\rho}$ and clauses by $A_0(\vec{\alpha_0}) \to A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})$, $\vec{\alpha_i} \in ((T \cup V)^*)^{k_i}$, $k_i = arity(A_i)$.

[1]In [Boullier, 1999a], we argue that this extra power can be used in NL processing.

We shall use several equivalent denotations for ranges in $\mathcal{R}_w$. If $w = w_1w_2w_3$ with $w_1 = a_1 \dots a_i$, $w_2 = a_{i+1} \dots a_j$ and $w_3 = a_{j+1} \dots a_n$, the range (i, j) can be denoted either by an explicit dotted term $w_1 \bullet w_2 \bullet w_3$, or by $\langle i..j \rangle_w$, or even by $\langle i..j \rangle$ when w is understood or of no importance. Given a range $\langle i..j \rangle$, the integer i is its *lower bound*, j is its *upper bound* and $j - i$ is its *size*. A range such that $i = j$ is an *empty* range. The three substrings w_1, w_2 and w_3 associated with $\langle i..j \rangle$ are respectively denoted by $w^{\langle 0..i \rangle}$, $w^{\langle i..j \rangle}$ and $w^{\langle j..n \rangle}$. Therefore we have, $w^{\langle j..j \rangle} = \varepsilon$, $w^{\langle j-1..j \rangle} = a_j$ and $w^{\langle 0..n \rangle} = w$. If $\vec{\rho} = \rho_1, \dots, \rho_i, \dots, \rho_p$ is a vector of ranges, by definition $w^{\vec{\rho}}$ denotes the tuple of strings $w^{\rho_1}, \dots, w^{\rho_i}, \dots, w^{\rho_p}$.

In any PRCG, terminals, variables and arguments in a clause are supposed to be bound to ranges by a substitution mechanism. Any couple (X, ρ) is called a *variable binding* denoted by X/ρ, ρ is the *range instantiation* of X, and w^ρ is its *string instantiation*. A set $\sigma = \{X_1/\rho_1, \dots, X_p/\rho_p\}$ of variable bindings is a *variable substitution* iff $X_i/\rho_i \neq X_j/\rho_j \Rightarrow X_i \neq X_j$. A couple (a, ρ) is a *terminal binding* denoted by a/ρ iff $\rho = \langle j-1..j \rangle$ and $a = a_j$.

The *concatenation of ranges* is a partial (associative) binary operation on $\mathcal{R}_w$ defined by $\langle i_1..j_1 \rangle_w \langle i_2..j_2 \rangle_w = \langle i_1..j_2 \rangle_w$ iff $j_1 = i_2$. If we consider a string $w \in T^*$, a string $\alpha = u_1 \dots u_i \dots u_p \in (T \cup V)^*$, a variable substitution σ and a range $\rho \in \mathcal{R}_w$, the couple (α, ρ) is a *string binding* for σ, denoted α/ρ iff

- for each u_i there exists a range ρ_i s.t.
  - if $u_i \in V$, $u_i/\rho_i \in \sigma$,
  - if $u_i \in T$, u_i/ρ_i is such that $u_i = w^{\rho_i}$,
- and $\rho_1 \cdots \rho_i \cdots \rho_p = \rho$

For a given variable substitution σ, a set $\omega = \{\alpha_1/\rho_1, \dots, \alpha_p/\rho_p\}$ is called a *string substitution* (for $\{\alpha_1, \dots, \alpha_p\}$), iff each α_i/ρ_i is a string binding for σ. A clause c is *instantiable* by ω iff there is a string substitution ω for the set of arguments of its predicates. If a clause is instantiable by ω, and if each of its arguments α is replaced by the range ρ s.t. $\alpha/\rho \in \omega$, we get a *positive instantiated clause* whose components are *positive instantiated predicates*. For example, $A(\langle g..h \rangle, \langle i..j \rangle, \langle k..l \rangle) \to B(\langle g{+}1..h \rangle, \langle i{+}1..j{-}1 \rangle, \langle k..l{-}1 \rangle)$ is a positive instantiation of the clause $A(aX, bYc, Zd) \to B(X, Y, Z)$ if the source text $a_1 \dots a_n$ is such that $a_{g+1} = a, a_{i+1} = b, a_j = c$ and $a_l = d$. In this case, the variables X, Y and Z are bound to $\langle g{+}1..h \rangle$, $\langle i{+}1..j{-}1 \rangle$ and $\langle k..l{-}1 \rangle$ respectively.[2]

For some $w \in T^*$, the set of *positive instantiated predicates* IP_w^+ is defined by $IP_w^+ = \{A(\vec{\rho}) \mid A \in N, \vec{\rho} \in \mathcal{R}_w^k, k = arity(A)\}$.

2.2 Derivation, Language, Derived Tree & Shared Forest

For a PRCG $G = (N, T, V, P, S)$ and a source text w, we define, on strings of positive instantiated predicates, a binary relation called *positive derive* and denoted by $\underset{G,w}{\overset{+}{\Rightarrow}}$. If Γ_1 and Γ_2 are strings in $(IP_w^+)^*$, we have

$$\Gamma_1 \; A_0(\vec{\rho_0}) \; \Gamma_2 \quad \underset{G,w}{\overset{+}{\Rightarrow}} \quad \Gamma_1 \; A_1(\vec{\rho_1}) \dots A_m(\vec{\rho_m}) \; \Gamma_2$$

[2]Often, for a variable binding X/ρ, instead of saying that *ρ is the range which is bound to X or denoted by X*, we shall say, the range X, or even for w^ρ, instead of saying *the string whose occurrence is denoted by the range which is bound to X*, we shall say the string X.

iff $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots A_m(\vec{\rho_m})$ is the instantiation of some clause $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \ldots A_m(\vec{\alpha_m})$ in P for some string substitution.

A sequence of strings of positive instantiated predicates $\Gamma_0, \ldots, \Gamma_{i-1}, \Gamma_i, \ldots, \Gamma_l$ s.t. $\forall i, 1 \leq i \leq l$: $\Gamma_{i-1} \underset{G,w}{+\!\!\Rightarrow} \Gamma_i$ is called a *derivation*, or more precisely a Γ_0-*derivation* or even a Γ_0/Γ_l-*derivation.* Each consecutive couple of strings (Γ_{i-1}, Γ_i) is a *derivation step.*

Definition 2 *The* (string) language *of a PRCG* $G = (N, T, V, P, S)$ *is the set*

$$\mathcal{L}(G) \quad = \quad \{w \mid S(\bullet w \bullet) \overset{+}{\underset{G,w}{+\!\!\Rightarrow}} \varepsilon\}$$

An input string $w \in T^*$, $|w| = n$ is a sentence iff the empty string (of positive instantiated predicates) can be derived from $S(\langle 0..n\rangle)$, the positive instantiation of the start predicate on the whole source text.

More generally, we define the string language of a nonterminal A by $\mathcal{L}(A) = \cup_{w \in T^*} \mathcal{L}(A, w)$ where $\mathcal{L}(A, w) = \{w^{\vec{\rho}} \mid \vec{\rho} \in \mathcal{R}^h_w, h = arity(A), A(\vec{\rho}) \overset{+}{\underset{G,w}{+\!\!\Rightarrow}} \varepsilon\}$. However, with this definition, we can note that $\mathcal{L}(S)$ and $\mathcal{L}(G)$ are different. Consider the grammar G s.t.

$$\begin{array}{rcl} S(X) & \rightarrow & A(Xa) \\ A(X) & \rightarrow & \varepsilon \end{array}$$

We can see that we have $\mathcal{L}(G) = \emptyset$ and $\mathcal{L}(S) = T^*$.[3] In fact, we have $\mathcal{L}(G) \subset \mathcal{L}(S)$, and the equality is reached for non-increasing grammars.[4]

As in the CF case, if we consider a derivation as a rewriting process, at each step, the choice of the (instantiated) predicate to be derived does not depend of its neighbors (the derivation process is context-free). All possible derivation strategies can be captured in a single canonical tree structure which abstracts all possible orders and which is called a *derived tree* (or *parse tree*). For any given $A(\vec{\rho})$-derivation, we can associate a single derived tree whose root is labeled $A(\vec{\rho})$. Conversely, if we consider a derived tree, there may be associated derivations which depend upon the way the tree is traversed (for example a top-down left-to-right traversal leads to a leftmost derivation). Note that from a derivation step (Γ, Γ'), it is not always possible to determine which predicate occurrence in Γ has been derived. Moreover, even if this occurrence is known, the clause used cannot be determined in the general case. This is due to the fact that $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots A_m(\vec{\rho_m})$ may be an instantiation of different clauses. But, of course, each of these interpretations is a valid one.

Consider a k-PRCG $G = (N, T, V, P, S)$, a terminal string w and the set $\mathcal{D}_w$ of all complete $S(\bullet w \bullet)/\varepsilon$-derivations. We define a terminal-free CFG $G_w = (N \times \mathcal{R}^k_w, \emptyset, P_w, S(\bullet w \bullet))$ whose set of rules P_w is formed by all the instantiated clauses $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \ldots A_m(\vec{\rho_m})$ used in the derivation steps in $\mathcal{D}_w$. This CFG is called the *shared forest* for w w.r.t. G. Note that, if $\mathcal{D}_w$ is not empty, the language of a shared forest for w is not $\{w\}$, as in the CF case, but is $\{\varepsilon\}$.

Moreover, this shared forest of polynomial size may be viewed as an exact packed representation of all the (unbounded number of) derived (parse) trees in G for w: the set of parse trees for G on the input w and the set of parse trees of its associated CF shared forest G_w (on the input ε), are identical.

[3]For every $w = u_1 u_2 a v, u_1 u_2 v \in T^*$, we have $S(u_1 \bullet u_2 \bullet av) \overset{+}{\underset{G,w}{+\!\!\Rightarrow}} \varepsilon$.

[4]A grammar is *non-increasing* iff for every instantiated clause and for each argument binding α'/ρ', $\rho' = \langle j..k\rangle$ in its RHS there exists in its LHS an argument binding α/ρ, $\rho = \langle i..l\rangle$ such that $i \leq j \leq k \leq l$. Non-increasing grammars represent an important and fairly large subclass of RCGs

The arguments of a given predicate may denote discontinuous or even overlapping ranges. Fundamentally, a predicate name A defines a notion (property, structure, dependency, ...) between its arguments, whose associated ranges can be arbitrarily scattered over the source text. PRCGs are therefore well suited to describe long distance dependencies. Overlapping ranges arise as a consequence of the non-linearity of the formalism. For example, the same variable (denoting the same range) may occur in different arguments in the RHS of some clause, expressing different views (properties) of the same portion of the source text.

Note that the order of predicate calls in the RHS of a clause is of no importance (in fact, RHS of clauses are sets of predicate calls rather than lists).

Example 1 *As an example of a PRCG, the following set of clauses describes the three-copy language $\{www \mid w \in \{a,b\}^*\}$ which is not a CFL and even lies beyond the formal power of tree adjoining grammars (TAGs).*

$$\begin{array}{lcl}
S(XYZ) & \rightarrow & A(X,Y,Z) \\
A(aX,aY,aZ) & \rightarrow & A(X,Y,Z) \\
A(bX,bY,bZ) & \rightarrow & A(X,Y,Z) \\
A(\varepsilon,\varepsilon,\varepsilon) & \rightarrow & \varepsilon
\end{array}$$

*Below, we check that the input string $w = a_1b_2a_3b_4a_5b_6$ is a sentence. Of course, indices are not part of the input letters, they are used to improve readability of ranges: for each couple (*letter*, index), we know where* letter *occurs in the source text. At each derivation step, we have made clear both the clause and the variable substitution used.*

$$\begin{array}{lcll}
S(a_1b_2a_3b_4a_5b_6) & \overset{S(XYZ)\rightarrow A(X,Y,Z)}{\underset{G,w}{\overset{+}{\Rightarrow}}} & A(a_1b_2,a_3b_4,a_5b_6) & \{X/a_1b_2, Y/a_3b_4, Z/a_5b_6\} \\
 & \overset{A(aX,aY,aZ)\rightarrow A(X,Y,Z)}{\underset{G,w}{\overset{+}{\Rightarrow}}} & A(b_2,b_4,b_6) & \{X/b_2, Y/b_4, Z/b_6\} \\
 & \overset{A(bX,bY,bZ)\rightarrow A(X,Y,Z)}{\underset{G,w}{\overset{+}{\Rightarrow}}} & A(\varepsilon,\varepsilon,\varepsilon) & \{X/\langle 2..2\rangle, Y/\langle 4..4\rangle, Z/\langle 6..6\rangle\} \\
 & \overset{A(\varepsilon,\varepsilon,\varepsilon)\rightarrow\varepsilon}{\underset{G,w}{\overset{+}{\Rightarrow}}} & \varepsilon & \emptyset
\end{array}$$

Example 2 *Another way to define the previous language is with the following set of clauses:*

$$\begin{array}{lcl}
S(XYZ) & \rightarrow & L(X)\ eq(X,Y)\ eq(X,Z) \\
L(\varepsilon) & \rightarrow & \varepsilon \\
L(Xa) & \rightarrow & L(X) \\
L(Xb) & \rightarrow & L(X) \\
L(Xc) & \rightarrow & L(X)
\end{array}$$

where the equality predicate *eq is defined by*

$$\begin{array}{lcl}
eq(Xt,Yt) & \rightarrow & eq(X,Y) \\
eq(\varepsilon,\varepsilon) & \rightarrow & \varepsilon
\end{array}$$

in which the first clause is a schema over all terminals $t \in T$.

Example 3 *The power of this formalism is shown by the following grammar that defines the non semi-linear language $\mathcal{L} = \{a^{2^p} \mid p \geq 0\}$*

$$\begin{array}{lcl}
S(XY) & \rightarrow & S(X)\ eq(X,Y) \\
S(a) & \rightarrow & \varepsilon
\end{array}$$

3 Negative Range Concatenation Grammars

Definition 3 *A* negative range concatenation grammar *(NRCG)* $G = (N, T, V, P, S)$ *is a 5-tuple, like a PRCG, except that some predicates occurring in RHS, have the form* $\overline{A(\alpha_1, \ldots, \alpha_p)}$.

A predicate call of the form $\overline{A(\alpha_1, \ldots, \alpha_p)}$ is said to be a *negative predicate call.*

Definition 4 *A* range concatenation grammar *(RCG) is a PRCG or a NRCG.*

The term PRCG (resp. NRCG) will be used to underline the absence (resp. presence) of negative predicate calls.

In a NRCG, the intended meaning of a negative predicate call is to define the complement language (w.r.t. T^*) of its positive counterpart: an instantiated negative predicate succeeds iff its positive counterpart (always) fails. This definition is based on a "negation by failure" rule.

More formally, let $G = (N, T, V, P, S)$ be a RCG, and let w be a string in T^*. The set of *negative instantiated predicate* IP_w^- is defined by $IP_w^- = \{\overline{A(\vec{\rho})} \mid A(\vec{\rho}) \in IP_w^+\}$,[5] and the set of *instantiated predicate* IP_w is defined by $IP_w = IP_w^+ \cup IP_w^-$.

For RCGs, we redefine the *positive derive* relation $\underset{G,w}{+\Rightarrow}$ in the following way. If Γ_1 and Γ_2 are strings in $(IP_w)^*$, we have

$$\Gamma_1 \, A_0(\vec{\rho_0}) \, \Gamma_2 \ \underset{G,w}{+\Rightarrow} \ \Gamma_1 \, \phi_1 \ldots \phi_m \, \Gamma_2$$

iff $A_0(\vec{\rho_0}) \rightarrow \phi_1 \ldots \phi_m$ is the instantiation of some clause $A_0(\vec{\alpha_0}) \rightarrow \psi_1 \ldots \psi_m$ in P for some string substitution ω. If $\vec{\alpha_i}/\vec{\rho_i} \in \omega$,[6] we have, either if $\psi_i = A_i(\vec{\alpha_i})$ then $\phi_i = A_i(\vec{\rho_i})$, or if $\psi_i = \overline{A_i(\vec{\alpha_i})}$ then $\phi_i = \overline{A_i(\vec{\rho_i})}$.

Note that negative instantiated predicates cannot be derived further on by positive derive relations.

We also define on strings of instantiated predicates a *negative derive* relation, denoted by $\underset{G,w}{-\Rightarrow}$. If Γ_1 and Γ_2 are strings in $(IP_w)^*$, we have

$$\Gamma_1 \, \overline{A(\vec{\rho})} \, \Gamma_2 \ \underset{G,w}{-\Rightarrow} \ \Gamma_1 \Gamma_2$$

iff $\overline{A(\vec{\rho})}$ is a negative instantiated predicate such that $(A(\vec{\rho}), \varepsilon) \notin \overset{+}{\underset{G,w}{+\Rightarrow}}$.

Note that this definition of $\underset{G,w}{-\Rightarrow}$ "erases" negative instantiated predicates whose positive counterpart is not related to the empty string (of instantiated predicates) by the transitive closure of $\underset{G,w}{+\Rightarrow}$. As a consequence of this definition, the structure (parse tree) associated with a negative derivation step is void, and, more generally, the structure of the (complement) language associated with a negative predicate call is void. In other words, within the RCG formalism, we cannot define any structure between a negative predicate call and the parts of an input string that it selects.

Let $\underset{G,w}{--\Rightarrow}$ be any subset of $\underset{G,w}{-\Rightarrow}$. We define a *positive/negative derive* relation $\underset{G,w}{\pm\Rightarrow}$ by

$$\underset{G,w}{\pm\Rightarrow} \ = \ \underset{G,w}{+\Rightarrow} \cup \underset{G,w}{--\Rightarrow}$$

We say that $\underset{G,w}{\pm\Rightarrow}$ is *consistent* (otherwise *inconsistent*) iff for each $A(\vec{\rho}) \in IP_w^+$ we have either $A(\vec{\rho}) \overset{+}{\underset{G,w}{\pm\Rightarrow}} \varepsilon$ or $\overline{A(\vec{\rho})} \underset{G,w}{\pm\Rightarrow} \varepsilon$, but not both. Note that the existence of both such derivations would show that the tuple of strings $w^{\vec{\rho}}$ simultaneously belongs to the language of A and to its complement!

[5]The previous definition of IP_w^+ still holds for (N)RCGs.

[6]$\vec{\alpha_i}/\vec{\rho_i}$ is a shorthand notation for $\{\alpha_{i1}/\rho_{i1}, \ldots, \alpha_{ip_i}/\rho_{ip_i}\}$ if $\vec{\alpha_i} = \alpha_{i1}, \ldots, \alpha_{ip_i}$ and $\vec{\rho_i} = \rho_{i1}, \ldots, \rho_{ip_i}$.

We say that a grammar G is *consistent* if for every $w \in T^*$, there exists a consistent relation $\underset{G,w}{\overset{\pm}{\Longrightarrow}}$ (otherwise G is *inconsistent*).

A consistent positive/negative derive relation is simply called *derive* and is denoted by $\underset{G,w}{\Rightarrow}$. If a derive relation exists, we can show that it is unique. However, such a derive relation does not always exist, and thus some grammars are inconsistent.

Consider the RCG G whose set of clauses is the singleton $P = \{S(X) \rightarrow \overline{S(X)}\}$. By definition, for any $w \in T^*$ and for any $\rho \in \mathcal{R}_w$, we have

$$\begin{array}{lcl} \{(S(\rho), \overline{S(\rho)})\} & \subset & \underset{G,w}{\overset{+}{\Rightarrow}} \\ \{(\overline{S(\rho)}, \varepsilon)\} & \subset & \underset{G,w}{\overset{-}{\Rightarrow}} \end{array}$$

There are two possibilities for $\underset{G,w}{\overset{-}{\Longrightarrow}}$: the couple $(\overline{S(\rho)}, \varepsilon)$ either does not belong or does belong to $\underset{G,w}{\overset{-}{\Longrightarrow}}$. In the first case, $\underset{G,w}{\overset{\pm}{\Rightarrow}}$ is inconsistent since we have both $(S(\rho), \varepsilon) \not\in \overset{+}{\underset{G,w}{\overset{\pm}{\Rightarrow}}}$ and $(\overline{S(\rho)}, \varepsilon) \not\in \underset{G,w}{\overset{\pm}{\Rightarrow}}$. In the second case, $\underset{G,w}{\overset{\pm}{\Rightarrow}}$ is also inconsistent since we have

$$\begin{array}{lcl} \{(S(\rho), \overline{S(\rho)}), (\overline{S(\rho)}, \varepsilon)\} & \subset & \underset{G,w}{\overset{\pm}{\Rightarrow}} \\ \{(S(\rho), \overline{S(\rho)}), (\overline{S(\rho)}, \varepsilon), (S(\rho), \varepsilon)\} & \subset & \overset{+}{\underset{G,w}{\overset{\pm}{\Rightarrow}}} \end{array}$$

thus G is inconsistent.

In the sequel, we shall only consider consistent RCGs. Of course, PRCGs are always consistent.

Definition 5 *The (string)* language *of a RCG $G = (N, T, V, P, S)$ is the set*

$$\mathcal{L}(G) \quad = \quad \{w \mid S(\bullet w \bullet) \underset{G,w}{\overset{+}{\Rightarrow}} \varepsilon\}$$

The language of inconsistent RCGs is undefined.

In the sequel, without loss of generality, we will both prohibit clauses whose RHS contains arguments that are in T^*, and assume that no argument has more than one instance of any variable.

4 A Parsing Algorithm for RCGs

In our prototype system, we have implemented a RCG parser which is based upon the algorithm depicted in Table 1. Let $G = (N, T, V, P, S)$ be a k-PRCG for which we also consider P as a sequence of clauses denoted by $\vec{P}$. In this algorithm, the functions *clause* and *prdct* are both memoized: their returned values are kept in auxiliary $1+k$-dimensional matrixes Γ and Π which are indexed by elements in $P \times \mathcal{R}_w^k$ and $N \times \mathcal{R}_w^k$. We assume that their elements are all initialized to **unset**. The internal loop at lines #5, #6 and #7 is executed for each possible instantiation of the current clause, with the only constraint that its LHS arguments must always be bound to the parameter $\vec{\rho_0}$. In function *prdct* at line #4, P_A designates the set of A-clauses. It is not difficult to see that we have implemented a top-down recognizer[7] for any PRCG if the function *prdct* is called, for some input string w, with *prdct*$(S, \bullet w \bullet)$.

To turn this recognizer into a parser, we simply add the statement

(6') **output** $(A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \dots A_j(\vec{\rho_j}) \dots A_m(\vec{\rho_m}))$

[7]Bottom-up algorithms can also be devised.

```
(1) function clause $(i, \vec{\rho_0})$ return boolean
(2)    if $\Gamma[i, \vec{\rho_0}] \neq$ unset then return $\Gamma[i, \vec{\rho_0}]$
(3)    let $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots A_j(\vec{\alpha_j}) \dots A_m(\vec{\alpha_m}) = \vec{P}[i]$
(4)    ret-val := $\Gamma[i, \vec{\rho_0}]$ := false
(5)    foreach $A_0(\vec{\rho_0}) \rightarrow A_1(\vec{\rho_1}) \dots A_j(\vec{\rho_j}) \dots A_m(\vec{\rho_m})$ do
(6)       ret-val := ret-val $\vee$ (prdct $(A_1, \vec{\rho_1}) \wedge \dots$ prdct $(A_j, \vec{\rho_j}) \wedge \dots$ prdct $(A_m, \vec{\rho_m})$)
(7)    end foreach
(8)    return $\Gamma[i, \vec{\rho_0}]$ := ret-val
(9) end function
```

```
(1) function prdct $(A, \vec{\rho})$ return boolean
(2)    if $\Pi[A, \vec{\rho_0}] \neq$ unset then return $\Pi[A, \vec{\rho_0}]$
(3)    ret-val := false
(4)    foreach $i$ such that $\vec{P}[i] \in P_A$ do
(5)       ret-val := ret-val $\vee$ clause $(i, \vec{\rho})$
(6)    end foreach
(7)    return $\Pi[A, \vec{\rho_0}]$ := ret-val
(8) end function
```

Table 1: A Recognition Algorithm for PRCGs.

after line #6 in *clause*, statement which must only be executed if the expression (*prdct* $(A_1, \vec{\rho_1}) \wedge \dots \wedge$ *prdct* $(A_m, \vec{\rho_m})$) succeeds.

In order to handle the full class of RCGs, we simply have to change line #6 in *clause* by something like

(6) *ret-val* := *ret-val* $\vee$ (*prdct* $(A_1, \vec{\rho_1}) \wedge \dots \overline{prdct(A_j, \vec{\rho_j})} \wedge \dots$ *prdct* $(A_m, \vec{\rho_m})$)

if $\vec{P}[i]$ has the form $A_0(\vec{\alpha_0}) \rightarrow A_1(\vec{\alpha_1}) \dots \overline{A_j(\vec{\alpha_j})} \dots A_m(\vec{\alpha_m})$.[8]

We can also note that the assignment of $\Gamma[i, \vec{\rho_0}]$ to **false** at line #4 in *clause* allows this recognizer to handle cyclic grammars.[9]

4.1 Its Parse Time Complexity

For a given k-RCG, and an input string $w \in T^*$, $|w| = n$, the number of instantiations of a given clause $\vec{P}[i]$, is less than or equal to $n^{2k(1+l_i)}$ where l_i is the number of predicate calls in the RHS of $\vec{P}[i]$. Thus, for a given clause, thanks to the memoization mechanism, the number of calls of the form *prdct* $(A_j, \vec{\rho_j})$ in line #6 of *clause*, is less than or equal to $l_i n^{2k(1+l_i)} \leq l_i n^{2k(1+l)}$ where l is the length of the longest clause. Thus, for all possible clauses this number is $\sum_{i=1}^{|P|} l_i n^{2k(1+l)} = |G| n^{2k(1+l)}$ if $|G| = \sum_{i=1}^{|P|} l_i$ is the *size* of the grammar. Thus, if we assume that each use of the function *prdct* takes a constant time, the time complexity of this algorithm is at most $\mathcal{O}(|G| n^{2k(1+l)})$,[10] and its space complexity is $\mathcal{O}(|P| n^{2k})$, the size of the memoization matrixes Γ and Π.

We emphasize the fact that a linear dependency upon the grammar size is extremely important in NL processing where we handle huge grammars and small sentences.

[8]Note that in the corresponding shared forest, *negative nodes* of the form $\overline{A_j(\vec{\rho_j})}$ must be considered either as leaves or, equivalently, we must add in the parser a statement such as **output** $(\overline{A_j(\vec{\rho_j})} \rightarrow \varepsilon)$.

[9]I.e., grammars for which there exist derivations such that $A(\vec{\rho}) \underset{G,w}{\overset{+}{\Rightarrow}} \Gamma_1 A(\vec{\rho}) \Gamma_2$. However, in order to get a parser for cyclic grammars, this simple mechanism had to be improved.

[10]In fact we have $\mathcal{O}(|G| n^{2r})$ where r is the maximum number of arguments in a clause.

In the above evaluation, we have assumed that arguments in a clause are all independent; this is rarely the case. If we consider a predicate argument $\alpha = u_1 \dots u_p \in (V \cup T)^*$ and the string binding α/ρ for some variable substitution σ, each position $0, 1, \dots, p$ in α is mapped onto a *source index* (a position in the source text) $i_0, i_1, \dots, i_p$ s.t. $i_0 \leq i_1 \leq \dots \leq i_p$ and $\rho = \langle i_0 .. i_p \rangle$. These source indexes are not necessarily independent (free). In particular, if for example $u_j \in T$, we have $i_j = i_{j-1} + 1$. Moreover, most of the time, in a clause, variables have multiple occurrences. This means that, in a clause instantiation, the lower source indexes and the upper source indexes associated with all the occurrences of the same variable are always the same, and thus are not free. In fact, the degree d of the polynomial which expresses the maximum parse time complexity associated with a clause is equal to the number of free bounds in that clause. For any RCG G, if d is its maximum number of free bounds, the parse time of an input string of length n takes at worst $\mathcal{O}(|G|n^d)$.

If we consider a bottom-up non-erasing[11] k-RCG G, by definition, there is no free bound in its RHS. Thus, in this case, the number of free bounds is less than or equal to $d = \max_{\vec{P}[i]}(k_i + v_i)$, where k_i and v_i are respectively the arity and the number of (different) variables in the LHS predicate of $\vec{P}[i]$.

In Example 2, we have defined a predicate named *eq* which may be useful in many grammars, thus, in our prototype implementation, we decided to predefine it, together with some others, among which we quote here *len* and *eqlen*:

len(l, X): checks that the size of the range denoted by the variable X is the integer l;

eqlen(X, Y): checks that the sizes of X and Y are equal;

eq(X, Y): checks that the substrings X and Y are equal.

It must be noted that these predefined predicates do not increase the formal power of RCGs insofar as each of them can be defined by a pure RCG. Their introduction is justified by the fact that they are more efficiently implemented than their RCG-defined counterpart and, more significantly, because they convey static information which can be used to decrease the number of free bounds and may thus lead to an improved parse time.

Consider again Example 2. This grammar is bottom-up non-erasing, and the most complex clause is the first one. Its number of free bounds is four (one argument with three variables), and thus its parse time complexity is at worst $\mathcal{O}(n^4)$. In fact, this complexity is at worst quadratic since neither the lower bound nor the upper bound of the argument XYZ is free since their associated source index values are always 0 and n respectively. Note that the lower bound of the definitions of the unary predicate L is not free either, since its value is always the source index zero. The parse time complexity of this grammar further decreases to linear if *eq* is predefined, because in that case, we statically know that the sizes of X, Y and Z must be equal (there is no more free bound within this first clause which is thus executed in constant time). The linear time comes from the upper bound of the LHS argument in the last three clauses. We can also check that the real parse time complexity of Example 3 is logarithmic in the length of the source text!

5 Closure Properties & Modularity

We shall show below that RCLs are closed under union, concatenation, Kleene iteration, intersection and complementation.

[11]A RCG is *bottom-up non-erasing* if, for each clause, all variables that occur in RHS also occur in LHS.

Let $G_1 = (N_1, T_1, V_1, P_1, S_2)$ and $G_2 = (N_2, T_2, V_2, P_2, S_2)$ be two RCGs defining the languages L_1 and L_2 respectively. Without loss of generality, we assume that $N_1 \cap N_2 = \emptyset$ and that S is a unary predicate name not in $N_1 \cup N_2$. Consider two RCGs $G' = (N_1 \cup \{S\}, T_1, V_1 \cup \{X\}, P_1 \cup P', S)$ and $G'' = (N_1 \cup N_2 \cup \{S\}, T_1 \cup T_2, V_1 \cup V_2 \cup \{X\}, P_1 \cup P_2 \cup P'', S)$ defining the languages L' and L'' respectively. By careful definition of the additional sets of clauses P' and P'', we can get $L'' = L_1 \cup L_2$, $L'' = L_1 L_2$ or $L'' = L_1 \cap L_2$ and $L' = L_1^*$ or $L' = \overline{L_1}$.

Union: $P'' = \{S(X) \to S_1(X), S(X) \to S_2(X)\}$

Concatenation: $P'' = \{S(XY) \to S_1(X)\ S_2(Y)\}$

Intersection: $P'' = \{S(X) \to S_1(X)\ S_2(X)\}$

Kleene iteration: $P' = \{S(\varepsilon) \to \varepsilon, S(XY) \to S_1(X)\ S(Y)\}$

Complementation: $P' = \{S(X) \to \overline{S_1(X)}\}$

In [Boullier, 1999d], we have shown that the emptiness problem for RCLs is undecidable and that RCLs are not closed under homomorphism. In fact, we have shown that a polynomial parse time formalism that extends CFGs cannot be closed both under homomorphism and intersection and we advocate that, for a NL description formalism, it is worth being closed under intersection rather than under homomorphism. This is specially true when this closure property is reached without changing the component grammars.

Let G_1 and G_2 be two grammars in some formalism $\mathcal{F}$, their sets of rules are P_1 and P_2 and they define the languages L_1 and L_2 respectively. We say that $\mathcal{F}$ is *modular* w.r.t. some closure operation f if the language $L = f(L_1, L_2)$ can be defined by a grammar G in $\mathcal{F}$ whose set of rules P is such that $P_1 \cup P_2 \subset P$. The idea behind this notion of sub-grammar is to preserve the structures (parse trees for G_1 and G_2) built by the component grammars. In that sense, we can say that CFGs are modular w.r.t. the union operation since CFGs have, on the one hand, the formal property to be closed under union and, on the other hand, this union is described without changing the component grammars G_1 and G_2 (we simply have to add the two rules $S \to S_1$ and $S \to S_2$). Conversely, CFGs are not modular w.r.t. intersection or complementation since we know that CFLs are not closed under intersection or complementation. If we now consider regular languages, we know that they possess the formal property of being closed under intersection and complementation; however we cannot say that they are modular w.r.t. these properties, since the structure is not preserved in any sense. For example, let us take a regular CFG G, defining the language L, we know that it is possible to construct a regular CFG whose language is $\overline{L}$, but its parse trees are not related with the parse trees of G.

Following our definition, we see that RCLs are modular w.r.t. union, concatenation, Kleene iteration, intersection and complementation. Of course it is of a considerable benefit for a formalism to be modular w.r.t. intersection and complementation.

Modularity w.r.t. intersection allows one to directly define a language with the properties $P_1 \wedge P_2$, assuming that we have two grammars G_1 and G_2 describing P_1 and P_2, without changing neither G_1 nor G_2.

Modularity w.r.t. complementation (or difference) allows for example to model the paradigm "general rule with exceptions". Assume that we have a property P defined by a general rule R with some exceptions E to this general rule. Thus, formally we have $P = R - E = R \cap \overline{E}$.

Within the RCG formalism, we simply have to add a clause of the form

$$P(X) \rightarrow R(X)\ \overline{E(X)}$$

assuming that P, R and E are unary predicate names. If, moreover, these exceptions are described by some rules say D, we simply have to add the clause

$$P(X) \rightarrow D(X)$$

6 Conclusion

In [Boullier, 1999d], we have shown that the 1-RCG subclass of RCGs with a single argument, is already a powerful extension of CFGs which can be parsed in cubic time and which contains both the intersection and the complement of CFLs. In [Boullier, 1999b&c], we have shown that unrestricted TAGs and set-local multi-component TAGs can be translated into equivalent PRCGs. Moreover, these transformations do not induce any over-cost. For example we have a linear parse time for regular CFGs, a cubic parse time for CFGs and a $\mathcal{O}(n^6)$ parse time for TAGs.

In this paper we present the full class of RCGs in which we can express several NL phenomena which are outside the formal power of MCS formalisms, while staying computationally tractable. The associated parsers work in time polynomial with the size of the input string and in time linear with the size of the grammar. Moreover, in a given grammar, only complicated (many arguments, many variables) clauses produce higher parse times whereas simpler clauses induce lower times.

For a given input string, the output of a RCG parser, that is an exponential or even unbounded set of derived trees, can be represented into a compact structure, the shared forest, which is a CFG of polynomial size and from which each individual derived tree can be extracted in time linear in its own size.

As CFGs, RCGs may themselves be considered as a syntactic backbone upon which other formalisms such as Herbrand's domain or feature structures can be grafted.

And lastly, we have seen that RCGs are modular This allows to imagine libraries of generic linguistic modules in which any language designer can pick up at will when he wants to specify such and such phenomena.

All these properties seem to advocate that RCGs might well have the right level of formal power needed in NL processing.

References

[Boullier, 1999a] Boullier P. (June 1999). Chinese Numbers, MIX, Scrambling, and Range Concatenation Grammars In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics (EACL'99)*, Bergen, Norway. See also *Research Report No 3614* at `http://www.inria.fr/RRRT/RR-3614.html`, INRIA-Rocquencourt, France, Jan. 1999, 14 pages.

[Boullier, 1999b&c] Boullier P. (July 1999). On TAG Parsing *and* On Multicomponent TAG Parsing. In $6^{ème}$ *conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN'99)*, Cargèse, Corse, France, pages 75–84 and pages 321–326. See also *Research Report No 3668* at `http://www.inria.fr/RRRT/RR-3668.html`, INRIA-Rocquencourt, France, Apr. 1999, 39 pages.

[Boullier, 1999d] Boullier P. (July 1999). A Cubic Time Extension of Context-Free Grammars In *Sixth Meeting on Mathematics of Language (MOL6)*, University of Central Florida, Orlando, Florida, USA. See also *Research Report No 3611* at `http://www.inria.fr/RRRT/RR-3611.html`, INRIA-Rocquencourt, France, Jan. 1999, 28 pages.

[Groenink, 1997] Groenink A. (Nov. 1997). Surface without Structure Word order and tractability in natural language analysis. PhD thesis, Utrecht University, The Nederlands, 250 pages.

[Joshi, 1985] Joshi A. (1985). How much context-sensitivity is necessary for characterizing structural descriptions — Tree Adjoining Grammars. In *Natural Language Processing — Theoretical, Computational and Psychological Perspective*, D. Dowty, L. Karttunen, and A. Zwicky, editors, Cambridge University Press, New-York, NY.

[Lang, 1994] Lang B. (1994). Recognition can be harder than parsing. In *Computational Intelligence*, Vol. 10, No. 4, pages 486–494.

[Rounds, 1988] Rounds W. (1988). LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. In *ACL Computational Linguistics*, Vol. 14, No. 4, pages 1–9.

[Shieber, 1985] Shieber S. (1985). Evidence against the context-freeness of natural language. In *Linguistics and Philosophy*, Vol. 8, pages 333–343.

[Vijay-Shanker, Weir, and Joshi, 1987] Vijay-Shanker K., Weir D. and Joshi A. (1987). Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL'87)*, Stanford University, CA, pages 104–111.

[Weir, 1988] Weir D. (1988). Characterizing Mildly Context-Sensitive Grammar Formalisms. PhD thesis, University of Pennsylvania, Philadelphia, PA.

AUTOMATED EXTRACTION OF TAGS FROM THE PENN TREEBANK

John Chen K. Vijay-Shanker
Department of Computer and Information Sciences
University of Delaware
Newark, DE 19716, USA
{jchen,vijay}@cis.udel.edu

Abstract
The accuracy of statistical parsing models can be improved with the use of lexical information. Statistical parsing using Lexicalized tree adjoining grammar (LTAG), a kind of lexicalized grammar, has remained relatively unexplored. We believe that is largely in part due to the absence of large corpora accurately bracketed in terms of a perspicuous yet broad coverage LTAG. Our work attempts to alleviate this difficulty. We extract different LTAGs from the Penn Treebank. We show that certain strategies yield an improved extracted LTAG in terms of compactness, broad coverage, and supertagging accuracy. Furthermore, we perform a preliminary investigation in smoothing these grammars by means of an external linguistic resource, namely, the tree families of an XTAG grammar, a hand built grammar of English.

1 Introduction

Lexicalized grammars have been shown to be not only linguistically appealing but also desirable for parsing disambiguation. For example, among others, [Charniak, 1996] and [Collins, 1996] have found that lexicalizing a probabilistic model substantially increases parsing accuracy. As introduced in [Schabes et al., 1988], lexicalized tree adjoining grammar (LTAG) is a lexicalized grammar formalism in which lexical items are associated with sets of grammatical structures. [Resnik, 1992] shows that parsing disambiguation can be aided by statistical knowledge of cooccurrence relationships between LTAG structures. [Srinivas, 1997] and [Chen et al., 1999] show that considerable parsing disambiguation is accomplished by assigning LTAG structures to words in the sentence using part of speech tagging techniques (supertagging).

An LTAG grammar G and a means of estimating parameters associated with G are prerequisites for probabilistic LTAG. [Schabes, 1992] shows how this may be done through grammar induction from an unbracketed corpus. The number of parameters that must be estimated, however, is prohibitively large for all but the most simple grammars. In contrast, [XTAG-Group, 1995] has developed XTAG, a complex, relatively broad coverage grammar for English. It is difficult, however, to estimate parameters with XTAG because it has been verified to accurately parse only relatively small corpora, such as the ATIS corpus. [Marcus et al., 1993] describes the Penn Treebank, a corpus of parsed sentences that is large enough to estimate statistical parameters. From the treebank, [Srinivas, 1997] heuristically derives a corpus of sentences where each word is annotated with an XTAG tree, thus allowing statistical estimation of an LTAG. This method entails certain drawbacks: the heuristics make several mistakes, some unavoidable because of discrepancies between how XTAG and the Penn Treebank annotate the same grammatical constructions, or because XTAG does not cover all of the

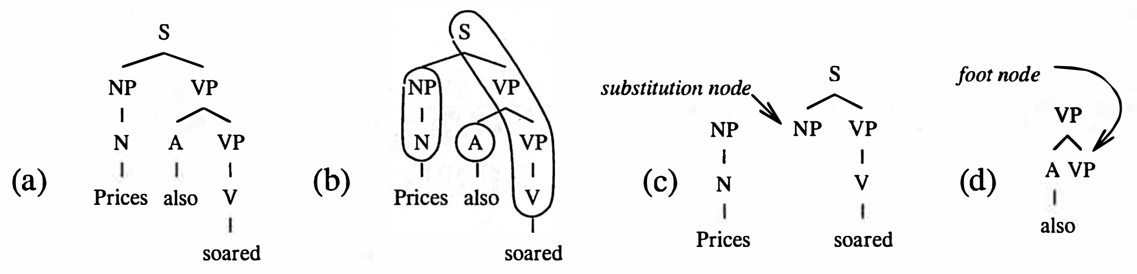


Figure 1: (a) Sentential structure (b) Sentential structure where nonterminals belonging to the same trunk have been circled (c) Localizing argument dependencies in the same elementary tree (d) Each instance of recursion is factored into a separate elementary tree.

grammatical phenomena found in the Penn Treebank. Furthermore, these corpus mistakes usually propagate to the statistical model.

In this work, we explore extraction of an LTAG from the Penn Treebank. This allows us not only to obtain a wide coverage LTAG but also one for which statistical parameters can be reliably estimated. First, we develop various methods for extracting an LTAG from the treebank with the aim of being consistent with current principles for developing LTAG grammars such as XTAG. Second, we evaluate each grammar resulting from these methods in terms of its size, its coverage on unseen data, and its supertagging performance. Third, we introduce a preliminary method to extend an extracted grammar in order to improve coverage. Fourth, we situate our current work among other's approaches for tree extraction. Lastly, we present our conclusions and designs for future work.

2 Tree Extraction Procedure

In this section, we first describe the goals behind a tree extraction procedure and then describe the tree extraction procedure and its variations.

An LTAG G is defined as a set of *elementary trees* T which are partitioned into a set I of *initial trees* and a set A of *auxiliary trees*. The frontier of each elementary tree is composed of a lexical *anchor*; the other nodes on the frontier are *substitution nodes*, and, in the case of an auxiliary tree, one node on the frontier will be a *foot node*. The foot node of a tree β is labeled identically with the root node of β. The *spine* of an auxiliary tree is the path from its root to its foot node. It is to be distinguished from the *trunk* of an elementary tree which is the path from its root node to the lexical anchor.

Although the formalism of LTAG allows wide latitude in how trees in T may be defined, several linguistic principles generally guide their formation. First, dependencies, including long distance dependencies, are typically localized in the same elementary tree by appropriate grouping of syntactically or semantically related elements; i.e. complements of a lexical item are included in the same tree as shown in Figure 1(c). Second, recursion is factored into separate auxiliary trees as shown in Figure 1(d).

The genesis of a tree γ lexicalized by a word $w \in S$, where S is a bracketed sentence in the Penn Treebank, using our tree extraction procedure proceeds as follows. First, a *head percolation table* is used to determine the trunk of γ. Introduced in [Magerman, 1995], a head percolation table assigns to each node in S a *headword* using local structural information. The trunk of γ is defined to be that path through S whose nodes are labeled with the headword w, examples of which are shown in Figure 1(b). Each node η' that is immediately dominated by a node η on the trunk may either be itself on the trunk, a complement of the trunk's headword—in which case it belongs to γ, or an adjunct of the trunk's headword—in which case it belongs to another (auxiliary) tree β which modifies γ.

It is therefore necessary to determine a node's status as a complement or adjunct. [Collins, 1997] introduces a procedure which determines just this from the treebank according to the node's label, its semantic tags, and local structural information. As described in [Marcus et al., 1994], a node's semantic tags provide useful information in determining the node's status, such as grammatical function and semantic role. Our procedure for identifying complements or adjuncts closely follows the method in [Collins, 1997]. The main differences lie in our attempt to treat those nodes as complements which are typically localized in LTAG trees. A critical departure from [Collins, 1997] is in the treatment of landing site of wh-movement. [Collins, 1997]'s procedure treats the NP landing site as the head and its sibling (typically labelled S) as a complement. In our procedure for extracting LTAG trees, we project from a lexical item up a path of heads. Then, by adopting [Collins, 1997]'s treatment, the landing site would be on the path of projection and from our extraction procedure, the wh-movement would not be localized. Hence, we treat the sibling (S node) of the landing site as the head child and the NP landing site as a complement. Figure 4(c) shows an example of a lexicalized tree we extract that localizes long-distance movement.

We have conducted experiments on two procedures for determining a node's status as complement or adjunct. The first procedure that we consider, "CA1," uses the label and semantic tags of node η and η's parent in a two step procedure. In the first step, exactly this information is used as an index into a manually constructed table, which determines complement or adjunct status. "IF current node is PP-DIR AND parent node is VP THEN assign adjunct to current node" is an example of an entry in this table. The table is sparse; should the index not be found in the table then the second step of the procedure is invoked:

1. Nonterminal PRN is an adjunct.

2. Nonterminals with semantic tags NOM, DTV, LGS, PRD, PUT, SBJ are complements.

3. Nonterminals with semantic tags ADV, VOC, LOC, PRP are adjuncts.

4. If none of the other conditions apply, the nonterminal is an adjunct.

Whereas CA1 uses the label and semantic tags of a node η and its parent η', the procedure described in [Xia, 1999], "CA2," uses the label and semantic tags of a node η, its *head sibling* η_h, and *distance* between η and η_h in order to determine the complement or adjunct status of node η. CA2 relies on two manually constructed tables: an argument table and a tagset table. The argument table votes for η being a complement given the label and semantic tags of η and η_h and the distance between them. For example, if η is marked as NP, then the argument table votes for η if η_h is labeled VB and if there is a less than four node separation between η and η_h. The tagset table votes for η being a complement based on the semantic tags of η alone. If both the argument table and the tagset table vote that η should be a complement, it is labeled as such. Otherwise, it is labeled as an adjunct.

A recursive procedure is used to extract trees bottom up given a particular treebank bracketing. Figure 2(a) shows one step in this process. Among all of the children of node η_2, one child η_1 is selected using the head percolation table so that the trunk ϕ associated with η_1 is extended to

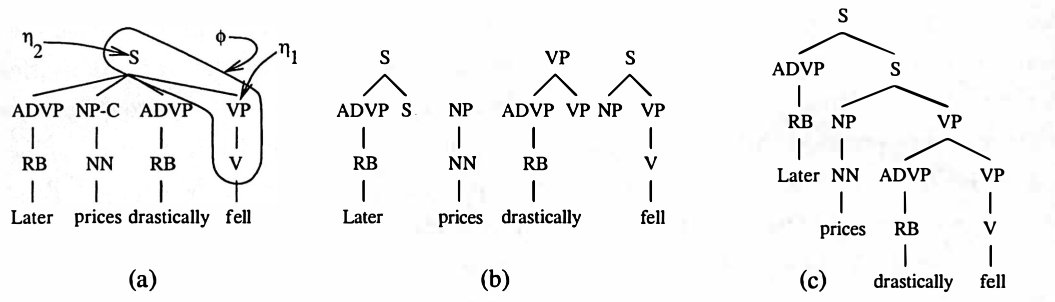


Figure 2: (a) Original Treebank bracketing with head sibling η_1 and its parent η_2 both on trunk of headword "fell," and siblings of η_1 marked "-C" for complement or no annotation for adjunct. (b) Extracted trees (c) Bracketing as defined from extracted trees.

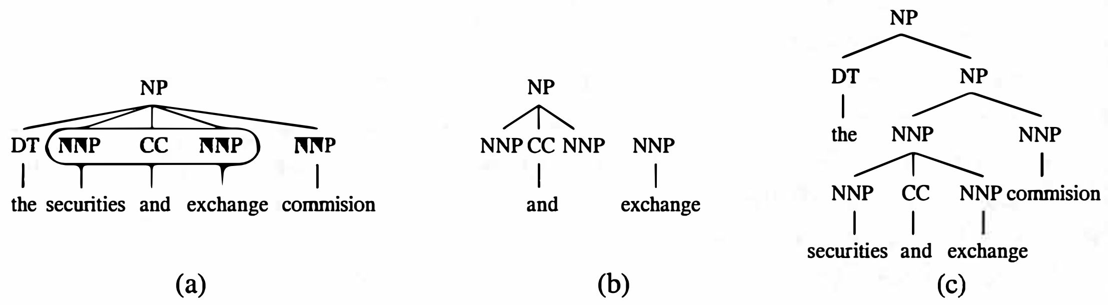


Figure 3: (a) Treebank bracketing of conjunction where instance of conjunction is circled (b) Trees extracted from conjunct (c) Bracketing as defined from extracted trees

η_2. η_1's siblings are subsequently marked as either complement or adjunct. Complement nodes are attached to trunk ϕ and the trees that they dominate become initial trees. Adjuncts are factored into auxiliary trees such that those farthest from η_1 adjoin to η_2 and those closest to η_1 adjoin to η_1, as seen in Figure 2(b). These are *modifier* auxiliary trees, not *predicative* auxiliary trees, which will be discussed later. Although the structure that is defined by the resulting grammar may differ from the original bracketing (see Figure 2(c)), none of the modified bracketings contradicts those in the original Treebank structure, unlike the heuristically derived corpus used by [Srinivas, 1997]. This is important for our goal of ultimately comparing a parser based on this grammar to others' parsers trained on the Penn Treebank. This factoring tends to reduce the number of trees in the resulting grammar. For example, the extracted trees in Figure 2(b) can be used to represent not only "Later prices drastically fell" but other sentences such as "Prices fell" and "Later prices fell."

Our tree extraction procedure also factors the recursion that is found in conjunction. Conjunction in the Treebank is represented as a flat structure such as Figure 3(a). We define an *instance of conjunction* to be a sequence of siblings in the tree $\langle\langle X\rangle_1\ \langle,\rangle_2\ \langle X\rangle_2\ \ldots\ \langle CC\rangle_k\ \langle X\rangle_k\rangle$ where $\langle,\rangle_i$, $\langle X\rangle_i$, and $\langle CC\rangle_i$ are labels of the siblings, and there are k conjuncts. This follows from a basic linguistic notion which states that only like categories can be conjoined. When this configuration occurs in a Treebank bracketing, each pair $\langle\langle,\rangle_i\ \langle X\rangle_i\rangle$ (or $\langle\langle CC\rangle_k\ \langle X\rangle_k\rangle$) is factored into elementary trees as follows. The $\langle,\rangle_i$th (or $\langle CC\rangle_k$th) sibling anchors an auxiliary tree β representing the ith (kth) conjunct. The $\langle X\rangle_i$th (or $\langle X\rangle_k$th) sibling anchors an elementary tree that substitutes into β. See Figure 3(b) for an example where $k = 1$. After factoring of recursion found in the conjunction and in the adjuncts of Figure 3(a), the tree extraction procedure returns a grammar that defines the structure in Figure 3(c).

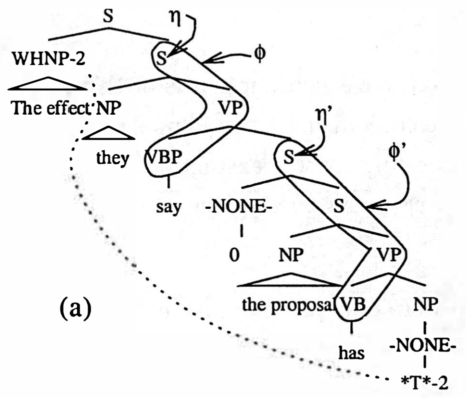
This only considers conjunction of like categories. Although most conjunction is of this nature, it sometimes occurs that constituents with unlike categories are conjoined. In the Penn Treebank, these are annotated with the nonterminal label UCP. Although our current tree extraction procedure does not treat these cases specially as conjunction, a similar strategy may be employed that does so, and in any case they are quite rare.

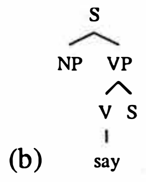
The commas that were found in instances of conjunction were only one example of numerous cases of punctuation that are found in the treebank. In general, these are treated the same as adjuncts. On the other hand, it was found difficult to form a coherent strategy for dealing with quotes. Many times, an open quote would be found in one sentence and the closed quote would be found in an entirely different sentence. Therefore, we chose the simple strategy that quotes would anchor auxiliary trees that would adjoin to a neighboring sibling, namely, that sibling that was closer to the head sibling.

The Penn Treebank has an extensive list of empty elements which are used to define phenomena that are not usually expressed in LTAG. Among these are *U*, expressing a currency value, and *ICH*, indicating constituency relationships between discontinuous constituents. This observation led us to try two different strategies to cope with empty elements. The first strategy "ALL" is to include all empty elements in the grammar. The second strategy "SOME" is to only include empty elements demarcating empty subjects (0), empty PRO and passive NP trace (*), and traces (*T*) of syntactic movement; these are usually found in LTAG grammars of English.

The set of nonterminal and terminal labels in the Penn Treebank is quite extensive. A large set generally means that a greater number of trees are extracted from the Treebank; these trees could miss some generalization and exacerbate the sparse data problem of any statistical model based on them. Also, some nonterminal labels are superfluous because they indicate structural configurations. For example, NX is used to label nodes in the internal structure of multi-word NP conjuncts inside an encompassing NP. If NX were replaced by NP, the tree extraction procedure can still determine that an instance of conjunction exists and take appropriate action. On the other hand, distinctions that are made in a larger set of labels may aid the statistical model. For these reasons, we evaluated two different strategies. One strategy, "FULL," uses the original Penn Treebank label set. Another strategy, "MERGED," uses a reduced set of labels. In the latter approach, the original set is mapped onto a label set similar to that used in the XTAG grammar ([XTAG-Group, 1995]). In our approach, headedness and status as complement or adjunct was first determined according to the full set of labels before the trees were relabeled to the reduced set of labels.

Besides modifier auxiliary trees, there are predicative auxiliary trees which are generated as follows. During the bottom up extraction of trees, suppose trunk ϕ has a node η that shares the same label as another node η', where η' is a complement, not on ϕ, but is immediately dominated by a node on ϕ. In this case, a predicative auxiliary tree is extracted where η is its root, η' is its foot and with ϕ serving as its trunk. Subsequently, the path ϕ' dominated by η' becomes a candidate for being extended further. See Figure 4(a). This mechanism works in concert with other parts of our tree extraction procedure (notably complement and adjunct identification, merging of nonterminal labels (from SBAR to S), and policy of handling empty elements) in order to produce trees that localize long distance movement as shown in Figure 4(c).





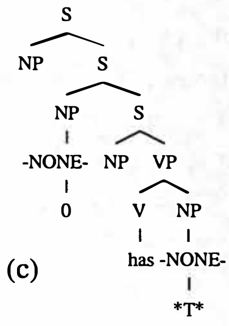


Figure 4: (a) Predicative auxiliary tree associated with ϕ is factored out because nodes η and η' have the same label, leaving trunk ϕ' to be extended to capture long distance movement. (b) The extracted predictive auxiliary tree (c) The tree showing long distance movement

Comp Adjunct	Empty Elements	Label Set	Grammar Size Frames	Grammar Size Lexicalized Trees
CA1	ALL	FULL	8996	118333
CA1	ALL	MERGED	5165	111220
CA1	SOME	FULL	8623	117527
CA1	SOME	MERGED	4911	110428
CA2	ALL	FULL	5354	116326
CA2	ALL	MERGED	2632	108370
CA2	SOME	FULL	4936	115335
CA2	SOME	MERGED	2366	107387

Table 1: Size of various extracted grammars in number of tree frames and number of lexicalized trees

3 Evaluation

Each variation of tree extraction procedure was used to extract a grammar from Sections 02-21 of the Penn Treebank. These grammars were evaluated according to size, well formedness of trees, their coverage on Section 22, and their performance in supertagging Section 22. We subsequently evaluated truncated forms of these grammars which we term *cutoff* grammars.

The grammars' sizes in terms of number of lexicalized trees and *tree frames* are shown in Table 1. Removing the anchor from a lexicalized tree yields a tree frame. In terms of different tree extraction strategies, MERGED yields more compact grammars than FULL, SOME more than ALL, and CA2 more than CA1. Perhaps the last dichotomy requires more of an explanation. Basically, CA2 factors more nodes into auxiliary trees, with the result being that there are fewer trees because each one is structurally simpler.

We may also qualitatively judge grammars according to how well they satisfy our goal of extracting well formed trees in the sense of selecting the appropriate domain of locality and factoring recursion when necessary. There is not much difference between SOME and ALL because the empty elements that SOME ignores are the ones that are not usually captured by any LTAG grammar. Likewise, there is little difference between MERGED and FULL because most of MERGE's label simplification does not occur until after completion of tree extraction. The main difference lies between CA1 and

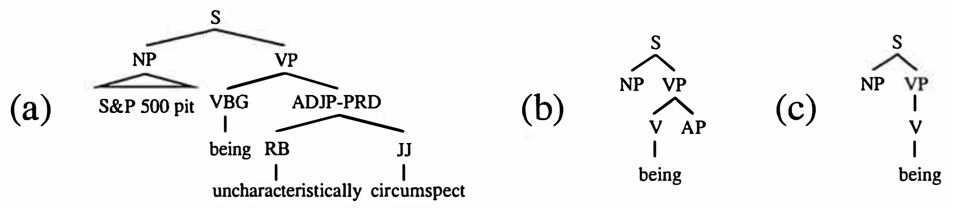


Figure 5: (a) Bracketed sentence S (b) Lexicalized tree extracted from S using strategy CA1 (c) Lexicalized tree extracted from S using strategy CA2

Comp Adj	Empty Elemt	Label Set	% found frames	% found lex trees	% miss in dict	% miss not in dict	Supertag Accuracy	Cutoff Accuracy
CA1	ALL	FULL	99.56	91.57	5.67	2.76	77.79	77.85
CA1	ALL	MERGED	99.82	92.18	5.06	2.76	78.70	78.57
CA1	SOME	FULL	99.60	91.66	5.58	2.76	78.00	78.07
CA1	SOME	MERGED	99.83	92.27	4.98	2.76	78.90	78.78
CA2	ALL	FULL	99.80	92.05	5.19	2.76	77.85	77.79
CA2	ALL	MERGED	99.94	92.71	4.53	2.76	78.25	78.25
CA2	SOME	FULL	99.83	92.14	5.10	2.76	78.07	78.08
CA2	SOME	MERGED	99.96	92.80	4.44	2.76	78.55	78.50

Table 2: Coverage of various extracted grammars and their corresponding supertagging performance. Coverage is in terms of percentage of tree frames and lexicalized trees in the test corpus. Missed coverage is divided into *in dict*—word seen in training corpus and *not in dict*—word not seen in training corpus. Supertagging performance is based on either the full grammar or cutoff grammar, cutoff value $k = 3$.

CA2, strategies for labeling complements and adjuncts.

Nodes detected as complements of a particular lexical item belong in the same elementary tree, thus satisfying the criterion of localizing dependencies. We believe that CA1 labels nodes closer to what is traditionally found in LTAG grammars such as XTAG than does CA2, in that use of CA2 will generate less appropriate subcategorization frames because it tends to factor what might be considered as complements into separate auxiliary trees. It is difficult, however, to quantify the degree to which strategies CA1 and CA2 are successful in distinguishing complements from adjuncts because there are no precise definitions of these terms. Here we resign ourselves to a qualitative comparison of an example of a lexicalized tree extracted from the same sentence by a CA1 derived grammar G_1 (CA1-SOME-MERGED) and a CA2 derived grammar G_2 (CA2-SOME-MERGED). First, a tree frame F is selected from G_1 that is absent from G_2. A bracketed sentence S out of which the CA1 approach extracts F is then culled from the training corpus at random. Figure 5(a) shows S, (b) shows the tree corresponding to the main verb extracted to G_1, (c) shows the tree corresponding to the main verb extracted to G_2. It is typical of the examples of divergence between CA1 and CA2 derived grammars: the CA1 approach leads to a verb subcategorization that is more complicated, yet more appropriate.

The various extracted grammars may also be evaluated according to breadth of coverage. In order to evaluate coverage of a particular grammar G, the strategy used to produce G was used to produce trees from held out data. We subsequently determined the degree of coverage of that strategy by the overlap in terms of tree frames and lexicalized trees as shown in Table 2. For lexicalized trees t extracted from held-out data such that $t \notin G$, we also show the percentage of time the *lexical anchors* of such trees t were or were not found in the training corpus (column *in dict* and *not in dict* respectively). For

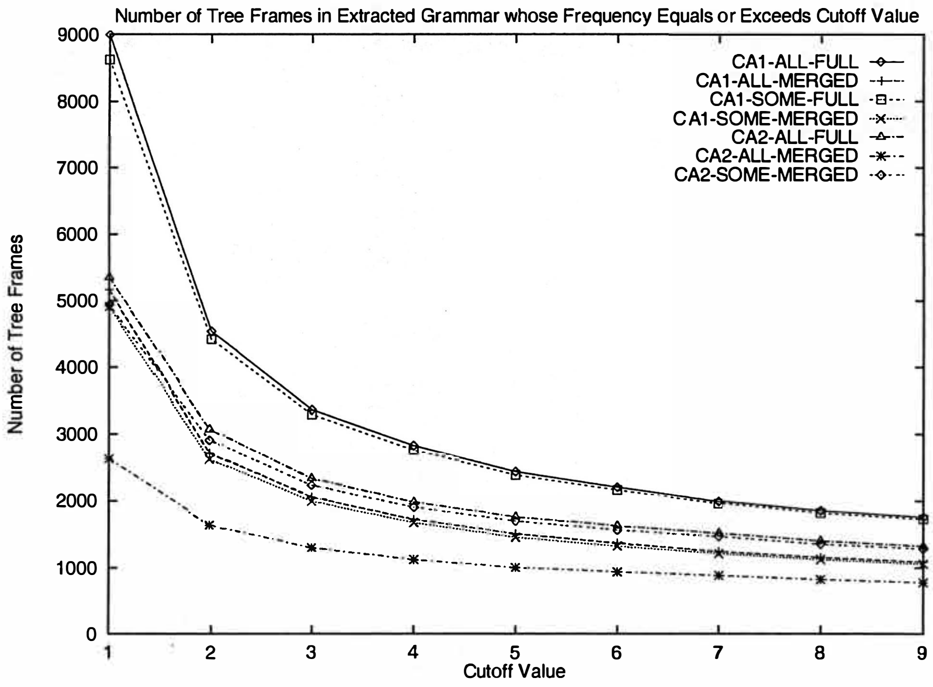


Figure 6: Number of tree frames occurring k times or more in the training corpus

example, the first row of Table 2 reports that use of strategy CA1-ALL-FULL resulted in a grammar such that 99.56% of instances of tree frames in held out data were available in the grammar, and 91.57% of instances of lexicalized trees in held out data were found in the grammar. Of the remaining 8.43%, 2.76% (*not in dict*) of the lexicalized trees in the held out data involved words not seen in the training corpus. The remaining 5.67% therefore are anchored by words in the training corpus but the corresponding associations of tree frames and lexical items were not made in the training corpus. The table shows that strategies that reduce the number of extracted trees (SOME, CA2, MERGED) tend to also increase coverage.

We also measure the accuracies of supertagging models which are based on the various grammars that we are evaluating. Results are shown in Table 2. Curiously, the grammars whose associated models achieved the highest accuracies did not also have the highest coverage. For example, CA1-SOME-MERGED beat CA2-SOME-MERGED in terms of accuracy of supertagging model although the latter achieved higher coverage. This could possibly be caused by the fact that a high coverage grammar might have been obtained because it doesn't distinguish between contexts on which a statistical model can make distinctions. Alternatively, the cause may lie in the fact that a particular grammar makes better (linguistic) generalizations on which a statistical model can base more accurate predictions.

A large grammar may lead to a statistical model that is prohibitively expensive to run in terms of space and time resources. Furthermore, it is difficult to obtain accurate estimates of statistical parameters of trees with low counts. And, in any case, trees that only infrequently appear in the training corpus are also unlikely to appear in the test corpus. For these reasons, we considered the effect on a grammar if we removed those tree frames that occurred less than k times, for some cutoff value k. We call these *cutoff grammars*. As shown in Figure 6, even low values for k yield substantial

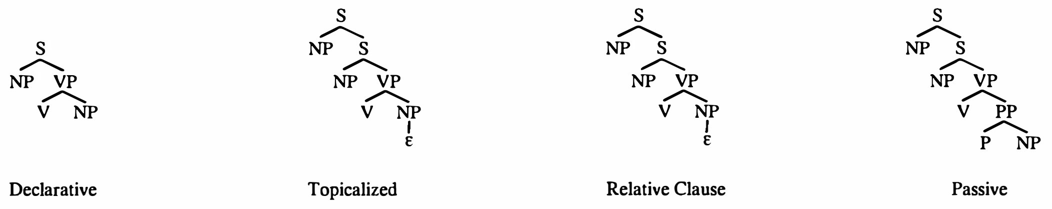


Figure 7: Some trees that are found in an XTAG tree family corresponding to transitive verbs

decreases in grammar size.

Even though a cutoff grammar may be small in size, perhaps a statistical model based on such a grammar would degrade unacceptably in its accuracy. In order to see if this could indeed be the case, we trained and tested the supertagging model on various cutoff grammars. In the training data for these supertagging models, if a particular full grammar suggested a certain tree t_i for word w_i, but the cutoff grammar did not include t_i then word w_i was tagged as *miss*. The cutoff value of $k = 3$ was chosen in order to reduce the size of all of the original grammars at least in half. By the results shown in Table 2, it can be seen that use of a cutoff grammar instead of a full extracted grammar makes essentially no difference in the accuracy of the resulting supertagging model.

4 Extended Extracted Grammars

The grammars that have been produced with the tree extraction procedure suffer from sparse data problems as shown in Table 2 by the less than perfect coverage that these grammars achieve on the test corpus. This is perhaps one reason for the relatively low accuracies that supertagging models based on these grammars achieve compared to, for example, [Srinivas, 1997] and [Chen et al., 1999]. Many approaches may be investigated in order to improve the coverage. For example, although XTAG may be inadequate to entirely cover the Penn Treebank, it may be sufficient to ameliorate sparse data. Here we discuss how linguistic information as encoded in XTAG tree families may be used for this purpose and deliver some preliminary results.

[XTAG-Group, 1995] explains that the tree frames anchored by verbs in the XTAG grammar are divided into tree families. Each tree family corresponds to a particular subcategorization frame. The trees in a given tree family correspond to various syntactic transformations as shown in Figure 7. Hence, if a word w_i is seen in the training corpus with a particular tree frame t_i, then it is likely for word w_i to appear with other tree frames $t \in T$ where T is the tree family to which t_i belongs.

This observation forms the basis of our experiment. The extracted grammar G_0 derived from CA1-SOME-MERGED was selected for this experiment. Call the extended grammar G_1. Initially, all of the trees $t \in G_0$ are inserted into G_1. Subsequently, for each lexicalized tree $t \in G_0$, lexicalized trees t' are added to G_1 such that t and t' share the same lexical anchor and the tree frames of t and t' belong to the same tree family. Out of approximately 60 XTAG tree families, those tree families that were considered in this experiment were those corresponding to relatively common subcategorization frames including intransitive, NP, PP, S, NP-NP, NP-PP and NP-S.

We achieved the following results. Recall that in Table 2 we divided the lapses in coverage of a particular extracted grammar into two categories: those cases where a word in the test corpus was not seen in the training corpus (*not in dict*), and those cases where the word in the test corpus was seen in training, but not with the appropriate tree frame (*in dict*). Because our procedure deals only with reducing the latter kind of error, we report results from the latter's frame of reference. Using grammar G_0, the *in dict* lapse in coverage occurs 4.98% of the time whereas using grammar G_1, such lapses occur 4.61% of the time, an improvement of about 7.4%. This improvement must be balanced against the increase in grammar size. Grammar G_0 has 4900 tree frames and 114850 lexicalized trees. In comparison, grammar G_1 has 4999 tree frames and 372262 lexicalized trees.

The results are somewhat encouraging and we believe that there are several avenues for improvement. The number of lexicalized trees in the extended extracted grammar can be reduced if we account for morphological information. For example, a verb "drove" cannot occur in passive forms. Instead of capitalizing on this distinction, our current procedure simply associates all of the tree frames in the transitive tree family with "drove." In related work, we are currently conducting an experiment to quantitatively extract a measure of similarity between pairs of supertags (tree frames) by taking into account the distribution of the supertags with words that anchor them. When a particular supertag-word combination does not appear in the training corpus, instead of assigning it a zero probability, we assign a probability that is obtained by considering similar supertags and their probability of being assigned with this word. This method seems to give promising results, circumventing the need for manually designed heuristics such as those found in the supertagging work of [Srinivas, 1997] and [Chen et al., 1999]. We plan to apply this strategy to our extracted grammar and verify if similar improvements in supertagging accuracy can be obtained.

5 Related Work

[Neumann, 1998] presents a method for extracting an LTAG from a treebank. Like our work, [Neumann, 1998] determines the trunks of elementary trees by finding paths in the tree with the same headword, headwords being determined by a head percolation table. Unlike our work, [Neumann, 1998] factors neither adjuncts nor instances of conjunction into auxiliary trees. As a result, [Neumann, 1998]'s method generates many more trees than we do. Using only Sections 02 through 04 of the Penn Treebank, [Neumann, 1998] produces about 12000 tree frames. Our approaches produces about 2000 to 9000 tree frames using Sections 02-21 of the Penn Treebank.

[Xia, 1999] also presents work in extracting an LTAG for a treebank, work that was done in parallel with our own work. Like our work, [Xia, 1999] determines the trunk of elementary trees by finding paths in the tree with the same headword. Furthermore, [Xia, 1999] factors adjuncts (according to CA2 only) into separate auxiliary trees. Also, following our suggestion [Xia, 1999] factors instances of conjunction into separate auxiliary trees. [Xia, 1999]'s approach yields either 3000 or 6000 tree frames using Sections 02-21 of the Penn Treebank, depending on the preterminal tagset used. Our work explores a wider variety of parameters in the extraction of trees, yielding grammars that have between about 2000 to 9000 tree frames on the same training data. Unlike our work in the extraction of an LTAG, [Xia, 1999] extracts a multi-component LTAG through coindexation of traces in the Penn Treebank. Another difference is that [Xia, 1999] is more concerned with extraction of grammar for the purpose of grammar development (for which [Xia, 1999] for example makes the distinction between extracted tree frames that are grammatically correct and those that are incorrect), whereas our current work in extraction of grammar betrays our ultimate interest in developing statistical models for parsing (for which we perform an investigation of coverage, supertagging accuracy, effect of cutoff frequency, as well as explore the issue of extending extracted grammars using XTAG tree families for eventual use in statistical smoothing).

This work raises the question as to how parsing with LTAG may compare to parsing where the model is based on a lexicalized context free formalism. Both recent work in parsing with lexicalized models and LTAG appear to manipulate basically the same kinds of information. Indeed, only a few trees in our extracted grammars from the Penn Treebank have the form to cause the generative capacity of the grammars to exceed those of lexicalized context free grammars. The work presented here makes it possible to see how a statistical parsing model based on an LTAG compares with models based on lexicalized context free grammars. Furthermore, supertagging as a preprocessing step may be used to improve the efficiency of a parsing using a statistical model based on an LTAG. We plan to explore these issues in future research.

6 Conclusions

Our work presents some new directions in both the extraction of an LTAG from the Penn Treebank as well as its application to statistical models. In the extraction of an LTAG from the Penn Treebank, we have extended [Neumann, 1998]'s procedure to produce less unwieldy grammars by factoring recursion that is found in adjuncts as well as in instances of conjunction. We have explored the effects that different definitions of complement and adjunct, whether or not to ignore empty elements, and extent of label sets have on the quality and size of the extracted grammar, as well as ability to cover an unseen test corpus. We have also evaluated those grammars according to supertagging accuracy. We have experimented with the notion of cutoff grammar, and seen that these grammars are more compact and yet yield little in the way of supertagging accuracy. We have introduced a preliminary technique for extending an extracted grammar using an external resource, namely, the tree families of XTAG. We have seen that this technique expands the coverage of an extracted grammar, and discussed how this technique may be developed in order to achieve better results.

There are a number of ways to extend this work of extracting an LTAG from the Penn Treebank. Because our goal is to develop a grammar around which to base statistical models of parsing, we are in particular interested in better procedures for extending the extracted grammars. Besides merely extending a grammar, it is also necessary to develop a method for estimating how often trees that are unseen in the training corpus, but are part of the extended grammar, are expected to occur in test data. After such issues are resolved, the extracted grammar could be used in a probabilistic model for LTAG, as delineated by [Schabes, 1992] and [Resnik, 1992]. This would provide not only another means of comparing different varieties of extracted grammar, but would also allow comparison of LTAG parsing against the many other lexicalized parsing models mentioned in the introduction.

Acknowledgements

This work was supported by NSF grants #SBR-9710411 and #GER-9354869.

References

[Charniak, 1996] Charniak, E. (1996). Tree-bank grammars. Technical Report CS-96-02, Brown University, Providence, RI.

[Chen et al., 1999] Chen, J., Bangalore, S., and Vijay-Shanker, K. (1999). New models for improving supertag disambiguation. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics*, Bergen, Norway.

[Collins, 1996] Collins, M. (1996). A new statistical parser based on bigram lexical dependencies. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics.*

[Collins, 1997] Collins, M. (1997). Three generative lexicalized models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics.*

[Magerman, 1995] Magerman, D. M. (1995). Statistical decision-tree models for parsing. In *Proceedings of the 33th Annual Meeting of the Association for Computational Linguistics.*

[Marcus et al., 1994] Marcus, M., Kim, G., Mary, Marcinkiewicz, MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The penn treebank: Annotating predicate argument structure. In *Proceedings of the 1994 Human Language Technology Workshop*, pages 110–115.

[Marcus et al., 1993] Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.

[Neumann, 1998] Neumann, G. (1998). Automatic extraction of stochastic lexicalized tree grammars from treebanks. In *Proceedings of the Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 120–123.

[Resnik, 1992] Resnik, P. (1992). Probabilistic tree-adjoining grammar as a framework for statistical natural language processing. In *Fifteenth International Conference on Computational Linguistics (COLING 92)*, pages 418–424.

[Schabes, 1992] Schabes, Y. (1992). Stochastic lexicalized tree-adjoining grammars. In *Fifteenth International Conference on Computational Linguistics (COLING 92)*, pages 426–432.

[Schabes et al., 1988] Schabes, Y., Abeillé, A., and Joshi, A. K. (1988). Parsing strategies with 'lexicalized' grammars: Application to tree adjoining grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary.

[Srinivas, 1997] Srinivas, B. (1997). Performance evaluation of supertagging for partial parsing. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 187–198, Cambridge, MA.

[Xia, 1999] Xia, F. (1999). Extracting tree adjoining grammars from bracketed corpora. In *Fifth Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, Beijing, China.

[XTAG-Group, 1995] XTAG-Group, T. (1995). A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS 95-03, University of Pennsylvania. Updated version available at http://www.cis.upenn.edu/~xtag/tr/tech-report.html.

FROM CASES TO RULES AND VICE VERSA: ROBUST PRACTICAL PARSING WITH ANALOGY

Alex Chengyu Fang
Department of Phonetics and Linguistics
University College London
Wolfson House, 4 Stephenson Way
London NW1 2HE, England
alex@phon.ucl.ac.uk

This article describes the architecture of the Survey Parser and discusses two major components related to the analogy-based parsing of unrestricted English. Firstly, it discusses the automatic generation of a large declarative formal grammar from a corpus that has been syntactically analysed. Secondly, it describes analogy-based parsing that employs both the automatically learned rules and the database of cases to determine the syntactic structure of the input string. Statistics are presented to characterise the performance of the parsing system.

1 Introduction

As the title indicates, this article describes two components related to the parsing of unrestricted English. Firstly, it discusses the automatic generation of a large declarative formal grammar from a collection of pre-analysed sentences of English. Secondly, it describes a parsing methodology that employs both the automatically learned rules and the database of cases to determine the syntactic structure of the input string. The discussions will be based on the Survey Parser that has been implemented by the author (Fang 1996a), in the course of which some of the statistics will be presented to characterise the parsing approach to be reported here.

1.1 Background

In 1993-1996, the Survey of English Usage of University College London was engaged in the machine-aided syntactic analysis of the mega-word British Component of the International Corpus of English (ICE-GB; Greenbaum 1988 and 1996). The corpus comprises 600,000 words of transcribed speech and 400,000 words of writing. The analysis of the corpus included wordclass tagging and syntactic parsing. Each word in the corpus is assigned a contextually appropriate tag from a set of 270 grammatically possible tag-feature combinations. The parsing scheme specifies the analysis for the category names (covering the clause and the canonical phrases) and their syntactic functions such as subject, verb, and direct object. Both the tagging and parsing schemes were based on work by the TOSCA group of Nijmegen University, the Netherlands (Oostdijk 1991) but substantially modified for the project. ICE-GB initially used a parser that required certain amount of manual pre-processing of the input text. For instance, all cases of coordination had to be manually marked and indicated. The parser then produced all the possible analyses for each sentence, one of which was to be selected and modified if necessary as the correct representation of the constituent structure. About 70% of the corpus was parsed before it became clear in 1995 that the residue represented syntactic constructions beyond the capability of the parser. A parser that I had been developing since 1994 was used instead. Together with a Windows-based graphic tree editor, the parser completed the analysis of the corpus in 1996. It is now known as the Survey Parser, whose further development was supported in 1995 for two years by the Engineering and Physical Sciences Research Council, UK.

1.2 Design Requirements

Because of the specific needs arising from the analyses of the ICE corpus, the design of the Survey Parser was conditioned by the following requirements:

- *Speed* – The project required that input strings be batch parsed overnight so that researchers could supervise and modify the analyses the following day. Since each researcher was assigned several texts a time and since there were about six such people, the parser was typically required to batch parse about 20 texts at a time, which represented 40,000 words. In addition, because of the correction of wrong wordclass tags, researchers needed to frequently reparse individual sentences.
- *Robustness* – Because of the unsupervised batch parsing overnight of unrestricted English, the parser was required to be capable of handling situations where the input string represents linguistic constructions beyond the descriptive power of the formal grammar. In practice, the parser should be robust enough to produce a partial analysis when a complete parse could not be achieved.
- *One analysis per input string* – The experience was that it took shorter for the researcher to modify one incorrect analysis than to select from a number of possible analyses. This required the parser to produce one analysis that entails minimal manual intervention. To achieve this, the parser should either ensure that the first solution is a good one or be able to compute the best analysis from the competing ones.
- *Ability to adapt to new grammatical constructions* – The parser should be able to 'learn' and generalise about new grammatical constructions not described by the current grammar. In practice, this meant that the parser should be able to analyse a similar construction once the construction was manually analysed.

1.3 Analogy-Based Parsing

The parsing approach adopted in the Survey Parser seems to meet the above requirements. Briefly speaking, the parsing methodology may be roughly described as analogy-based, variously discussed under case-, explanation-, and example-based learning (Mitchell et al 1986; Minton 1988; van Harmelen and Bundy 1988; Knodner 1993). In the light of analogy-based approach to problem solving, solutions to old problems can be used for new but similar problems. In terms of parsing, this approach is conceptually very simple: given a database of input strings that have already been syntactically analysed, the parsing of a new string is seen as identifying a same or similar case in the database. Once the similarity is established, the analysis stored in the database is then transferred onto the new input string. This approach to parsing has been explored by Samuelsson and Rayner (1991), Neumann (1994), Samuelsson (1994), and Srinivas and Joshi (1995).

Such a parsing methodology requires a collection of syntactically analysed sentences as a case base and a mechanism to establish the similarity between the input string and one of the cases. Since a string may be represented as a sequence of words, a sequence of wordclass tags, or a sequence of grammatical phrases, there are three obvious options to establish such similarity: Two sentences are judged as structurally identical or similar if there is an exact match in terms of lexical items, wordclass tags, or phrases. Intuitively, these three matching criteria have different levels of generalisability or coverage. The use of wordclass tags as a measurement of similarity, for instance, should have a higher chance of finding a match than the use of lexical items because of the data sparsity problem that is typically related to word sequences, a problem that has been extensively addressed within the speech recognition community. The use of phrase sequence will, in turn, represent a more generalised model to measure sentence similarities. A related problem, however, is that a more general model tends to produce less reliable similarity indications. A match at the phrase level is less a guarantee than a match at the wordclass level that the two sentences are structurally similar or identical. The simple phrase sequence NP VP NP, for example, has at least three different syntactic structures according to the ICE-GB scheme. Thus a practical issue in analogy-based parsing is to increase the coverage of the case base while maintaining an acceptable degree of confidence that the retrieved syntactic structure is a good one.

1.4 An Overview of the Survey Parser

Generally put, the Survey Parser establishes analogy or similarity between the input string and a case in the knowledge base through a match at the phrase level. To ensure an acceptable degree of confidence, the phrase sequence is constrained by features inherited from the lexical properties of the head. The parser has two major components. The first is the syntactic knowledge base constructed on the basis of the syntactically analysed ICE-GB, from which we may automatically extract bi-gram wordclass transitional probability, phrasal rules anchored to wordclass tags or terminal symbols, and clausal rules anchored to phrase types. Each phrasal or clausal rewrite rule is associated to a tree structure. The parsing component comprises wordclass analysis (tagging), phrasal analysis, and clausal analysis. The stochastically selected tags serve as indexes that allow for the retrieval of a sub-tree for any phrase identified by the phrasal rules in a left-to-right longest match manner. At the stage of clausal analysis, phrase types are used as indexes to identify a similar clause and retrieve the tree structure as the proposal analysis for the input string. When the process fails to find an identical clause from the database, the sequence of phrases with their internal structures analysed is treated as an intermediate or partial analysis. The architecture described above is illustrated by Figure 1, where double arrows indicate system queries to components of the knowledge base.

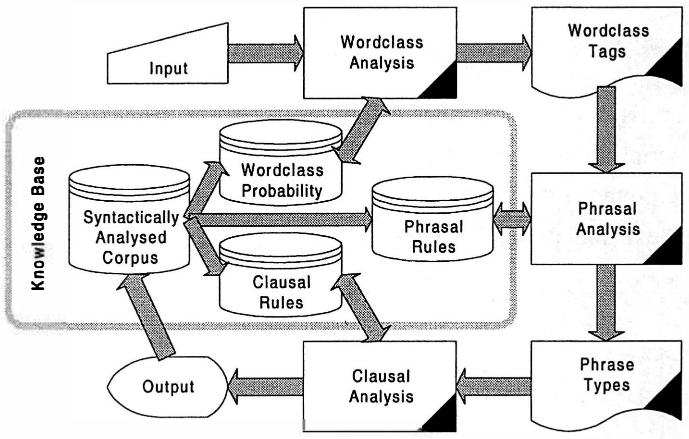


Figure 1: The major components in the Survey Parser

The following discussions will be divided into three parts. I shall first of all describe the construction of the database, which is a process of automatic extraction of syntactic rules. I shall then describe the various analyses performed on the input string, including wordclass tagging and phrasal and clausal analyses. Finally, statistics will be presented to characterise the performance.

2 The Construction of the Syntactic Knowledge Base[1]

The syntactic knowledge base consists of two components: phrase structure (*PS*) rules and phrase structure cluster (*PSC*) rules. The purpose of *PS* rules is to analyse sequences of wordclass tags into grammatical phrases, while *PSC* rules mainly handles sequences of grammatical phrases and assigns the final hierarchical structure.

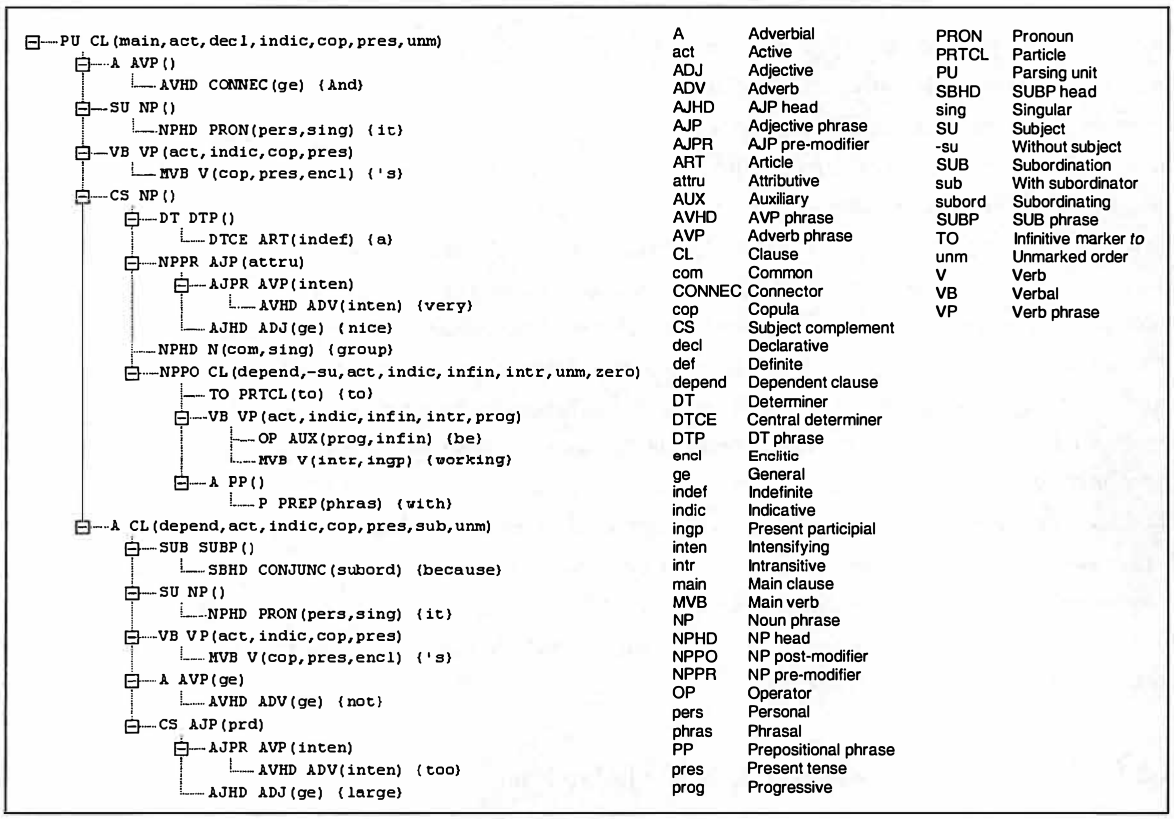
[1] See also Fang (1996c) for additional information.

2.1 Phrase Structure Rules

PS rules determine the analysis of wordclass sequences into phrases including noun phrases (NP), verb phrases (VP), adjective phrases (AJP), adverb phrase (AVP), and prepositional phrase (PP). The automatic generation of such rules is achieved by collecting all the tags as terminal symbols attributed to a particular phrase. Since the syntactic analyses of ICE-GB explicitly specify the boundaries of constituent structures as well as their syntactic functions, the extraction is a fairly straightforward matter. As a general rule, complementation and post-modification are not included. Thus, for instance, PS rules describing NPs all terminate at the head of the phrase; similarly those describing VPs all terminate at the main verb. Differently, however, PS rules for PPs cover the complete span of the phrase. Here is an example illustrating the extracting process.

[1] *And it's a very nice group to be working with because it's not too large*

The syntactic tree for [1] is graphically represented in Figure 2, where each constituent has two elements of description, the first being the name of syntactic function and the second that of category type. Thus, **SU NP()** is interpreted as "noun phrase functioning as clausal subject". As another example, **AVHD CONNEC(ge) {And}** is read as "lexical item *And* is a general connective and functions as the head of the adverb phrase".



```
⊟---PU CL(main,act,decl,indic,cop,pres,unm)
      ⊟---A AVP()
      |     └---AVHD CONNEC(ge) {And}
      ⊟---SU NP()
      |     └---NPHD PRON(pers,sing) {it}
      ⊟---VB VP(act,indic,cop,pres)
      |     └---MVB V(cop,pres,encl) {'s}
      ⊟---CS NP()
      |     ⊟---DT DTP()
      |     |     └---DTCE ART(indef) {a}
      |     ⊟---NPPR AJP(attru)
      |     |     ⊟---AJPR AVP(inten)
      |     |     |     └---AVHD ADV(inten) {very}
      |     |     └---AJHD ADJ(ge) {nice}
      |     |---NPHD N(com,sing) {group}
      |     ⊟---NPPO CL(depend,-su,act,indic,infin,intr,unm,zero)
      |           |---TO PRTCL(to) {to}
      |           ⊟---VB VP(act,indic,infin,intr,prog)
      |           |     |---OP AUX(prog,infin) {be}
      |           |     └---MVB V(intr,ingp) {working}
      |           ⊟---A PP()
      |                 └---P PREP(phras) {with}
      ⊟---A CL(depend,act,indic,cop,pres,sub,unm)
            ⊟---SUB SUBP()
            |     └---SBHD CONJUNC(subord) {because}
            ⊟---SU NP()
            |     └---NPHD PRON(pers,sing) {it}
            ⊟---VB VP(act,indic,cop,pres)
            |     └---MVB V(cop,pres,encl) {'s}
            ⊟---A AVP(ge)
            |     └---AVHD ADV(ge) {not}
            ⊟---CS AJP(prd)
                  ⊟---AJPR AVP(inten)
                  |     └---AVHD ADV(inten) {too}
                  └---AJHD ADJ(ge) {large}
```

A	Adverbial	PRON	Pronoun
act	Active	PRTCL	Particle
ADJ	Adjective	PU	Parsing unit
ADV	Adverb	SBHD	SUBP head
AJHD	AJP head	sing	Singular
AJP	Adjective phrase	SU	Subject
AJPR	AJP pre-modifier	-su	Without subject
ART	Article	SUB	Subordination
attru	Attributive	sub	With subordinator
AUX	Auxiliary	subord	Subordinating
AVHD	AVP phrase	SUBP	SUB phrase
AVP	Adverb phrase	TO	Infinitive marker *to*
CL	Clause	unm	Unmarked order
com	Common	V	Verb
CONNEC	Connector	VB	Verbal
cop	Copula	VP	Verb phrase
CS	Subject complement		
decl	Declarative		
def	Definite		
depend	Dependent clause		
DT	Determiner		
DTCE	Central determiner		
DTP	DT phrase		
encl	Enclitic		
ge	General		
indef	Indefinite		
indic	Indicative		
ingp	Present participial		
inten	Intensifying		
intr	Intransitive		
main	Main clause		
MVB	Main verb		
NP	Noun phrase		
NPHD	NP head		
NPPO	NP post-modifier		
NPPR	NP pre-modifier		
OP	Operator		
pers	Personal		
phras	Phrasal		
PP	Prepositional phrase		
pres	Present tense		
prog	Progressive		

Figure 2: The tree structure for Example [1]

From the analysis of [1], PS rules for the five major phrases can be extracted and stored in the syntactic knowledge base. Each rule comprises a sequence of wordclass tags and is associated to a constituent structure that specifies the analysis of the sequence. Table 1 illustrates such rules from [1].

Phrase	*Rule*	*Associated constituent structure*	*Example*

AJP	ADV(inten)
	ADJ(ge)
		⊟--- AJP(prd)
		⊟---AJPR AVP(inten)
		└---AVHD ADV(inten) {-}
		└---AJHD ADJ(ge) {-}
			too
			large

AVP	CONNEC(ge)	⊟--- AVP() └--- AVHD CONNEC(ge) {-}	furthermore
	ADV(ge)	⊟--- AVP(ge) └--- AVHD ADV(ge) {-}	briefly
NP	PRON(pers,sing)	⊟--- NP() └---NPHD PRON(pers,sing) {-}	it

	ART(indef)
	ADV(inten)
	ADJ(ge)
	N(com,sing)
		⊟--- NP()
		⊟---DT DTP()
		└---DTCE ART(indef) {-}
		⊟---NPPR AJP(attru)
		⊟---AJPR AVP(inten)
		└---AVHD ADV(inten) {-}
		└---AJHD ADJ(ge) {-}
		└---NPHD N(com,sing) {-}
			a
			very
			nice
			group

PP	PREP(phras)	⊟--- PP() └---P PREP(phras) {-}	with
VP	V(cop,pres,encl)	⊟--- VP(act,indic,cop,pres) └---MVB V(cop,pres,encl) {-}	's

	AUX(prog,infin)
	V(intr,ingp)
		⊟--- VP(act,indic,infin,intr,prog)
		├---OP AUX(prog,infin) {-}
		└---MVB V(intr,ingp) {-}
			be
			working

Table 1: PS rules automatically extracted from [1]

2.2 Phrase Structure Cluster Rules

The second component of the syntactic knowledge base deals with phrase structure clusters and superimposes the hierarchical structure of this cluster. In most of the cases, these clusters correspond to the conventional clause, but occasionally they represent co-ordinated or juxtaposed phrases. Again, such rules may be automatically extracted from a set of pre-analysed sentences. For example, the syntactic analysis of [1] yields one PSC rule as represented in Table 2:

Type	*PS Cluster*	*Associated Tree*
Clause	AVP NP VP:cop NP TO VP:intr:infin PP:ps CONJUNC:subord NP VP:cop AVP AJP	⊟---PU CL(main,act,decl,indic,cop,pres,unm) ⊞---A AVP() ⊞---SU NP() ⊞---VB VP(act,indic,cop,pres) ⊟---CS NP() ⊟---NPPO CL(depend,-su,act,indic,infin,intr,unm,zero) ---TO PRTCL(to) {to} ⊞---VB VP(act,indic,infin,intr,prog) ⊞---A PP() ⊟---A CL(depend,act,indic,cop,pres,sub,unm) ⊞---SUB SUBP() ⊞---SU NP() ⊞---VB VP(act,indic,cop,pres) ⊞---A AVP(ge) ⊞---CS AJP(prd)

Table 2: A phrase cluster automatically extracted from [1]

As mentioned at the beginning, analogy-based parsing has two practical issues: confidence and coverage. Confidence is the system assurance that the retrieved tree structure for the input string is a good one while coverage is the adaptation of the indexed cases so that they are useful to as many structural variations as possible. In the Survey Parser, confidence is maintained through the use of feature constraints and the increase of coverage is achieved through the identification and removal of non-obligatory syntactic elements.

Feature Constraints

To ensure the correct association between the PS cluster and the corresponding tree, some of the phrase types are normally described or restricted with features. This typically applies to VPs, whose sub-categorisation determines the analysis of their complements. For example, the phrase cluster NP VP PP may have at least two different analyses for the complementing PP: as subject complement if the VP is copula and as adverbial if the VP is intransitive. In the case of a non-finite VP, the very same phrase cluster needs to be analysed as a noun phrase post-modified by a non-finite clause, e.g., *countries pressurised by the decision* and *countries voting against the decision.* Feature constraints inherited from the main verb help to dissolve such ambiguities. In Table 2, the first and second VPs are described by **`cop`**, a feature name meaning *copula.* This feature ensures that the complementing NPs are correctly analysed as subject complement (**`CS`**). The other two constraint features for VPs are **`intr`** (intransitive) and **`tr`** (transitive). Non-finite VPs are described with additional features to indicate their forms, e.g., **`infin`** for infinitive, **`edp`** for past participle, and **`ingp`** for present participle.

Non-obligatory Elements

Non-obligatory elements include AVPs and PPs that do not complement any particular VPs. They are called non-obligatory in the sense that their removal does not affect the overall syntactic structure of the sentence. In order to maximise the coverage of phrase cluster rules, such elements are removed. The example in Table 2, for instance, is in fact written as `NP VP:cop NP TO VP:intr:infin PP:ps CONJUNC:subord NP VP:cop AJP.`

Adverbial clauses may also be treated as non-obligatory and, indeed, they are probably the most active constructions that contribute to the complexity of the clause. There is very good reason to expect a greatly increased coverage if such clauses could be treated separately and removed from the host clause. For the time being, however, this has not been implemented in the Survey Parser.

3 Parsing with PS and PSC Rules

The Survey Parser has three major modules that handle (1) assigning a wordclass tag to each item in the input string, (2) chunking the tags into a *PSC*, and (3) querying in the knowledge base for possible analyses for this *PSC*.

3.1 The Analysis of Wordclasses

The Survey Parser currently uses AUTASYS (Fang and Nelson 1994; Fang 1996b) as a pre-processor that tokenises the input string into lexical items and then assigns one wordclass tag to each of the tokens. This tagger has a probabilistic backbone supported by a list of rules in order to achieve the informationally rich tagset designed for the ICE-GB project. The tagset features 22 general wordclasses with around 70 descriptive features, totalling about 270 grammatically possible tags (Fang 1994; Greenbaum and Ni 1994). The descriptive features represent a detailed system of lexical sub-categorisation critical for parsing with wide-coverage lexicalised grammars (Briscoe and Carroll 1997). Consider

[2] *The search menu in the Circulation module may make additional search methods available to library staff.*

AUTASYS assigns one ICE tag to each of the lexical items in [2] and the result is illustrated in Table 3. It is worth noting that the tagging process provides the maximum grammatical information at this stage. For example, all compound nouns have already been marked up with ‘ditto tags’ that carry sequential numbers to indicate the boundary of the compound. The grammatical features related to the compound are selected according to the head.

As a result, *search* in the compound noun *search methods* is tagged as a plural common noun, the first in the two-item sequence. Lexical verbs are also analysed for detailed sub-categorisations and, as Figure 3 shows, there are 7 different types of verbs in the ICE tagging scheme. The verb *make* in [2], for instance, is tagged as infinitive ditransitive though it should be correctly analysed as complex transitive complemented by both a NP and an AJP.

The	**ART(def)**
search	**N(com,sing):1/2**
menu	**N(com,sing):2/2**
in	**PREP(ge)**
the	**ART(def)**
Circulation	**N(com,sing):1/2**
module	**N(com,sing):2/2**
may	**AUX(modal,pres)**
make	**V(ditr,infin)**
additional	**NUM(ord)**
search	**N(com,plu):1/2**
methods	**N(com,plu):2/2**
available	**ADJ(ge)**
to	**PREP(ge)**
library	**N(com,sing):1/2**
staff	**N(com,sing):2/2**
.	**PUNC(per)**

Table 3: Grammatical tagging of [2]

As mentioned in Section 2.2 where feature constraints are discussed, VPs are described as copula, intransitive, and transitive only. Detailed transitivity information for the verb is mainly for passivised VPs, which undergo the following valency shifts: mono-transitive→intransitive, complex-transitive→copula, ditransitive→mono-transitive. The transitivity of a passivised complex-transitive VP, for example, is shifted to that of copula in order to cater for the analysis of the complementiser as subject complement, e.g., *The home front was kept ignorant of the reality.* Detailed verb transitivity sub-categorisation is of great use when the system fails to find a global analysis for the input string and has to label syntactic functions from limited context.

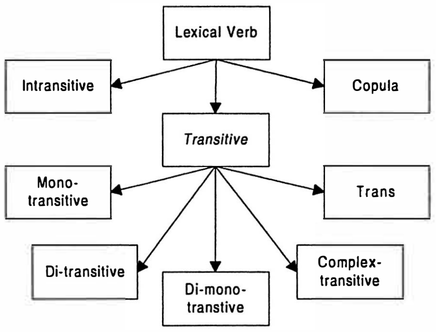
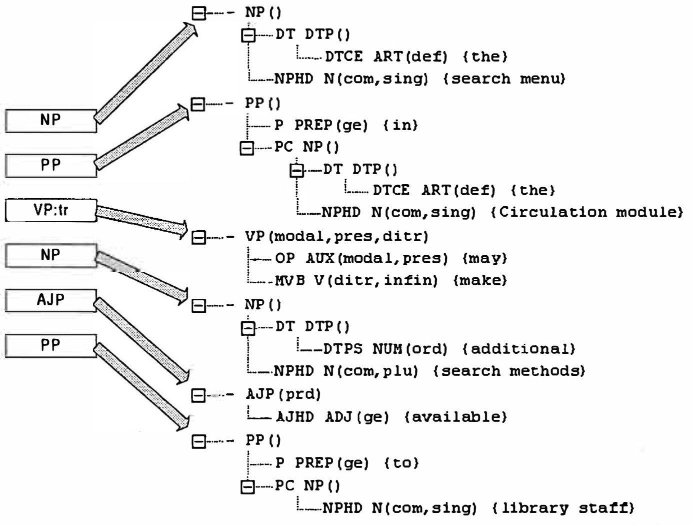


Figure 3: Subcategorisation of verbs

3.2 The Analysis of Phrases

The input string, as a sequence of wordclass tags, is then processed at the phrase level according to the *PS* rules, which are applied deterministically on a left-to-right longest match heuristic. When applied, these *PS* rules chunk the input string into a cluster of *PS*s, with feature information about the head. They also assign phrase types, boundaries, and internal structures to tag sequences that have a direct match in the PS rule base. Analysed at this level, the input string is represented as a cluster of syntactic phrases, each of which is now associated to a sub-tree. As demonstrated by Figure 4, the sub-trees already present a neat representation of the constituent structure of the phrases. Note that the VP is described by a feature (**tr**, meaning transitive) inherited from the corresponding wordclass tag except that there is no further distinction of sub-categorisation for transitive verbs (Figure 3). What still remains uncertain at this stage is the syntactic function to be determined by the PSC rules.



```
⊟--- NP()
     ⊟--DT DTP()
     |      └--DTCE ART(def) {the}
     └--NPHD N(com,sing) {search menu}
⊟--- PP()
     |--P PREP(ge) {in}
     ⊟--PC NP()
            ⊟--DT DTP()
            |      └--DTCE ART(def) {the}
            └--NPHD N(com,sing) {Circulation module}
⊟--- VP(modal,pres,ditr)
     |--OP AUX(modal,pres) {may}
     └--MVB V(ditr,infin) {make}
⊟--- NP()
     ⊟--DT DTP()
     |      └--DTPS NUM(ord) {additional}
     └--NPHD N(com,plu) {search methods}
⊟--- AJP(prd)
     └--AJHD ADJ(ge) {available}
⊟--- PP()
     |--P PREP(ge) {to}
     ⊟--PC NP()
            └--NPHD N(com,sing) {library staff}
```

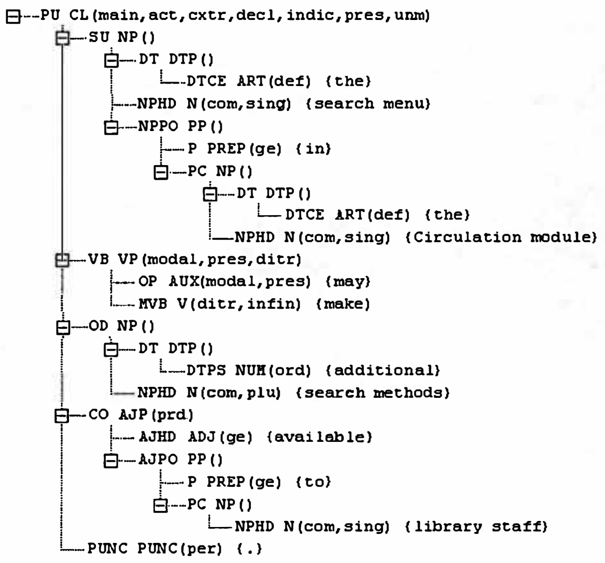
Figure 4: Input string as a PS cluster with associated sub-trees

3.3 The Analysis of Clauses

As a final step, the input string as a cluster of phrase structures is then queried in the database of PSC rules to see if an identical sequence can be located. With a positive feedback from the database, the associated tree structure for that sequence in the database is retrieved and used to specify the labelling and the attachment of the phrases of the input string. In our example, the parser determined that the PSC of the input string was the same as that of [3], already analysed and stored in the database:

[3] *The cellular anatomy of the peripheral nervous system renders it vulnerable to injury.*

Accordingly, the parser retrieved the analysis for [3] and superimposed it on [2]. The final analysis is shown in Figure 5. In this particular example, the parser successfully retrieved the correct tree structure for the input string. The incorrect sub-categorisation of the verb *make* as ditransitive at the stage of tagging did not prevent the parser from correctly labelling the sentence-final AJP as *object complement* (**CO**), suggesting the possibility of a post-parsing correction of wordclass tags.

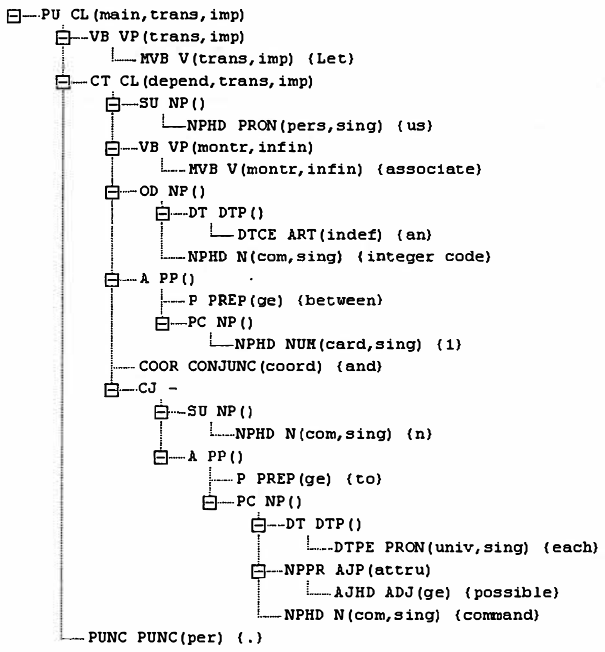


```
⊟--PU CL(main,act,cxtr,decl,indic,pres,unm)
    ⊟--SU NP()
    |   ⊟--DT DTP()
    |   |   └--DTCE ART(def) {the}
    |   |--NPHD N(com,sing) {search menu}
    |   ⊟--NPPO PP()
    |       |--P PREP(ge) {in}
    |       ⊟--PC NP()
    |           ⊟--DT DTP()
    |           |   └--DTCE ART(def) {the}
    |           └--NPHD N(com,sing) {Circulation module}
    ⊟--VB VP(modal,pres,ditr)
    |   |--OP AUX(modal,pres) {may}
    |   └--MVB V(ditr,infin) {make}
    ⊟--OD NP()
    |   ⊟--DT DTP()
    |   |   └--DTPS NUM(ord) {additional}
    |   └--NPHD N(com,plu) {search methods}
    ⊟--CO AJP(prd)
    |   |--AJHD ADJ(ge) {available}
    |   ⊟--AJPO PP()
    |       |--P PREP(ge) {to}
    |       ⊟--PC NP()
    |           └--NPHD N(com,sing) {library staff}
    └--PUNC PUNC(per) {.}
```

Figure 5: The final analysis of [2]

Partial Analysis

If a global analysis cannot be achieved, the parser will enter a *fallback mode*, where all the non-obligatory PPs are removed from the phrase cluster and a second attempt is made to find a match. When successful, the parser will paste the removed PPs back into the host clause. Failure in the fallback mode will then put the parser in the *partial mode*, where all the associated sub-trees in the phrase cluster are written out as a partial analysis. The boundaries and the internal structures of the component phrases have already been analysed and labelled. The parser also naively assigns missing names guess-estimated from neighbouring phrases.[2] Figure 6 is an example of a partial analysis. The major cause for the failure to find a global analysis was the exclusion of the coordinating conjunction *and* from the PP *between 1 and n*. The parser automatically inserted a **CT CL** node after the initial VP, indicating that what follows is a non-finite clause with an overt subject. This was correctly achieved



```
⊟--PU CL(main,trans,imp)
    ⊟--VB VP(trans,imp)
    |   └--MVB V(trans,imp) {Let}
    ⊟--CT CL(depend,trans,imp)
    |   ⊟--SU NP()
    |   |   └--NPHD PRON(pers,sing) {us}
    |   ⊟--VB VP(montr,infin)
    |   |   └--MVB V(montr,infin) {associate}
    |   ⊟--OD NP()
    |   |   ⊟--DT DTP()
    |   |   |   └--DTCE ART(indef) {an}
    |   |   └--NPHD N(com,sing) {integer code}
    |   ⊟--A PP()
    |   |   |--P PREP(ge) {between}
    |   |   ⊟--PC NP()
    |   |       └--NPHD NUM(card,sing) {1}
    |   |--COOR CONJUNC(coord) {and}
    |   ⊟--CJ -
    |       ⊟--SU NP()
    |       |   └--NPHD N(com,sing) {n}
    |       ⊟--A PP()
    |           |--P PREP(ge) {to}
    |           ⊟--PC NP()
    |               ⊟--DT DTP()
    |               |   └--DTPE PRON(univ,sing) {each}
    |               ⊟--NPPR AJP(attru)
    |               |   └--AJHD ADJ(ge) {possible}
    |               └--NPHD N(com,sing) {command}
    └--PUNC PUNC(per) {.}
```

Figure 6: A partial analysis

[2] As Fang (forthcoming) argues, it is feasible to determine the syntactic functions of, for instance, prepositional phrases according to lexical descriptions. It is also possible to label the syntactic functions of non-finite clauses in a similar fashion.

through the sub-categorisation of the antecedent VP as **`trans`**. Limited context and verb sub-categorisation information also enabled the correct labelling of the subject, the verb, and the direct object of the dependent clause.

4 Evaluation

In this section, I report empirical tests carried out to evaluate the performance of the Survey Parser. In particular, these tests were designed to indicate the coverage of the extracted grammar in terms of PS and PSC rules, labelling precision, the accuracy of analysis according to human judgements, and finally the processing speed.

4.1 Evaluating the Coverage of Phrase Structure Rules

The syntactic knowledge base currently makes use of about 50,000 syntactically analysed utterances from ICE-GB. Table 4 presents statistics about the rule extractions from the set.

Type	*AJP*	*AVP*	*NP*	*PP*	*VP*	*PSC*
No	103	49	3,695	8,974	987	178,045

Table 4: A summary of PS and PSC rules extracted from ICE-GB

To estimate the coverage of PS rules automatically extracted from ICE-GB, the analysed corpus was divided into 10 equal sets (1-10) of about 70,000 words each, with each individual set used as test data and the rest as training data in a rotating manner. Precautions were taken to make sure that the test set did not make up the training data. Figure 7 displays the coverage of VP rules when tested by the training data of varying sizes. The *Y*-axis represents the coverage (0.0-1.0), though the graph only displays a region of .95 to 1.0 as all the figures were higher than .95. The *X*-axis represents the size of the training data, which is an increment of one sample up to nine samples. Training data of one sample in size yields a coverage of over .95 for all the four testing samples. The coverage constantly rises with the increase of the training data size. The nine-sample training data produced a coverage of over .98 for the four testing samples used in the experiment. The start of plateau visibly rests on eight-sample training data. For discussions on the forms and sub-categorisations of these verbs, see Fang (1997).

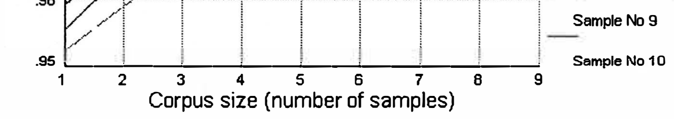


Figure 7: The coverage of VP rules extracted from ICE-GB

4.2 Evaluating the Coverage of Phrase Structure Cluster Rules

The corpora used for the evaluation included Air Travel Information System (ATIS), the Wall Street Journal (WSJ), and the Survey of English Usage Corpus (SEU; Fang and Nelson 1994), all different in terms of subject matter, style, and national variety. A total of 3,654 sentences were randomly selected to test the coverage of the PSC rules in order to estimate the probability of the input string as a phrase structure cluster to have a direct match in the database. According

Corpus	*Sent No.*	*Coverage*
ATIS	654	64.9%
WSJ	2,000	60.0%
SEU	1,000	60.3%

Table 5: The recall rates

to Table 5, such probability is 60% for test sets from WSJ and SEU. The ATIS set had a higher percentage mainly because of the relatively shorter sentence length in this corpus than the other two. Input strings that cannot be described by the PSC rules were given partial analyses.

4.3 Evaluating the Labelling Precision

A set of 60 AMALGAM sentences[3] from a computer manual were used to measure the precision of labelling by the Survey parser. A scoring program for evaluating the performance of speech recognition systems was used that measures not only the correct and wrong labels but also insertion and deletion rates. As Table 6 indicates, 90.1% of the constituents for the 60 sentences were correctly labelled by the Survey Parser. With wrong labels and deletion and insertion rates considered, the overall precision rate was 86.9%.

Sent No.	60	
Constituent No.	4,597	
Correct	4,142	90.1%
Wrong	272	5.9%
Deletion	183	4.0%
Insertion	147	
Overall	86.9%	

Table 6: Labelling precision

4.4 Evaluating the Accuracy of Analysis

Finally, the performance of the parser was subjected to the strictest human evaluation where an input string was judged to be wrong with a single parsing error in terms of labelling, attachment, or tokenisation. For this purpose, the test used a set of 117 dictionary definitions extracted from the Longman Dictionary of Contemporary English (see Briscoe and Carroll, 1991). Table 7 summarises the results. Of these input strings, 84 were fully parsed, a coverage of 71.8%. Of the fully parsed strings, 77 were correctly labelled and attached, a precision rate of 65.8% of the total number of input strings. It is significant that the majority of the fully analysed strings (91.7%) were correct even according to the strictest requirements, indicating a high level of system confidence that the proposed analysis is a good one.

Definition No.	117	
Full analysis	84	71.8%
Correct analysis	77	65.8%

Table 7: Accuracy of analysis

4.5 Evaluating the Processing Speed

Two sub-language corpora were used to measure the processing speech of the Survey Parser. One is a corpus of the English for science and technology collected at the Shanghai Jiao Tong University (JDEST; Huang 1991) and the other a corpus of the English of computer science collected at the Hong Kong University of Science and Technology (HKUST; Fang 1992). The statistics summarised in Table 8 were obtained with Dell OptiPlex Gxa. The tagging module, according to the test, ran at a speed of 6,012 words per second. The parsing module was able to process 177 words per second.

Corpus Source	*No. of words*	*No. of sentences*	*Tagging No. of seconds*	*Parsing No. of seconds*
JDEST	104,014	4,692	18	540
HKUST	70,331	4,297	11	443
Total	174,345	8,989	29	983

Table 8: The processing speed of the Survey Parser

[3] Available at http://www.scs.leeds.ac.uk/amalgam/amalgam/corpus/tagged/raw/ipsm_raw.html. The output from the Survey Parser for this set of sentences, manually checked and corrected by me, are available at http://www.scs.leeds.ac.uk/amalgam/amalgam/corpus/parsed/ice.html

5 Concluding Remarks

In this article, I have described the architecture of the Survey Parser as well as the construction of the syntactic knowledge base that comprises PS and PSC rules automatically extracted from a syntactically analysed corpus, ICE-GB. I then reported evaluation statistics that characterise the performance of analogy-based parsing. They indicate that while the clause structure rules have a coverage of 60%, the grammar has a high coverage in terms of phrase structures as the statistics for the verb phrase indicated. One conclusion we can draw here is that it is indeed feasible to automatically generalise a comprehensive formal grammar from a syntactically analysed corpus the size of ICE-GB and to apply it to large-scale practical parsing. A second conclusion is that analogy-based parsing enjoys a high degree of analysis precision, with over 90% of the constituents correctly labelled. When subjected to human inspection, 91.7% of the complete parses were found to be correct. Other observed advantages include high parsing speed and the ability to produce an analysis for every input string. A true strength of analogy-based parsing is its intrinsic ability to learn over the acquisition of new phrase structure clusters. Since the formal grammar can be automatically learned, the parser can easily adapt itself to new constructions. Unlike probabilistic grammars, the automatically constructed grammar can be visually supervised and manipulated because of its declarative nature. The empirical tests suggested that the analogy-based parsing reported here is capable of the design requirements outlined at the beginning of the article.

As can be envisaged at this stage, the future work on the analogy-based parser will focus on methods to increase the coverage of the clause cluster rules. I have already mentioned the removal of non-obligatory elements such as AVP and PP in order to increase the coverage. Another effective method, yet to be tested, is the segmentation of the input string into component finite clauses, which are then parsed individually and glued back into the host clause. The syntactic functions of the clauses can be reliably assessed independent of the host clause because of the conjunction markers. The key process to achieve this is an algorithm that automatically, and reliably, identifies the boundaries of component clauses.

Acknowledgement

The author would like to thank the three anonymous reviewers for their valuable comments on an earlier version. This article was supported in part by Engineering and Physical Sciences Research Council, UK, Grant No GR/L81406.

References

Briscoe, E., and J. Carroll. 1991. *Generalised Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars*. Technical Report No. 224. University of Cambridge.

Briscoe, E., and J. Carroll. 1997. Automatic extraction of subcategorization from corpora. In *Proceedings of the 5th ACL Conference on Applied Natural Language Processing, Washington, DC.* pp 356-363.

Fang, A.C. 1992. Building a Corpus of the English of Computer Science. In *English Language Corpora: Design, Analysis and Exploitation*, ed. by Aarts, de Haan and Oostdijk. Amsterdam: Rodopi.

Fang, A.C. 1994. ICE: Applications and Possibilities in NLP. In *Proceedings of International Workshop on Directions of Lexical Research, 15-17 August 1994, Beijing.* pp 23-44.

Fang, A.C. 1996a. The Survey Parser: Design and Development. In S. Greenbaum. pp 142-160.

Fang, A.C. 1996b. Grammatical Tagging and Cross-Tagset Mapping. In S. Greenbaum. pp 110-124.

Fang, A.C. 1996c. Automatically Generalising a Wide-Coverage Formal Grammar. In *Synchronic Corpus Linguistics*, ed. by C. Percy, C. Meyer, and I. Lancashire, Amsterdam: Rodopi. pp 207-222.

Fang, A.C. 1997. Verb Forms and Subcategorisations. In *Oxford Literary and Linguistic Computing, 12:4.*

Fang, A.C. forthcoming. A Lexicalist Approach towards the Automatic Determination for the Syntactic Functions of Prepositional Phrases. To appear in *the Journal of Natural Language Engineering.*

Fang, A.C., and G. Nelson. 1994. Tagging the Survey Corpus: a LOB to ICE experiment using AUTASYS. *ALLC Literary & Linguistic Computing,* 9:2. pp 189-194.

Greenbaum, S. 1988. A Proposal for an International Computerized Corpus of English. In *World Englishes 7, 315.*

Greenbaum, S. 1996. *The International Corpus of English.* Oxford: Oxford University Press.

Greenbaum, S., and Ni, Y. 1994. Tagging the British ICE Corpus: English Word Classes. In *Corpus-Based Research into Language*, ed. by N. Oostdijk and P. de Haan. Amsterdam: Rodopi. pp 33-45.

Huang, RJ. 1991. *A Technical Report of the JDEST Corpus Tagging Project.* Shanghai: Shanghai Jiao Tong University.

Kolodner, J. 1993. *Case-Based Reasoning.* San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Minton, S. 1988. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of 7th AAAI Conference, Saint Paul, Minnesota,* pp 564-569.

Mitchell, T., R. Keller, and S. Kedar-Carbelli. 1986. Explanation-based generalization: A unifying view. In *Machine Learning 1:1,* pp 47-86.

Neuman, G. 1994. Application of explanation-based learning for efficient processing of constraint-based grammars. In *10th IEEE Conference on Artificial Intelligence for Applications, San Antonio, Texas.*

Oostdijk, N. 1991. *Corpus Linguistics and the Automatic Analysis of English.* Amsterdam: Rodopi.

Samuelsson, C. 1994. Grammar specialization through entropy thresholds. In *32nd Meeting of the Association for Computational Linguistics, Las Cruces, New Mexico.*

Samuelsson, C. and M. Rayner. 1991. Quantitative evaluation of explanation-based learning as an optimization tool for large-scale natural language system. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence, Sydney, Australia.*

Srinivas, B. and A. Joshi. 1995. Some novel applications of explanation-based learning to parsing lexicalized tree-adjoining grammars. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics (ACL-95), Morgan Kaufmann, San Francisco, 1995.*

van Harmelen, F. and A. Bundy. 1988. Explanation-based generalisation = Partial evaluation. In *Artificial Intelligence, 36.* pp 401-412.

A Transformation-based Parsing Technique With Anytime Properties

Kilian Foth, Ingo Schröder, Wolfgang Menzel
(foth | ingo | wolfgang@nats.informatik.uni-hamburg.de)
Fachbereich Informatik, Universität Hamburg
Vogt-Kölln-Straße 30, 22527 Hamburg, Germany

Abstract

A transformation-based approach to robust parsing is presented, which achieves a strictly monotonic improvement of its current best hypothesis by repeatedly applying local repair steps to a complex multi-level representation. The transformation process is guided by scores derived from weighted constraints. Besides being interruptible, the procedure exhibits a performance profile typical for anytime procedures and holds great promise for the implementation of time-adaptive behaviour.

1 Introduction

Parsing procedures always have to be designed around a number of pre-specified requirements which arise from specific conditions of the individual application area in mind. Text retrieval tasks, for instance, can be accomplished already with a rather shallow analysis whereas speed and fail-soft behaviour are of utmost importance. Grammar checking and foreign language learning applications, on the other hand, must provide for highly precise error detection capabilities but differ considerably in their coverage requirements. One of the most demanding combination of target specifications results from the development of spoken language dialogue systems. Since this task is an attempt to model central capabilities of the human language faculty, rather strong criteria have to be met in order to achieve natural communicative behavior on a competitive level:

- **Robustness:**

  A spoken language dialogue system is typically confronted with a rich variety of linguistic constructs and will almost inevitably have to deal with extragrammatical input sooner or later. Also, repairs, hesitations, and other grammatical deviations will frequently produce ungrammatical utterances, while the recognition uncertainty inherent in spoken language input further increases the ambiguity.

  The parsing component must be able to cope with these problems in a robust way. Besides being able to return (possibly partial) analyses even for unexpected and arbitrarily distorted input it is also necessary to provide some kind of measure of how sure the parser is about its results.

- **Complete disambiguation:** Natural language utterances typically exhibit ambiguity when treated in isolation. Nevertheless, a simple enumeration of different (structural) readings almost never can be considered a sensible contribution to a practical language processing task. Although interactive applications can engage the speaker in a kind of clarification dialogue, usually this possibility brings many additional complications and should only be considered a measure of last resort. Instead, a well-designed system should make use of all the available information to obtain a single interpretation of the utterance, which is only abandoned if the user explicitly signals a communication failure.

- **Multiple-source disambiguation:** A vast variety of knowledge sources can contribute to disambiguation: Syntactic constraints, semantic preferences, prosodic cues, domain knowledge, the dialogue history, etc. All the available knowledge should be put to use as soon as possible so that local ambiguity will not create a large space of useless hypotheses during processing. For reasons of perspicuity and accessibility the integration of these knowledge sources should be organized in a way which maintains their modularity. Only then can the respective contributions of individual components be evaluated and properly balanced against each other.

- **Time-aware behavior:** Three closely related aspects must be considered with respect to the temporal behavior of a language processing component: Efficiency, incremental processing and temporal adaptivity. Whereas efficiency always has been an issue of major concern, explorations into incremental and time-adaptive parsing attracted more attention only recently [GKWS96, Amt99, Men94]. Since speech unfolds in time, speaking time is a valuable resource and an immediate response capability can be achieved only if incoming information is processed in an on-line fashion. Temporal adaptivity, on the other hand, is the capability of a component to dynamically control its processing regime depending on how much time is available to complete the task. In principle, such an *anytime behaviour* can be achieved by trading time against quality. Therefore a baseline performance will be required which allows the quality of available results to grow monotonically as more effort is made, and which is robust enough so that results of slightly reduced quality can still be considered being acceptable in a certain sense.

  Obviously, the most natural measure of external temporal pressure is given by the speaking rate of the dialogue partner. Thus temporal adaptivity makes sense first of all under an incremental processing scheme. Its basic mechanisms, however, can also be studied in the far simpler non-incremental case.

This paper investigates a non-standard parsing approach, which attempts to reconcile two different kinds of robustness, namely robustness against unexpected and ill-formed input and robustness against external temporal pressure. It is based on the application of constraint satisfaction techniques to the problem of structural disambiguation and allows the parser to include a wide variety of possibly contradicting informational contributions. Different solution procedures are presented and compared taking into account solution quality and the observed temporal behavior.

2 Parsing As A Consistent Labeling Problem

Although most contemporary unification-based grammars can be said to employ constraints, none of them fulfill the traditional definition of a Constraint Satisfaction Problem (CSP) consisting of a fixed number of *variables*, which receive their *values* from *domains*, i. e. sets of alternative value assignments. The global consistency of a value assignment is defined by means of local *constraints*, which can be understood as sets of admissible value tuples, specified intensionally.[1] A *conflict* then can be defined as a tuple outside a constraint, and a *solution* is a complete value assignment without conflicts. Finally, Consistent Labeling Problems (CLP) are characterized by their domains being discrete, finite and known in advance.

Parsing a given utterance can be viewed as an instance of such a problem in that every word form fulfills a specific function (i. e. complement, specifier, etc.) in a particular analysis. As long as a grammar theory is defined over a finite number of such functions, they can immediately be used as the values of a CSP. Since an utterance often contains multiple instances of the same grammatical function (i. e. a determiner modifying a noun), it always has to be specified in what context the word form fulfills its function. Therefore, every word form in an utterance is annotated with a *pair* consisting of a label (i. e. the grammatical function of the form) and a pointer to another word form, which is modified by the first one. This way, an entire dependency tree can be encoded in a CSP of a fixed size.

Conversely, some word forms require other word forms to fulfill certain functions for them as obligatory complements. Such valence requirements can be modeled as constraint requirements as well. Furthermore additional relations between word forms can be included easily (e. g. semantic arguments or aspects of the informational structure), thus creating a fairly rich structural representation.

Figure 1 shows an example of such a multi-level representation including syntactic dependencies (the arcs above the word forms), semantic arguments (e. g. the arcs labeled agens and patiens) and a bunch of valence requirements (all those with the label prefix n_, e. g. n_subj, n_inf etc.). Admissible combinations between dependency relations on two different representational levels are also defined by means of suitable constraints.

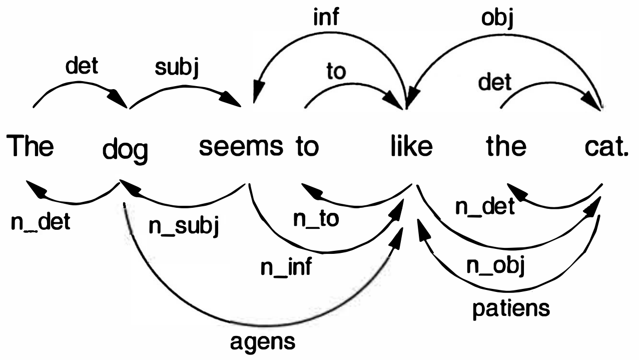


Figure 1: A CDG analysis

The notion of a constraint dependency grammar was first introduced by Maruyama [Mar90]. Harper et al. [HH94] extended the idea in order to cope with ambiguity arising from lexicon lookup and speech recognizer uncertainty. Menzel [MS98a] incorporated soft constraints, i. e., constraints that can be violated by a valid solution if no other solution can be found otherwise. For this purpose, constraints are annotated with a weight or score between zero and one that determines how easily that constraint may be violated. Hard constraints have a weight of zero

[1]From this perspective, the constraint-solving mechanism of a typed unification grammar, like the one underlying e. g. HPSG, might be considered a degenerate CSP with only a single variable, a domain consisting of all possible top-level feature structures, and constraints licensing *parts* of a complex value assignment.

and must not be violated by any solution. Constraints which reflect regularities of the grammar receive a small weight greater than zero while constraints that model mere preferences will be weighted close to one. A solution candidate to the CSP can then be assigned a score by determining the product of the weights of all the constraints which are violated somewhere in the structural description. Under these premises, parsing becomes a multidimensional optimization problem.

For reasons of efficiency at most binary constraints can be allowed, hence the universal expressive power of constraints is not available in practice. This serious shortcoming, however, can be neutralized to a certain degree by approximating higher order constraints using binary ones. Another disadvantage is the lack of a variable binding mechanism like the one which is provided by a unification operator together with the missing notion of a constituent (which, however, is shared with most dependency grammar approaches). Experience with grammar writing has confirmed that nevertheless nontrivial subsets of grammar can be encoded successfully, although some phenomena such as long-distance dependencies can only be modeled approximatively [SMFS].

Constraint dependency grammar is a purely declarative formalism. This property makes it amenable to a variety of problem solving strategies that can be compared, e. g. with respect to their temporal behaviour. The possibility to add further representational levels supports the integration of knowledge contributions from very different sources into a single solution space without sacrificing the strict modularity of the grammar and of the structural representation. Of course, this possibility is limited to only those knowledge sources that can meaningfully attach information to single word forms. A single interpretation of the incoming utterance can be obtained by using all available evidence, including minor preference indicators like ordering, distance or default cases. A truly ambiguous sentence will usually allow several analyses with only small differences between their scores, which can be ignored if desired.

The approach exhibits a remarkable robustness against unexpected and ill-formed input [MS98b], which obviously can be attributed to three important characteristics:

- the use of weighted constraints, which provides for the accommodation of conflicting evidence and therefore makes the analysis of deviating structures possible,
- the redundancy between loosely coupled representational levels, which allows conflicting information on one level to be overridden by sufficient evidence from a complementary one, and
- the possibility to license arbitrary categories as an acceptable, but in most cases highly disfavored, top node of a dependency tree, thus introducing a partial parsing scheme as a natural extension of the normal mode of operation.

Note that the resulting robust behavior follows immediately from the fundamental principles of the approach and no error rules or special operations become necessary.

Two other characteristics of the approach contribute to the rich potential for obtaining the desired anytime behaviour. In contrast to other parsing approaches the space of all possible analyses for constraint dependency parsing is always finite and very regularly structured. Parsing therefore becomes a process of selection between different analyses with virtually identical formal properties, which considerably facilitates their mutual comparison.

3 Solution Procedures

3.1 Consistency-based Methods

The canonical method for solving a constraint satisfaction problem is to establish a certain degree of consistency in it by deleting incompatible values from its domains, and then select an assignment to all constraint variables from the remaining values. This approach contrasts strongly with common parsing methods that are *constructive* in nature. Usually, grammatical structures are built up recursively from simpler structures, and ultimately from the information associated with the lexical items present in an utterance. Thus, the number of structures available increases over time. In contrast, the achievement of consistency is an *eliminative* process: The more progress is made, the fewer values remain in the problem. An attractive property of this kind of parsing is that it can be exactly determined, at any time, how much progress has already been made and how much work remains to be done until disambiguation is completed. This information will be of great use to a time-aware solution procedure.

Various well-defined degrees of consistency can be achieved in a CSP, and general algorithms exist to establish any desired degree of consistency. Consistency-based methods are used extensively in a constraint dependency parser by Harper et al. [HHZ+95]. However, this approach does not lend itself to robust processing of deviating input. Since consistency algorithms only remove those values that cannot appear in any solution, only hard constraints are effective. If a constraint grammar predominantly employs soft constraints, a consistency algorithm may remove very few values or none at all.

A suitable algorithm for consistency in a partial CSP should remove all those values that do not appear in the *optimal* solution—a property that is much more difficult to determine. The usual consistency algorithm will find a value that cannot appear in a solution by noting that it cannot appear in a valid n-ary assignment. A similar approach for partial CSP would be to select those values for deletion that only occur in n-ary assignments with low scores. The obvious method is to define a fixed limit and consider all scores below it unacceptable; this has much the same effect as employing the unmodified algorithms on a grammar in which more constraints are hard.

Another method known as *pruning* [MS98b] goes one step further. While a consistency algorithm cannot guarantee how much progress it will achieve, a pruning method will invoke a selection function at regular intervals to select exactly one value for deletion. If this function uses a fixed amount of time, an exact appraisal can be given not only of the amount of work to be done, but also of the actual runtime left until termination.

To guarantee that a value is selected for deletion within the allotted time, the selection function will usually have to be heuristic in nature. A simple selection function mimics the behaviour of a 2-consistency algorithm: Tables of mutual support are constructed for all pairs of domains in the problem. The support of a value v from another domain d can be defined as the maximal or the average compatibility of v with any value from d, or in a more elaborate way. The value whose maximal support by any other domain is smallest is selected for deletion.

Since the globally optimal solution may consist of values that are locally suboptimal, in general this method of assessing values exclusively by *local* information may remove the wrong values from a problem. While the CDG formalism ensures that the remaining values form a complete assignment, in general it cannot be guaranteed that this assignment will be the optimal solution, or even a grammatically valid one. Thus, a heuristic consistency algorithm may fail without result even though there is a valid solution, which defies the purpose of robust processing.

3.2 Enumeration

Most parsers use some kind of search algorithm to enumerate all alternatives for local or global ambiguities arising in the analysis of their input. A great number of search variants has been invented for different parsing applications (top-down vs. bottom-up parsing, depth-first vs. breadth-first search, linear vs. island parsing), and choosing the right method can have dramatic impact on the efficiency of a parser. In general, considering every possible alternative ensures that an algorithm is complete as well as correct, but may require so many resources that it becomes impractical to apply.

Since in a problem in CDG, consistency-based methods cannot guarantee either complete or correct behaviour, a complete method of solution is desirable even if it has other disadvantages. For instance, a complete but inefficient algorithm will still be of great use to the developer of a constraint grammar to check the validity of their model, or to verify the results of an incomplete method.

For the CSP, a complete search of all possible assignments can be conducted that is guaranteed to find the optimal solution.

In the partial CSP, a normal *best first* search can be employed which finds the optimal solution without ever having to expand a partial solution with a lower score. The current implementation of the CDG parser provides a straightforward best-first search in which the variables of a problem are instantiated in a fixed order. This will usually be the order of the words corresponding to the constraint variables. Compared with other parsers, this would be classified as a heuristically driven left-to-right search. It resembles bottom-up parsing in that each word can immediately be integrated into a tree that forms part of the complete dependency structure. A top-down parsing method could be simulated by arranging the search so that every additional dependency edge must modify a word that has already been analyzed, starting with those words that can modify the root of the dependency tree.

Since the CSP is $\mathcal{NP}$-complete, probably any complete solution method will have an exponential worst-case complexity. Although the actual runtime of a complete search algorithm is usually far below the worst possible case, and heuristic re-ordering of both domains and values can greatly improve the efficiency, it is difficult to predict even approximately how long a particular instance of the problem will take to search. Therefore, a complete search is inadequate as a solution method when time-aware behaviour is required. However, in contrast to other methods a complete search can easily enumerate globally near-optimal structures such as those defined by syntactically ambiguous sentences.

3.3 Transformation

An obvious way to overcome the unacceptable temporal behaviour of complete algorithms is to employ suboptimal methods. A strategy that works well in practice is that of heuristic repair. Rather than attempting to build the correct structure by selecting correct values step by step, this method first constructs an arbitrary dependency structure with errors in it and then tries to correct the errors.

Suppose that the dependency structure shown in Figure 2 has been constructed, in which both nouns have been analyzed as subjects of the same verb.

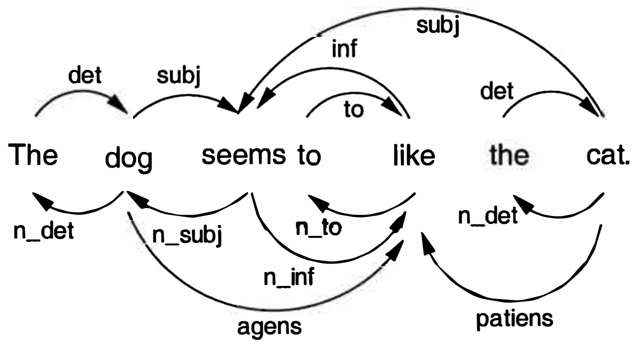


Figure 2: A suboptimal dependency analysis

Figure 3 a) shows part of the corresponding matrix of constraint variables. If a constraint exists that excludes multiple subjects, one of the two annotations with the label 'subj' must be replaced. By analyzing the word 'cat' as an object instead, this assignment can be transformed into the assignment shown as Figure 3 b). Note that when providing an object for the verb 'like' we also changed the corresponding value of the ancillary level OBJ to reflect the fact that the object required by the verb is actually present now. The new assignments represent the optimal analysis shown above as Figure 1.

a)	Syntax	SUBJ	OBJ	...
the$_1$	det/2	–	–	...
dog$_2$	subj/3	–	–	...
seems$_3$	–	n_subj/2	–	...
to$_4$	to/5	–	–	...
like$_5$	inf/3	–	–	...
the$_6$	det/7	–	–	...
cat$_7$	**subj/3**	–	–	...

$\Longrightarrow$

b)	Syntax	SUBJ	OBJ	...
the$_1$	det/2	–	–	...
dog$_2$	subj/3	–	–	...
seems$_3$	–	n_subj/2	–	...
to$_4$	to/5	–	–	...
like$_5$	inf/3	–	**n_obj/7**	...
the$_6$	det/7	–	–	...
cat$_7$	**obj/5**	–	–	...

Figure 3: Transformation of a dependency structure.

This method has several advantages as opposed to the previous ones:

- Because a complete dependency analysis is maintained at all times, the algorithm may be interrupted at any time and still return a meaningful answer, though not always the optimal one. Thus, it automatically fulfills a strong anytime criterion.
- The constraints that cause conflicts in a suboptimal assignment can suggest which value is inappropriate and what other value should be substituted.
- Because all analyses of a given utterance comprise the same number of values, it is guaranteed that the optimal solution (if any exists) can be constructed from any other assignment by successively replacing one value at a time.

A transformation step is usually defined as the exchange of one value of a constraint variable for another. By this definition, the correct solution of a CSP of degree n can always be reached in not more than n transformation steps, if the correct replacement value is chosen at any point. To accomplish this, however, *every* value in the problem has to be considered as an alternative in each step, whereas a backtracking search only has to try out all values from one particular domain. Obviously the transformation algorithm will encounter a much greater branching factor. However, the search space is now graph-shaped, and so not every alternative must be pursued further because the order in which several values are inserted into an analysis does not matter. Instead, the number of alternatives that are tried out at all can be used as a parameter to speed up the computation.

Obviously, the efficiency of such a repair algorithm depends on its ability to select the correct values for repair. Even a totally uninformed repair method can ultimately find the correct solution, since it is simply a random walk through the problem space. For better results, heuristic decision methods must be found to guide the selection. The simple *hill climbing* method will always choose the value that results in the best immediate improvement. The principal difficulty with this method is that it can get stuck in local optima of the problem space where no immediate improvement is possible.

Different methods exist to allow an algorithm to leave such misleading areas of the search space.

- Occasional downhill steps may be allowed so that an algorithm may escape from a local maximum. For example, in the popular method of *simulated annealing* downhill steps are allowed but gradually discouraged as analysis progresses.

- In another approach, hill climbing does not optimize the score of an analysis itself, but an ancillary cost function that is adapted in each step, so that the local optimum can be turned into an ascent.

- The definition of a transformation step can be changed so that several values may be replaced simultaneously. Assignments which differ in several variables will then become adjacent to the current assignment. Again, the number of values that may be changed in one step can be used as parameter to influence the speed of the algorithm.

A subsequent difficulty is that after such a repair algorithm has converged to the optimal solution, it may not terminate. If the optimal solution still causes some minor conflicts, the algorithm will continually try to repair these conflicts without success, since there is no simple way to distinguish a local optimum from the global one. In this case the individual constraints of the grammar can be used as a taboo criterion: If no repair step is allowed to re-introduce the conflict that prompted it in the first place, termination can be guaranteed.

Although repair-based solution methods cannot guarantee to find the optimal solution in all cases, in practice they achieve results comparable to those of exhaustive methods. This demonstrates that the values contained in a complete parse are helpful in selecting correct values even though some of them may themselves be incorrect [Minton92]. In this way, parsing by transformation can make use of global information without suffering the full combinatorial explosion of a complete search.

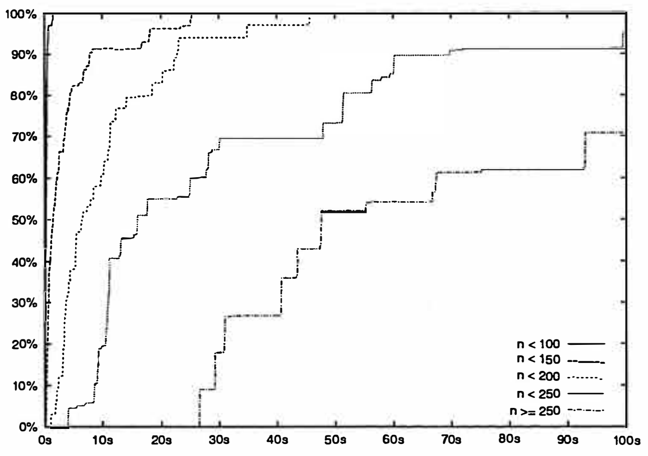


Figure 4: Solution quality as a function of time.

In a series of experiments, repair methods have been successfully applied to the the constraint parsing problem. The corpus comprised 200 consecutive utterances from the Verbmobil corpus of spoken dialogue utterances, segmented and slightly regularized by hand. Figure 4 illustrates the average temporal behaviour of a transformation-based algorithm grouped by the number n of constraint variables[2] in the resulting CSP. For each problem class, the average score after the indicated runtime is given relatively to the optimal score attainable.

With the observed increase of the solution quality the parser has a performance profile typical for anytime procedures [BD89]. In 90% of all problems, the solutions found were either identical to those found by a complete search or had even better scores (since the search uses a finite agenda, it may actually become incomplete in large problems). In the final analyses, 99.7% of all dependency links were established correctly.

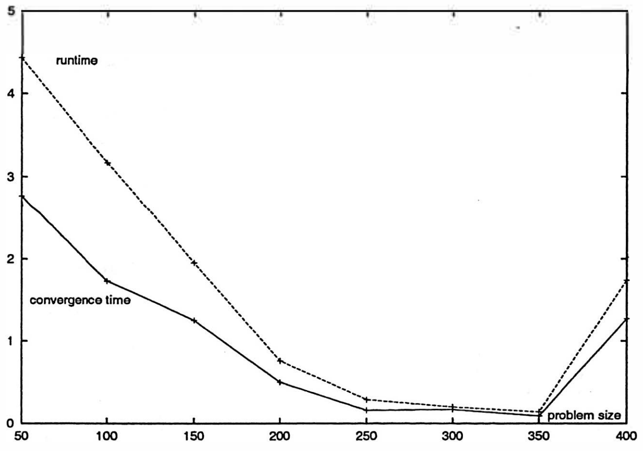


Figure 5: Ratio of runtime for transformation and search methods.

[2]In this particular constraint grammar, a sentence of five words will typically map to a CLP with about 100 variables.

In general, a near-optimal structure will be constructed after a short time. Finding the exact optimal analysis may take considerably more time in medium-sized and large optimization problems. However, particularly in these cases the algorithm will be consistently faster than a complete search. Figure 5 gives the average time that the transformation-based solution method takes to terminate, measured in terms of the runtime of a best-first search of the same problems. Clearly, the repair method is faster in most large problems except for the highest problem class (which only has two members in our corpus, however). Comparison of the time until the last successful repair with the total runtime shows that the algorithm will usually terminate not long after having reached the optimal analysis.

For applications that have to react to varying external time constraints, a parameter is provided that adjusts the number of alternatives to be tried out at each transformation step. In the cases investigated so far, decreasing this parameter can speed up the repair substantially, with an acceleration by a factor of about 3 reducing the accuracy to 86% of correct dependency links.

4 Related work

There is a striking analogy between constraint dependency parsing and customary approaches to the task of tagging natural language data: In both cases each word form in the utterance is annotated with a label from a finite set of alternatives and the approaches differ only in the information content of the labels. Although tagging usually means to classify word forms into (syntactic and semantic) types, the idea can also be extended to the use of functional tags in a straightforward way. Karlsson et al. [KVHA95] use such functional descriptions in Constraint Grammar parsing. Their tagset contains elements like "subject" or "a determiner modifying a noun to the right". Usually these tags are underspecified, because the exact identity of the modified item is unknown. Constraint Dependency Grammar as discussed in this paper extends this approach to fully specified structural representations. Since the identity of the modified word is now included into the composite tags, the size of the tagset additionally depends on the number of word forms in the given utterance. Moreover, the extension to multi-level disambiguation allows to treat considerably richer representations as compared to what a usual tagger is taking into account.

Another approach using complex tags is Supertagging [BJ99], where complex tree fragments are attached to word forms by means of a stochastic model. These tags represent considerably more structure than values in CDG, which correspond to single dependency edges. However, tags are treated in isolation and the compatibility between adjacent structures is modeled only probabilistically. In order to combine tags to complete parse trees an additional processing step has to be carried out after the tagging itself.

The idea to obtain a structural description of natural language utterances by applying a sequence of transformations which successively modifies an intermediate representation has first been pursued within the framework of parsing as tree-to-tree transduction [BGQA82], although no explicit notion of scoring and quality improvement was involved at that time. The dynamics of the transformation process was fully under the control of the grammar writer, taking into account the precedence ordering implicit in a sequence of rules and some additional means to influence the degree of non-determinism. Transformation-based approaches have later been applied to the problem of syntactic tagging [Bri95]. However, focus was on inducing an appropriate set of transformation rules from the information contained in an annotated corpus.

Possibilities to model grammar by means of contradicting principles are investigated currently in the framework of optimality theory [PS91]. The grammatical principles postulated there are ranked rather than weighted, with higher ranked regularities completely overriding the influence of the lower ones. First applications have been identified in phonology and syntax.

5 Conclusion

A novel approach to parsing as constraint-based structural disambiguation has been presented. By combining techniques for robust parsing (graded constraints, multi-level-disambiguation and partial parsing) with the idea of a transformation-based problem solving mechanism, a parser can be created that shows the typical temporal behavior of an interruptible anytime algorithm.

Further investigations will focus on

1. transferring these procedural characteristics to the case of incremental parsing, thus addressing particularly the problem of processing time for long utterances,
2. a more thorough investigation into time-adaptive behaviour, which should be able to speed up the convergence towards the optimum solution under temporal pressure, at the risk of missing it completely, and
3. the combination of different solution techniques to improve the termination behaviour.

The resulting parsing method mimics human language processing in that it is time-adaptive and robust and therefore lends itself to the implementation of human-machine dialogue systems.

References

[Amt99] Jan W. Amtrup. *Incremental Speech Translation*, volume 1735 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Heidelberg, New York, 1999.

[BJ99] Srinivas Bangalore and Aravind K. Joshi. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2):237–265, 1999.

[BD89] Mark Boddy and Thomas L. Dean. Solving Time-dependent Problems. in *Proceedings 11th Int. Joint Conference on Artificial Intelligence*, Detroit, 1989.

[BGQA82] Ch. Boitet, P. Guillaume, and M. Quezel-Ambrunaz. Implementation and conversational environment of ariane-78. In *Proceedings 9th International Conference on Computational Linguistics, Coling '82*, pages 19–28, Prague, CSSR, 1982.

[Bri95] Eric Brill. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565, 1995.

[GKWS96] Günther Görz, Marcus Kesseler, Hans Weber, and Jörg Spilker. Research on architectures for integrated speech/language systems in verbmobil. In *Proceedings 16th*

International Conference on Computational Linguistics, Coling '96, pages 484–489, Kopenhagen, Denmark, 1996.

[HH94] Mary P. Harper and Randall A. Helzerman. Managing multiple knowledge sources in constraint-based parsing of spoken language. Technical Report EE-94-16, School of Electrical Engineering, Purdue University, West Lafayette, 1994.

[HHZ+95] Mary P. Harper, Randall A. Helzermann, C. B. Zoltowski, B. L. Yeo, Y. Chan, T. Steward, and B. L. Pellom. Implementation issues in the development of the parsec parser. *Software - Practice and Experience*, 25(8):831–862, 1995.

[KVHA95] Fred Karlsson, Atro Voutilainen, Juha Heikkilä, and Arto Anttila, editors. *Constraint Grammar – A Language-Independent System for Parsing Unrestricted Text.* Mouton de Gruyter, Berlin, New York, 1995.

[Mar90] H. Maruyama. Structural disambiguation with constraint propagation. In *Proceedings 28th Annual Meeting of the ACL*, pages 31–38, 1990.

[Men94] Wolfgang Menzel. Parsing of spoken language under time constraints. In T. Cohn, editor, *Proceedings 11th European Conference on Artificial Intelligence*, pages 560–564, Amsterdam, 1994.

[MS98a] Wolfgang Menzel and Ingo Schröder, Constraint-Based Diagnosis for Intelligent Language Tutoring Systems. In *Proceedings IT & KNOWS, XV. IFIP World Computer Congress*, 484–497, Wien and Budapest, 1998.

[MS98b] Wolfgang Menzel and Ingo Schröder. Decision procedures for dependency parsing using graded constraints. In Sylvain Kahane and Alain Polguère, editors, *Proc. of the Joint Conference COLING/ACL Workshop: Processing of Dependency-based Grammars*, Montréal, Canada, 1998.

[Minton92] Minton, Steven, Johnston, Mark D., Philips, Andrew B. and Laird, Philip. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. In *Artificial Intelligence* 58, 161–205, 1992.

[PS91] Alan Prince and Paul Smolensky. Linguistics 247: Notes on connectionism and harmony theory in linguistics. In Technical Report CU-CS-533-91, Department of Computer Science, University of Colorado, Boulder, Colorado, 1991.

[SMFS] Ingo Schröder, Wolfgang Menzel, Kilian Foth and Michael Schulz. Dependency modeling with restricted constraints. Submitted to *Traitement automatique de langage*, Special Issue on Dependency Grammar.

SOUP: A PARSER FOR REAL-WORLD SPONTANEOUS SPEECH

Marsal Gavaldà
Interactive Systems, Inc.
1900 Murray Ave. Suite 203
Pittsburgh, PA 15217, U.S.A.
marsal@interactivesys.com

Abstract

This paper describes the key features of SOUP, a stochastic, chart-based, top-down parser, especially engineered for real-time analysis of spoken language with very large, multi-domain semantic grammars. SOUP achieves *flexibility* by encoding context-free grammars, specified for example in the Java Speech Grammar Format, as probabilistic recursive transition networks, and *robustness* by allowing skipping of input words at any position and producing ranked interpretations that may consist of multiple parse trees. Moreover, SOUP is very efficient, which allows for practically instantaneous backend response.

1 Introduction

Parsing can be defined as the assignment of structure to an utterance according to a grammar, i.e., the mapping of a sequence of words (utterance) into a parse tree (structured representation). Because of the ambiguity of natural language, the same utterance can sometimes be mapped into more than one parse tree; statistical parsing attempts to resolve ambiguities by preferring most likely parses. Also, spontaneous speech is intrinsically different from written text (see for example [Lavie 1996]), therefore when attempting to analyze spoken language, one must take a different parsing approach. For instance one must allow for an utterance to be parsed as a *sequence* of parse trees (which cover non-overlapping segments of the input utterance), rather than expect a single tree to cover the entire utterance and fail otherwise.

The SOUP parser herein described, inspired by Ward's PHOENIX parser [Ward 1990], incorporates a variety of techniques in order to achieve both *flexibility* and *robustness* in the analysis of spoken speech: flexibility is given by the lightweight formalism it supports, which allows for rapid grammar development, dynamic modification of the grammar at run-time, and fast parsing speed; robustness is achieved by its ability to find multiple-tree interpretations and to skip words at any point, thereby recovering in a graceful manner not only from false starts, hesitations, and other speech disfluencies but also from insertions unforeseen by the grammar. SOUP is currently the main parsing engine of the JANUS speech-to-speech translation system [Levin et al. 2000, Woszczyna et al. 1998].

Section 2 briefly describes the grammar representation, section 3 sketches the parsing process, section 4 presents some performance results, section 5 emphasizes the key features of SOUP, and section 6 concludes this paper.

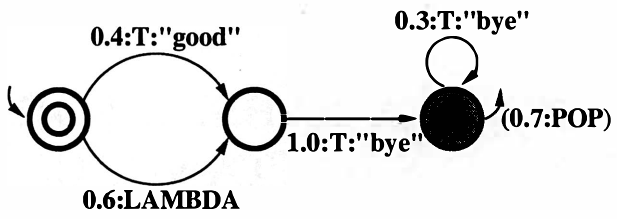


Figure 1: Representation of right-hand side *good +bye as a probabilistic recursive transition network (PRTN). A PRTN has a unique initial node (double circle) and possibly many final nodes (painted gray). A pop arc leaving each final node is implicit.

2 Grammar Representation

The grammar formalism supported by SOUP is purely context-free. Each nonterminal (whose value is simply a label), has a set of alternative rewrite rules, which consist of possibly optional, possibly repeatable terminals and nonterminals. We have found over the years [Mayfield et al. 1995, Woszczyna et al. 1998] that, at least for task-oriented semantic grammars used in speech translation systems, the advantages in parsing speed and ease of grammar construction of such a formalism outweight the lack of the more expressive power offered by richer formalisms (cf., for example, the [Verbmobil Semantic Specification 1994]).

SOUP represents a context-free grammar (CFG) as a set of probabilistic recursive transition networks (PRTNs), where the nodes are marked as initial, regular or final, and the directed arcs are annotated with (*i*) an arc type (namely, specific-terminal (which matches and consumes a particular word), any-terminal (which matches and consumes any out-of-vocabulary word or any word present in a given list), nonterminal (which recursively matches a subnet and, in the parsing process, spawns a subsearch episode) or lambda (the empty transition, which can always occur)), (*ii*) an ID to specify which terminal or nonterminal the arc has to match (if arc type is specific-terminal or nonterminal), and (*iii*) a probability (so that all outgoing arcs from the same node sum to unity). For example, the right-hand side *good +bye (where * indicates optionality and + repeatability and therefore matches *good bye*, *bye*, *bye bye* (and also *good bye bye*, etc)) is represented as the PRTN in Figure 1.

SOUP also directly accepts grammars written in the Java Speech Grammar Format (see section 5.5).

Grammar arc probabilities are initialized to the uniform distribution but can be perturbed by a training corpus of desired (but achievable) parses. Given the direct correspondence between parse trees and grammar arc paths, training the PRTNs is very fast (see section 4).

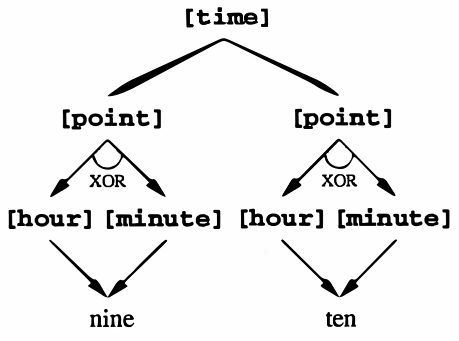
There are two main usages of this stochastic framework of probabilities at the grammar arc level: one is to incorporate the probabilities into the function that scores partial parse lattices, so that more likely ones are preferred; the other is to generate synthetic data, from which, for instance, a language model can be computed.

The PRTNs are constructed dynamically as the grammar file is read; this allows for eventual on-line modifications of the grammar (see section 5.4). Also, strict grammar source file consistency is enforced, e.g., all referenced nonterminals must be defined, warnings for nonterminal redefinitions are issued, and a variety of grammar statistics are provided. Multiple grammar files representing different semantic domains as well as a library of shared rules are supported, as described in [Woszczyna et al. 1998].

The lexicon is also generated as the grammar file is being read, for it is simply a hash table of grammar terminals.

```
s[time]
          ( +[point] )
[point]
          ( [hour] )
          ( [minute] )
[hour]
          ...
          ( nine )
          ( ten )
          ...
[minute]
          ...
          ( nine )
          ( ten )
          ...
```

(a)



(b)

Figure 2: (a) Grammar fragment to illustrate ambiguity packing. The s indicates that nonterminal `[time]` is a starting symbol of the grammar. (b) Parse lattice for input *nine ten* according to the grammar fragment. It can give rise to four different parse trees, but some will be more likely than others.

3 Sketch of the Parsing Algorithm

Parsing is a particular case of search. In SOUP, parsing proceeds in the following steps:

1. *Construction of the input vector*: Given an utterance to be parsed, it is converted into a vector of terminal IDs. Special terminals <s> and </s> are added at the beginning and end of an utterance, respectively, so that certain rules only match at those positions. Also, user-defined global search-and-replace string pairs are applied, e.g., to expand contractions (as in *I'd like* $\rightarrow$ *I would like*) or to remove punctuation marks. Other settings allow to determine whether out-of-vocabulary words should be removed, or whether the input utterances are case-sensitive.

2. *Population of the chart*: The first search populates the chart (a two-dimensional table indexed by input-word position and nonterminal ID) with parse lattices. (A parse lattice is a compact representation of a set of parse trees (similar to Tomita's shared-packed forest [Tomita 1987]); see Figure 2 for an example).

   This beam search involves top-down, recursive matching of PRTNs against the input vector. All top-level nonterminals starting at all input vector positions are attempted. The advantage of the chart is that it stores, in an efficient way, all subparse lattices found so far, so that subsequent subsearch episodes can reuse existing subparse lattices.

   To increase the efficiency of the top-down search, the set of allowed terminals with which a nonterminal can start is precomputed (i.e., the FIRST set), so that many attempts to match a particular nonterminal at a particular input vector position can be preëmpted by the lack of the corresponding terminal in the FIRST set. This bottom-up filtering technique typically results in a threefold speedup.

   The beam serves to restrict the number of possible subparse lattices under a certain nonterminal and starting at a certain input position, e.g., by only keeping those subparse lattices whose score is

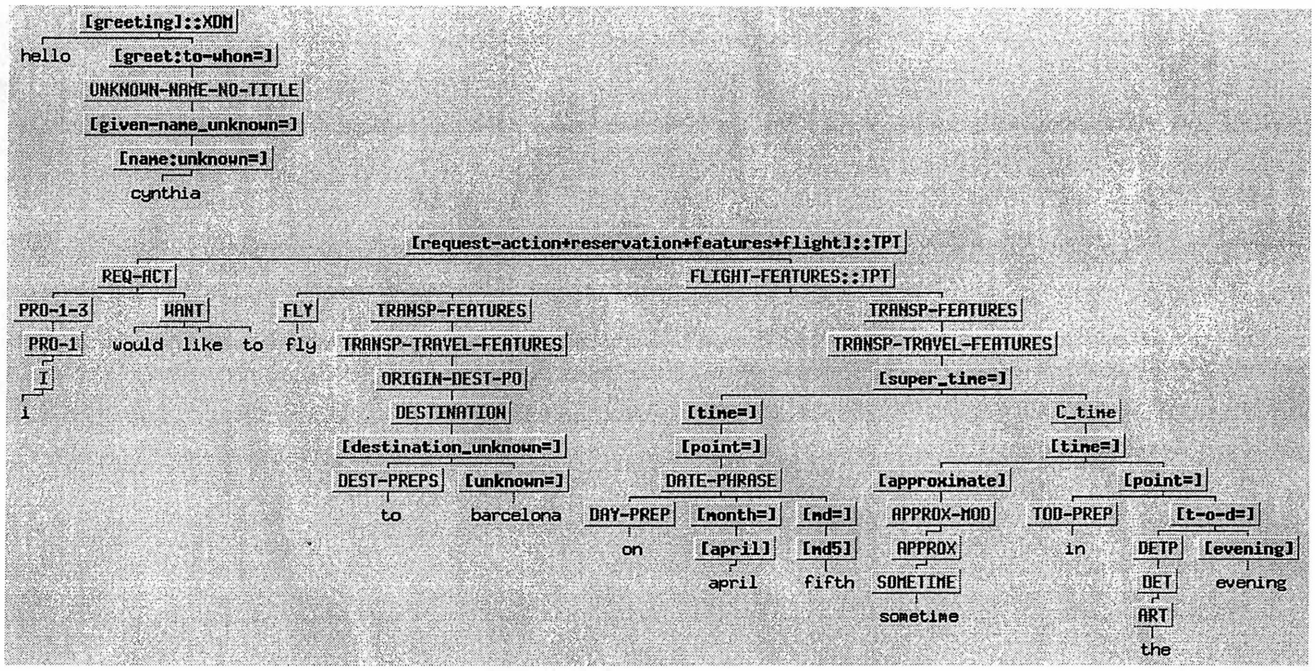


Figure 3: Parse of *Hello Cynthia I'd like to fly to Barcelona on April fifth sometime in the evening.* Uppercased nonterminals such as PRO-1-3 denote auxiliary nonterminals and are typically removed from the parse trees before backend processing. Note the ability to combine rules from different task domains (XDM for cross-domain, TPT for transportation) and to parse out-of-vocabulary words (*Cynthia*, *Barcelona*).

at least 30% of the best score. The score function is such that (*i*) coverage (number of words parsed) and (*ii*) sum of arc probabilities are maximized, whereas (*iii*) parse lattice complexity (approximated by number of nonterminals) and (*iv*) usages of the wildcard (approximated by maximal number of any-terminal arcs along the parse lattice) are minimized. Also, pruning of structurally-equal parse lattices is performed, thereby eliminating the redundancy that arises from several right-hand sides matching the same input vector span under the same nonterminal.

3. *Finding the best interpretations*: Once the chart is populated, a second beam search finds the best N interpretations, i.e., the best N sequences of top-level, non-overlapping parse lattices that cover the input vector. Scoring of interpretations adds, to the above scoring function, a fifth factor, namely the minimization of parse fragmentation (number of parse trees per utterance). This search problem can be divided into subproblems (divide and conquer strategy) since both unparsed words and words parsed by a single parse lattice offer a natural boundary to the general problem. A beam search is conducted for each subproblem. In this case, the beam limits the number of active sequences of top-level, non-overlapping parse lattices that form a partial interpretation. Since the single best interpretation is simply the concatenation of the best sequence of each subproblem, even when asked to compute the top N interpretations ($N > 1$), the best interpretation is always computed separately and output immediately so that backend processing can begin without delay. The final result is a ranked list of N interpretations, where the parse lattices have been expanded into parse trees.

Figure 3 shows a sample interpretation in a travel domain.

	Scheduling Grammar	Scheduling + Travel Grammar
Nonterminals	600 (21 top-level)	6,963 (480 top-level)
Terminals	831	9,640
Rules	2,880	25,746
Nodes	9,853	91,264
Arcs	9,866	97,807
Average cardinality of FIRST sets	44.48 terminals	240.31 terminals
Grammar creation time	143 ms	3,731 ms
Training time for 1000 example parse trees	452 ms	765 ms
Memory	2 MB	14 MB
Average parse time	10.09 ms/utt	228.99 ms/utt
Maximal parse time	53 ms	1070 ms
Average coverage	85.52%	88.64%
Average fragmentation	1.53 trees/utt	1.97 trees/utt

Table 1: Grammar measurements and performance results of parsing 606 naturally-occurring scheduling domain utterances (average length of 9.08 words) with scheduling grammar and scheduling plus travel grammar on a 266-MHz Pentium II running Linux.

4 Performance

SOUP has been coded in C++ and Java and compiled for a variety of platforms including Windows (95, 98, NT) and Unix (HP-UX, OSF/1, Solaris, Linux). The upper portion of Table 1 lists some parameters that characterize the complexity of two grammars, one for a scheduling domain and the other for a scheduling plus travel domain; the lower portion lists performance results of parsing a subset of transcriptions from the English Spontaneous Speech Scheduling corpus (briefly described in [Waibel et al. 1996]).

Parsing time increases substantially from a 600-nonterminal, 2,880-rule grammar to a 6,963-nonterminal, 25,746-rule grammar but it is still well under real-time. Also, as depicted in Figure 4, although worst-case complexity for chart parsing is cubic on the number of words, SOUP's parse time appears to increase only linearly. Such behavior, similar to the findings reported in [Slocum 1981], is due, in part, to SOUP's ability to segment the input utterance in parsable chunks (i.e., finding multiple-tree interpretations) during the search process.

Therefore, even though comparisons of parsers using different grammar formalisms are not well-defined, SOUP appears to be faster than other "fast parsers" described in the literature (cf., for example, [Rayner and Carter 1996] or [Kiefer and Krieger 1998]).

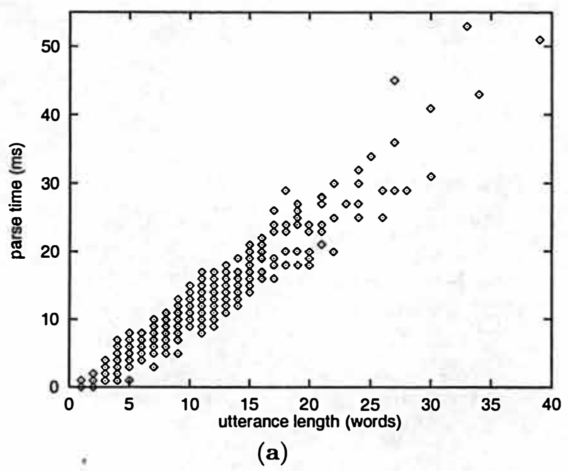
5 Key Features

The following are some of the most interesting features of SOUP.

5.1 Skipping

Given the nature of spoken speech it is not realistic to assume that the given grammar is complete (in the sense of covering all possible surface forms). In fact it turns out that a substantial portion of parse errors comes from unexpected insertions, e.g., adverbs that can appear almost anywhere.

SOUP is able to skip words both between top-level nonterminals (inter-concept skipping) and inside any nonterminal (intra-concept skipping). Inter-concept skipping is achieved by the second search step



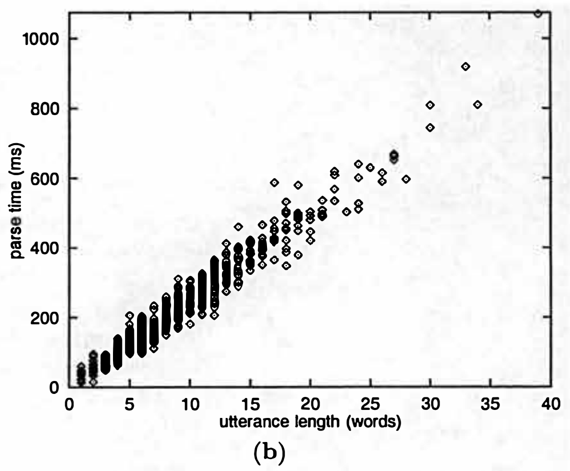


Figure 4: Utterance length vs. parse time for (**a**) scheduling grammar and (**b**) scheduling plus travel grammar. Same test and machine as in Table 1. Parse time appears to increase only linearly with regard to utterance length.

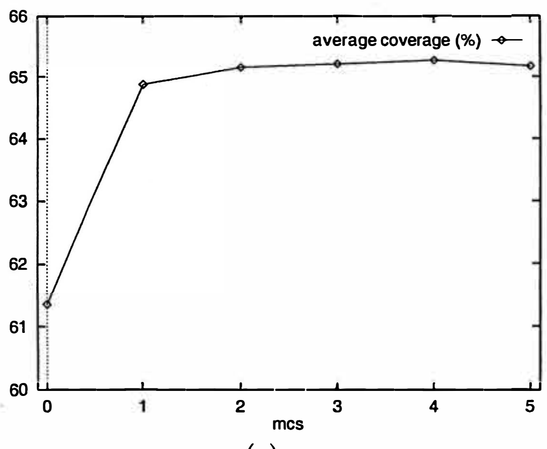
described in section 3 (the search that finds the best interpretation as a sequence of non-overlapping parse lattices), since an interpretation may naturally contain gaps between top-level parse lattices. Intra-concept skipping, on the other hand, occurs during the first search step, by allowing, with a penalty, insertions of input words at any point in the matching of a net. The resulting exponential growth of parse lattices is contained by the beam search. A word-dependent penalty (e.g. one based on word saliency for the task at hand) can be provided but the experiments reported here use a uniform penalty together with a list of non-skippable words (typically containing, for example, the highly informative adverb *not*). The parameter mcs regulates the maximal number of contiguous words that can be skipped within a nonterminal. Figure 5 plots coverage and parse times for different values of mcs. These results are encouraging as they demonstrate that coverage lost by skipping words is offset (up to mcs = 4) by the ability to match longer sequences of words.

5.2 Character-level Parsing

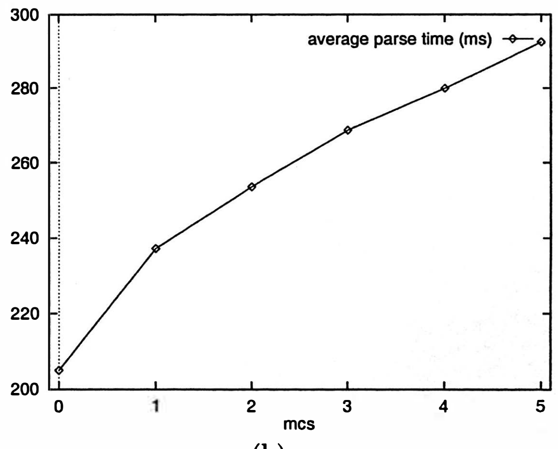
To facilitate the development of grammars for languages with a rich morphology, SOUP allows for nonterminals that operate at the character-level (see Figure 6 for an example). Character-level parsing is achieved using the same functions that parse at the word-level. In fact it is during word-level parsing that character-level parses are spawned by exploding the current word into characters and recursively calling the parse functions. The only difference is that, in a character-level parse, the desired root nonterminal is already known and no skipping or multiple-tree interpretations are allowed.

5.3 Multiple-tree Interpretations

SOUP is designed to support a modular grammar architecture and, as we have seen, performs segmentation of the input utterance into parse trees as part of the parsing process itself. Different interpretations of the same utterance may have different segmentations and the most likely one will be ranked first. Knowledge of the grammar module (which usually corresponds to a task domain) that a nonterminal is from is used in the computation (see [Woszczyna et al. 1998] for details on the statistical model employed).



(a)



(b)

Figure 5: (a) Average coverage and (b) parse times for different values of mcs (maximal number of contiguous words that can be skipped within a nonterminal). Same test set and machine as in Table 1 but with travel grammar only.

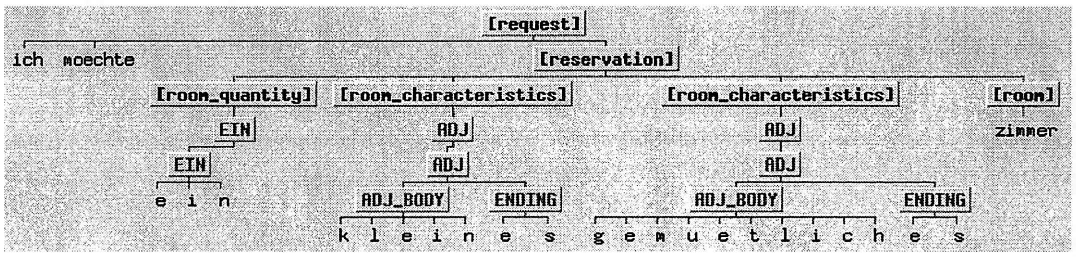


Figure 6: Parse of German *Ich möchte gern ein kleines gemütliches Zimmer* to exemplify character-level nonterminals and skipping. Repeated nonterminals (as in EIN with daughter EIN) indicate the joint between word-level and character-level parse trees. Also note the intra-concept skipping of *gern* (an adverb with practically zero information content).

5.4 Dynamic Modifications

Encoding the grammar as a set of PRTNs gives SOUP the flexibility to activate/deactivate nonterminals and right-hand sides at run-time. For example, grammar nonterminals can be marked as belonging only to a specific speaker side (say, agent vs. client); then, at run-time and for each utterance, nonterminals not belonging to the current speaker are deactivated. Also, in the case of a multi-domain grammar, one could have a topic-detector that deactivates all non-terminals not belonging to the current topic, or at least lowers their probability.

More generally, nonterminals and right-hand sides can be created, modified and destroyed at run-time, which allows for the kind of interactive grammar learning reported in [Gavaldà and Waibel 1998].

5.5 Parsing JSGF Grammars

SOUP has been extended to natively support grammars written according to the specifications of the Java Speech Grammar Format [JSGF 1998]. The JSGF is part of the Java Speech Application Programming Interface [JSAPI 1998] and is likely to become a standard formalism for specifying semantic grammars, at least in the industrial environment.

```
#JSGF V1.0 ISO8859-1 en;
grammar Toy;
public <get> =  <polite>* (get | obtain | request) <obj>+;
    <polite> =  please;
       <obj> =  apple | pear | orange;
```

Figure 7: Toy JSGF grammar used in Figures 8 and 9. In this case * indicates the Kleene star (i.e., optionality and repeatability), + repeatability, and | separates alternatives.

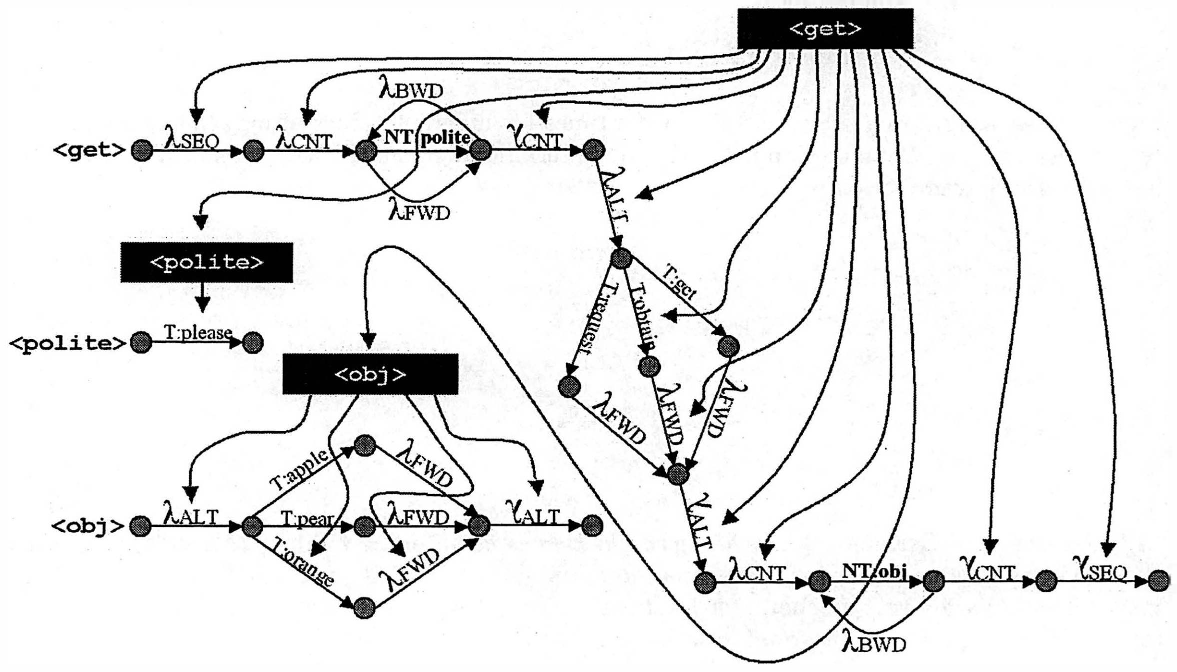


Figure 8: PRTNs for the grammar in Figure 7 and schematic parse of *Please obtain orange.* Upside-down lambdas mark end of JSGF `Rule` scope.

SOUP is able to represent a `RuleGrammar` as defined by JSGF with the same underlying PRTNs. This is accomplished by the usage of lambda arcs to encode the JSGF `Rule` source, so that, out of a parse tree, the corresponding `RuleParse` (the result of a parse as specified by the JSAPI), can be constructed. In more detail, for each JSGF `RuleSequence`, `RuleAlternatives`, `RuleCount` and `RuleTag`, a corresponding lambda-SEQ, lambda-ALT, lambda-CNT or lambda-TAG arc is built in the PRTNs, as well as a closing lambda arc (pictured upside-down) to indicate the end of scope of the current JSGF `Rule`.

Figure 7 shows a toy grammar in the JSGF formalism. Figure 8 depicts the corresponding PRTNs as well as a schematic sample parse tree. Figure 9 shows the resulting `RuleParse` object constructed from such parse.

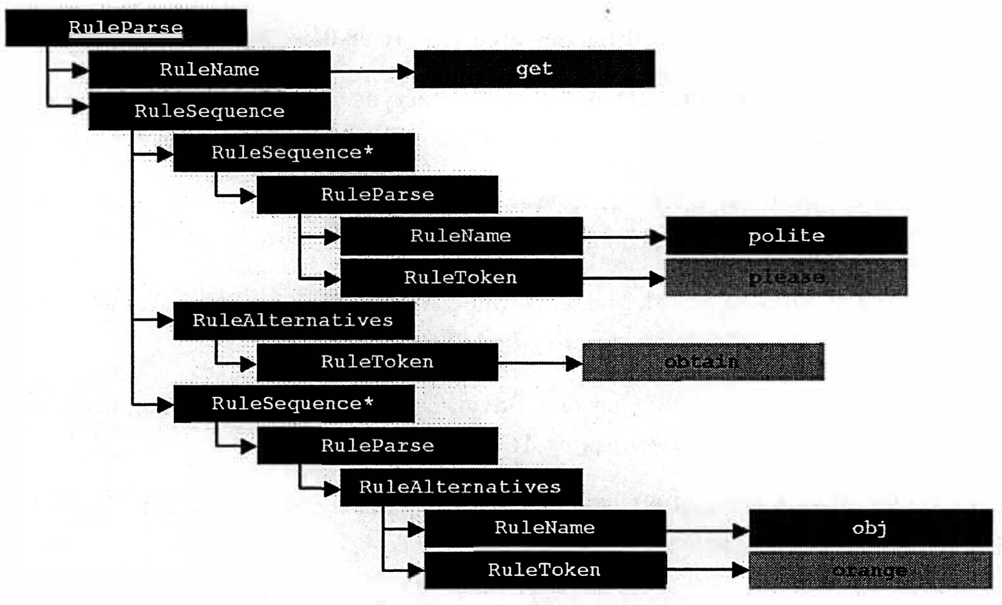


Figure 9: Resulting JSGF `RuleParse` constructed from the parse in Figure 8. Note that `RuleCounts` become `RuleSequences` as specified by the JSAPI (marked with $*$).

6 Conclusion

We have presented Soup, a parser designed to analyze spoken language under real-world conditions, in which analysis grammars are very large, input utterances contain disfluencies and never entirely match the expectations of the grammar, and yet backend processing must begin with minimal delay. Given its robustness to ill-formed utterances, general efficiency and support for emerging industry standards such as the JSAPI, Soup has the potential to become widely used.

Acknowledgements

Laura Mayfield Tomokiyo wrote the Scheduling grammar, Donna Gates and Dorcas Wallace wrote the Travel grammar; Detlef Koll programmed the methods that read in a JSGF `RuleGrammar`. Klaus Zechner, Matthias Denecke and three anonymous reviewers gave helpful comments to earlier versions of this paper.

References

[Gavaldà and Waibel 1998] Marsal Gavaldà and Alex Waibel. 1998. Growing Semantic Grammars. In *Proceedings of COLING/ACL-1998.*

[JSAPI 1998] Java^{TM} Speech API, version 1.0. 1998. `http://java.sun.com/products/java-media/speech/`

[JSGF 1998] Java^{TM} Speech Grammar Format, version 1.0. 1998. `http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/`

[Kiefer and Krieger 1998] Bernd Kiefer and Hans-Ulrich Krieger. 1998. A Bag of Useful Techniques for Efficient and Robust Parsing. DFKI Research Report 98-04.

[Levin et al. 2000] Lori Levin, Alon Lavie, Monika Woszczyna, Donna Gates, Marsal Gavaldà, Detlef Koll and Alex Waibel. 2000. The JANUS-III Translation System. To appear in *Machine Translation.*

[Lavie 1996] Alon Lavie. 1996. *GLR*: A Robust Grammar-Focused Parser for Spontaneously Spoken Language.* Doctoral dissertation. School of Computer Science, Carnegie Mellon University.

[Mayfield et al. 1995] Laura Mayfield, Marsal Gavaldà, Wayne Ward and Alex Waibel. 1995. Concept-Based Speech Translation. In *Proceedings of ICASSP-1995.*

[Rayner and Carter 1996] Manny Rayner and David Carter. 1996. Fast Parsing using Pruning and Grammar Specialization. In *Proceedings of ACL-1996.*

[Slocum 1981] Jonathan Slocum. 1981. A Practical Comparison of Parsing Strategies. In *Proceedings of ACL-1981.*

[Tomita 1987] Masaru Tomita. 1987. An Efficient Augmented-Context-Free Parsing Algorithm. In *Computational Linguistics*, Volume 13, Number 1-2, pages 31–46.

[Verbmobil Semantic Specification 1994] Universität des Saarlandes. 1994. The Verbmobil Semantic Specification. Verbmobil Report 1994-6.

[Waibel et al. 1996] Alex Waibel, Michael Finke, Donna Gates, Marsal Gavaldà, Thomas Kemp, Alon Lavie, Lori Levin, Martin Maier, Laura Mayfield, Arthur McNair, Ivica Rogina, Kaori Shima, Tilo Sloboda, Monika Woszczyna, Torsten Zeppenfeld and Puming Zhan. 1996. JANUS-II: Translation of Spontaneous Conversational Speech. In *Proceedings of ICASSP-1996.*

[Ward 1990] Wayne Ward. 1990. The CMU Air Travel Information Service: Understanding spontaneous speech. In *Proceedings of the DARPA Speech and Language Workshop.*

[Woszczyna et al. 1998] Monika Woszczyna, Matthew Broadhead, Donna Gates, Marsal Gavaldà, Alon Lavie, Lori Levin and Alex Waibel. 1998. A Modular Approach to Spoken Language Translation for Large Domain. In *Proceedings of AMTA-1998.*

A RECOGNIZER FOR MINIMALIST GRAMMARS

Henk Harkema
Department of Linguistics
University of California
Los Angeles, CA 90025, USA
harkema@humnet.ucla.edu

Abstract

Minimalist Grammars are a rigorous formalization of the sort of grammars proposed in the linguistic framework of Chomsky's Minimalist Program. One notable property of Minimalist Grammars is that they allow constituents to move during the derivation of a sentence, thus creating discontinuous constituents. In this paper we will present a bottom-up parsing method for Minimalist Grammars, prove its correctness, and discuss its complexity.

1 Introduction

It seems to be a general feature of natural language that the elements of a sentence are pronounced in one position, while at the same time serving a function in another part of the structure of the sentence. Linguistic theories in the transformational tradition have tried to capture this fact by proposing analyses that involve movement of constituents. Stabler ([8]) presents a formalism for defining minimalist grammars that allow for movement of constituents. This formalism is based on Chomsky's Minimalist Program ([2]).

Michaelis ([5]) provides an argument showing that minimalist grammars as defined in ([8]) are weakly equivalent to multiple context-free grammars as described in Seki et al. ([6]).[1] Multiple context-free grammars are non-concatenative in the sense that a non-terminal symbol in this grammar can dominate a sequence of strings of terminal symbols, rather than just one string, as in the case of ordinary context-free grammars. Each of the strings dominated by a non-terminal symbol in a multiple context-free grammar will be a substring of a sentence whose derivation includes this non-terminal, but in the sentence these strings are not necessarily adjacent. The main insight contained in [5] is that minimalist grammars are non-concatenative in a similar way. In minimalist grammars, non-concatenativity arises as the result of movement. Thus, in a minimalist grammar a constituent can dominate non-adjacent substrings of a sentence. Seki et al. ([6]) also describe an algorithm for recognizing multiple context-free grammars. Stabler ([10]) sketches how this algorithm may be extended to minimalist grammars.

This paper contains a formal specification of a recognizer for minimalist grammars. Furthermore, it is shown that the recognizer is sound and complete, and that its time complexity is polynomial in the length of the input string. Besides this introduction, this paper has six sections. Section

[1]Hence, minimalist grammars are also weakly equivalent to linear context-free rewriting systems ([11]), multi-component tree-adjoining grammars ([12]), and simple positive range concatenation grammar ([1]), as these formalisms are equivalent to multiple context-free grammars.

2 introduces the minimalist grammars as defined in [8]. Section 3 contains the specification of the recognizer. Sections 4 and 5 present the proofs of soundness and completeness. Section 6 describes some complexity results. The paper concludes with some directions for future work.

2 Minimalist Grammars

Minimalist grammars manipulate minimalist trees. Minimalist trees are finite, binary ordered trees whose leaves are labeled with sequences of syntactic and non-syntactic features. New trees are built by either merging two trees into one, or by moving a subtree in a tree. The application of merge and move operations is driven by the syntactic features labeling the leaves of the trees. The language defined by a minimalist grammar consists of the yields of a particular subset of the trees generated by the grammar.

Formally, following [8], a minimalist grammar G is defined to be quadruple (V, Cat, Lex, F), where V is a finite set of non-syntactic features, Cat is a finite set of syntactic features, Lex is a finite set of lexical expressions built from V and Cat, and F is a finite set of structure building functions.

The set of non-syntactic features V is made up of a set of phonetic features P and a set of semantic features I: V=P ∪ I. The set Cat comprises four kinds of syntactic features: Cat=*base* ∪ *select* ∪ *licensors* ∪ *licensees*. The elements of *base* represent basic syntactic categories. The set *base* minimally contains the distinguished category feature c. For each category feature x ∈ *base*, there will be a selection feature =x ∈ *select*, although *select* does not necessarily contain a feature =c. The features in *base* and *select* play a role in merge operations. The features in *licensors* and *licensees* regulate movement operations. For each feature -y ∈ *licensees*, there will be a feature +y ∈ *licensors*.

A minimalist tree τ is given by a non-empty set of nodes N_τ, a function $Label_\tau$, and three relations on N_τ: dominance, precedence and projection. Immediate dominance and immediate precedence and their reflexive and transitive closures are defined as for trees of the usual kind. For any two sister nodes x, y in a minimalist tree τ, either x projects over y, or y projects over x, notated x < y and y < x, respectively, or x > y and y > x. The function $Label_\tau$ assigns to each leaf of τ a string from (V ∪ Cat)*, in particular, a string from *select* licensors* select* base licensees**P*I*. The other nodes of τ are not labeled. The elements of Lex are are assumed to be trees consisting of one node.

This paragraph will introduce some arboreal notions that will be used in the remainder of the paper. For nodes x, y in a tree τ, x is the head of y if and only if y is a leaf and x=y, or there is a node z in τ such that y immediately dominates z, z projects over its sister and x is the head of z. The head of a tree τ is the head of the root of τ. A node y in a tree τ is a maximal projection of a head x in τ if and only if x is the head of y and y's sister projects over y. A tree τ is maximal if and only if its root is the maximal projection of some head. A tree τ is complex if and only if it has more than one node, otherwise, τ is simple. A tree τ is said to have feature f ∈ V ∪ Cat if the first element of the sequence that labels the head of τ is f. For τ and υ minimalist trees, $[_<\tau, \upsilon]$ denotes a tree with immediate subtrees τ and υ where the root of τ projects over and precedes the root of υ, and $[_>\tau, \upsilon]$ denotes a tree with immediate subtrees τ and υ where the root of τ precedes the root of υ and the root of υ projects over the root of τ. A tree τ is complete if its head does not contain any syntactic features except for the distinguished category feature c, and no node in τ other than the head has any syntactic features. The yield $Y(\tau)$ of a tree τ is the concatenation of the phonetic features in the labels of the leaves of τ, ordered by precedence.

There are two structure building functions: F={*merge*, *move*}. A pair of trees τ, υ is in the domain of *merge* if τ has feature =x and υ has feature x for some x $\in$ *base*. Then,

merge$(\tau, \upsilon)=[_<\tau', \upsilon']$ if τ is simple, and
merge$(\tau, \upsilon)=[_>\upsilon', \tau']$ if τ is complex,

where τ' is like τ except that =x is deleted, and υ' is like υ except that x is deleted. A tree τ is in the domain of *move* if τ has feature +y $\in$ *licensors* and τ has exactly one maximal subtree τ_0 that has feature -y $\in$ *licensees*.[2] Then,

move$(\tau)=[_>\tau_0', \tau']$,

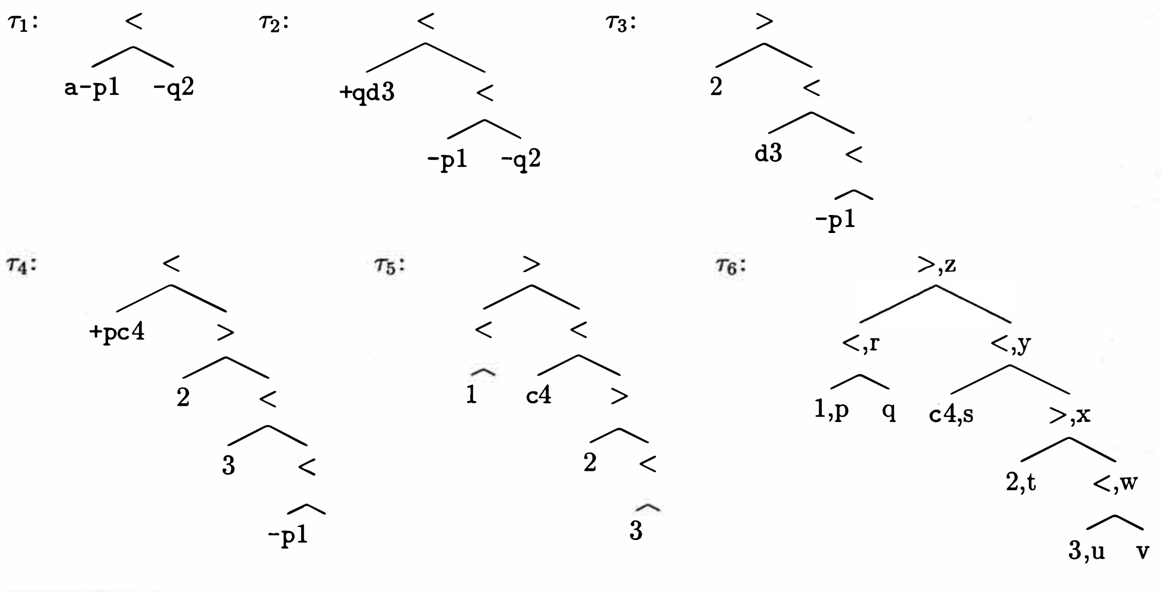
where τ_0' is like τ_0 except that -y is deleted, and τ' is like τ except that +y is deleted and subtree τ_0 is replaced by a single node without features.[3]

Let G=(V, Cat, Lex, F) be a minimalist grammar. Then $\mathrm{CL}(G)=\bigcup_{k\in\mathbb{N}} \mathrm{CL}^k(G)$ is the closure of the lexicon under the structure building functions, where $\mathrm{CL}^k(G)$, $k \in \mathbb{N}$, are inductively defined by:

1. $\mathrm{CL}^0(G)=\mathrm{Lex}$
2. $\mathrm{CL}^{k+1}(G)=\mathrm{CL}^k(G) \cup \{merge(\tau, \upsilon) | (\tau, \upsilon) \in \mathrm{Dom}(merge) \cap \mathrm{CL}^k(G)\times\mathrm{CL}^k(G)\} \cup \{move(\tau) | \tau \in \mathrm{Dom}(move) \cap \mathrm{CL}^k(G)\}$,

where Dom(*merge*) and Dom(*move*) denote the domains of the functions *merge* and *move*. The language derivable by G consists of the yields of the complete trees in the closure of the lexicon under the structure building functions: $L(G)=\{Y(\tau) | \tau \in \mathrm{CL}(G)$ and τ is complete$\}$.

Example 1: Consider the minimalist grammar G defined by the following sets: I=$\emptyset$, P={1, 2, 3, 4}, *base*={a, b, c, d}, *select*={=a, =b, =d}, *licensors*={+p, +q}, *licensees*={-p, -q}, Lex={=ba-p1, b-q2, =a+qd3, =d+pc4}. The sentence '1423' is derived as follows: *merge*(=ba-p1, b-q2)=$[_<$a-p1, -q2], or tree τ_1 below; *merge*(=a+qd3, τ_1)= τ_2; *move*(τ_2)= τ_3; *merge*(=d+pc4, τ_3)= τ_4; *move*(τ_4)= τ_5.



[2]Since +y≠-y, τ and τ_0 have different head labels. Hence, τ_0 is properly included in τ.
[3]The structure building functions in Stabler ([8]) also allow for head movement, which is not discussed in this paper. The recognizer described in this paper is easily adapted to deal with this kind of movement.

Tree τ_5 is a complete tree.[4] Tree τ_6 is the same as tree τ_5. The nodes in τ_6 are named for illustrational purposes only; these names have no significance in the grammar. In tree τ_6, node u is the head of nodes u, w and x. Node s is the head of nodes s, y and z. Node t is the head of node t and no other node. Nodes z, r, q, x, t and v are maximal projections; nodes s, p, q, u, t, v are their respective heads. Node s has feature c. No other node has a syntactic feature. τ_w=[$_<$ τ_u, τ_v], and τ_x=[$_>$ τ_t, τ_w], where τ_n denotes the subtree of τ_6 whose root is named n. □

3 Specification of the Recognizer

This section contains a formal definition of an agenda-driven, chart-based recognizer for minimalist languages. Taking a logical perspective on parsing as presented in Shieber et al. ([7]), the definition of the recognizer includes a specification of a grammatical deductive system and a specification of a deduction procedure. The formulae of the deductive system, which are commonly called items, express claims about grammatical properties of strings. Under a given interpretation, these claims are either true or false. For a given grammar and input string, there is a set of items that, without proof, are taken to represent true grammatical claims. These are the axioms of the deductive system. Goal items represent the claim that the input string is in the language defined by the grammar. Since our objective is to recognize a string, the truth of the goal items is of particular interest. The deductive system is completed with a set of inference rules, for deriving new items from old ones. The other component of the definition of the recognizer is the specification of a deduction procedure. This is a procedure for finding all items that are true for a given grammar and input string.

3.1 Deduction Procedure

The deduction procedure used in the recognizer presented in this paper is taken from Shieber et al. ([7]). It uses a chart holding unique items in order to avoid applying a rule of inference to items to which the rule of inference has already applied before. Furthermore, there is an agenda for temporarily keeping items whose consequences under the inference rules have not been generated yet. The procedure is defined as follows:

1. Initialize the chart to the empty set of items and the agenda to the axioms of the deduction system.
2. Repeat the following steps until the agenda is exhausted:
   a) Select an item from the agenda, called the trigger item, and remove it.
   b) Add the trigger item to the chart, if the item is not already in the chart.
   c) If the trigger item was added to the chart, generate all items that can be derived from the trigger item and any items in the chart by one application of a rule of inference,[5] and add these generated items to the agenda.
3. If a goal item is in the chart, the goal is proved, i.e., the string is recognized, otherwise it is not.

[4]Incidentally, note that the last movement of this derivation is an instance of remnant movement. Remnant movement is movement of a constituent from which material has already been extracted. In this particular case it creates a configuration in which the moved constituent with yield '2' no longer c-commands its trace. This example shows that remnant movement is easily modeled in minimalist grammars (see Stabler ([9]) for further discussion). Recently, remnant movement has gained some linguistic popularity, e.g. [4], [3].

[5]Note that successful applications of Move-1 and Move-2 as defined in section 3.2.2 do not involve any items from the chart.

Shieber et al. ([7]) prove that the deductive procedure is sound – it generates only items that are derivable from the axioms – and complete – it generates all the items that are derivable from the axioms.

3.2 Deductive System

Given an input string $w=w_1 \dots w_n$ and minimalist grammar G=(V, Cat, Lex, F), the items of the deductive system will be of the form $[\alpha_0, \alpha_1, \dots, \alpha_m]_t$, where $m \leq |licensees|$, $t \in \{s, c\}$. For $0 \leq i \leq m$, α_i is of the form $(x_i, y_i){:}\gamma_i$, where $0 \leq x_i \leq y_i \leq n$, $n=|w|$, and $\gamma_i \in Cat^*$.

The proper interpretation of the items requires the notion of narrow yield of a tree. The narrow yield $Y_n(\phi)$ of a minimalist tree ϕ is defined in the following way. If ϕ is a complex tree, then either $\phi=[_>\tau, \upsilon]$, or $\phi=[_<\tau, \upsilon]$. If $\phi=[_>\tau, \upsilon]$, then:

$Y_n(\phi)=Y_n(\tau)\cdot Y_n(\upsilon)$ if τ does not have a feature -f $\in$ *licensees*[6]
$Y_n(\phi)=Y_n(\upsilon)$ otherwise.

If $\phi=[_<\tau, \upsilon]$, then:

$Y_n(\phi)=Y_n(\tau)\cdot Y_n(\upsilon)$ if υ does not have a feature -f $\in$ *licensees*
$Y_n(\phi)=Y_n(\tau)$ otherwise.

If ϕ is not a complex tree, it must be a simple tree. In that case:

$Y_n(\phi)=Y(\phi)$

Informally, the narrow yield of a tree is that part of its yield that will not move out of the tree by some application of the function *move*. For example, the trees from example 1 have the following narrow yields: $Y_n(\tau_1)=1$; $Y_n(\tau_2)=3$; $Y_n(\tau_3)=23$; $Y_n(\tau_4)=423$; $Y_n(\tau_5)=Y(\tau_5)=1423$.

Now, an item $[(x_0, y_0){:}\gamma_0, (x_1, y_1){:}\gamma_1, \dots, (x_m, y_m){:}\gamma_m]_t$ is understood to assert the existence of a tree $\tau \in CL(G)$ which has the following properties:

1. If t=s, τ is a simple tree; if t=c, τ is a complex tree.
2. The head of τ is labeled by $\gamma_0\pi\iota$, $\pi\iota \in P^*I^*$.
3. For every $(x_i, y_i){:}\gamma_i$, $1 \leq i \leq m$, there is a leaf in τ labeled $\gamma_i\pi\iota$, $\pi\iota \in P^*I^*$.
4. Besides the nodes labeled by $\gamma_i\pi\iota$, $\pi\iota \in P^*I^*$, $0 \leq i \leq m$, there are no other nodes with syntactic features in τ.
5. The narrow yield of the subtree whose root is the maximal projection of the node labeled by $\gamma_i\pi\iota$, $\pi\iota \in P^*I^*$, is $w_{x_i+1} \dots w_{y_i}$, $0 \leq i \leq m$.

Axioms and Goals

The set of axioms of the deductive system is specified in the following way. For each lexical item in Lex with syntactic features $\gamma \in Cat^*$ and whose phonetic features cover $w_{i+1} \dots w_j$ of the input string, there will be an axiom $[(i, j){:}\gamma]_s$ in the deductive system.

There will be two goal items: $[(0, n){:}c]_s$ and $[(0, n){:}c]_c$, $c \in$ *base* being the distinguished category feature. These are appropriate goal items for a recognizer, since their truth under the interpretation provided above requires the existence of a complete tree $\tau \in CL(G)$, either simple or complex, with narrow yield $Y_n(\tau)=w_1 \dots w_n=w$. Since τ is complete, $Y_n(\tau)=Y(\tau)$. Therefore, $w \in L(G)=\{Y(\tau)|\tau \in CL(G)$ and τ is complete$\}$.

[6]Recall that 'to have a feature' is a formal notion, defined in section 2.

Rules of Inference

The deductive system has six rules of inference, grouped into Merge rules and Move rules:
Merge-1:

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}{=}\mathrm{x}\gamma]_s,\ [(\mathrm{q},\ \mathrm{v}){:}\mathrm{x},\ \alpha_1,\ \ldots,\ \alpha_k]_t}{[(\mathrm{p},\ \mathrm{v}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k]_c}$$

Merge-2: ($\delta \neq \emptyset$)

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}{=}\mathrm{x}\gamma]_s,\ [(\mathrm{v},\ \mathrm{w}){:}\mathrm{x}\delta,\ \alpha_1,\ \ldots,\ \alpha_k]_t}{[(\mathrm{p},\ \mathrm{q}){:}\gamma,\ (\mathrm{v},\ \mathrm{w}){:}\delta,\ \alpha_1,\ \ldots,\ \alpha_k]_c}$$

Merge-3:

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}{=}\mathrm{x}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k]_c,\ [(\mathrm{v},\ \mathrm{p}){:}\mathrm{x},\ \iota_1,\ \ldots,\ \iota_l]_t}{[(\mathrm{v},\ \mathrm{q}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k,\ \iota_1,\ \ldots,\ \iota_l]_c}$$

Merge-4: ($\delta \neq \emptyset$)

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}{=}\mathrm{x}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k]_c,\ [(\mathrm{v},\ \mathrm{w}){:}\mathrm{x}\delta,\ \iota_1,\ \ldots,\ \iota_l]_t}{[(\mathrm{p},\ \mathrm{q}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_k,\ (\mathrm{v},\ \mathrm{w}){:}\delta,\ \iota_1,\ \ldots,\ \iota_k]_c}$$

Move-1:

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}{+}\mathrm{y}\gamma,\ \alpha_1,\ \ldots,\ \alpha_{i-1},\ (\mathrm{v},\ \mathrm{p}){:}{-}\mathrm{y},\ \alpha_{i+1},\ \ldots,\ \alpha_k]_c}{[(\mathrm{v},\ \mathrm{q}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_{i-1},\ \alpha_{i+1},\ \ldots,\ \alpha_k]_c}$$

Move-2: ($\delta \neq \emptyset$)

$$\frac{[(\mathrm{p},\ \mathrm{q}){:}{+}\mathrm{y}\gamma,\ \alpha_1,\ \ldots,\ \alpha_{i-1},\ (\mathrm{v},\ \mathrm{w}){:}{-}\mathrm{y}\delta,\ \alpha_{i+1},\ \ldots,\ \alpha_k]_c}{[(\mathrm{p},\ \mathrm{q}){:}\gamma,\ \alpha_1,\ \ldots,\ \alpha_{i-1},\ (\mathrm{v},\ \mathrm{w}){:}\delta,\ \alpha_{i+1},\ \ldots,\ \alpha_k]_c}$$

The rules Move-1 and Move-2 come with an additional condition on their application: if $(\mathrm{x}_j,\ \mathrm{y}_j){:}\gamma_j$ is one of the $\alpha_1,\ \ldots,\ \alpha_{i-1},\ \alpha_{i+1},\ \ldots,\ \alpha_k$, then the first feature of γ_j is not `-f`. For all rules the following holds: $0 \leq \mathrm{p},\ \mathrm{q},\ \mathrm{v},\ \mathrm{w} \leq \mathrm{n}$; $0 \leq \mathrm{i},\ \mathrm{k},\ \mathrm{l} \leq \mathrm{m}$; and $\mathrm{t} \in \{\mathrm{s},\ \mathrm{c}\}$, $=\mathrm{x} \in$ *select*, $\mathrm{x} \in$ *base*, $+\mathrm{y} \in$ *licensors*, and $-\mathrm{y} \in$ *licensees*.

Example 2: The deductive system for the minimalist grammar G given in example 1 has four axioms: $[(0,\ 1){:}{=}\mathrm{ba}{-}\mathrm{p}]_s$, $[(2,\ 3){:}\mathrm{b}{-}\mathrm{q}]_s$, $[(3,\ 4){:}{=}\mathrm{a}{+}\mathrm{qd}]_s$, and $[(1,\ 2){:}{=}\mathrm{d}{+}\mathrm{pc}]_s$. The goal item $[(0,\ 4){:}\mathrm{c}]_c$ is deduced in the following way (in this simple example the agenda will not be mentioned explicitly): $[(0,\ 1){:}{=}\mathrm{ba}{-}\mathrm{p}]_s$, $[(2,\ 3){:}\mathrm{b}{-}\mathrm{q}]_s \Rightarrow [(0,\ 1){:}\mathrm{a}{-}\mathrm{p},\ (2,\ 3){:}{-}\mathrm{q}]_c$ (Merge-2); $[(3,\ 4){:}{=}\mathrm{a}{+}\mathrm{qd}]_s$, $[(0,\ 1){:}\mathrm{a}{-}\mathrm{p},\ (2,\ 3){:}{-}\mathrm{q}]_c \Rightarrow [(3,\ 4){:}{+}\mathrm{qd},\ (0,\ 1){:}{-}\mathrm{p},\ (2,\ 3){:}{-}\mathrm{q}]_c$ (Merge-2); $[(3,\ 4){:}{+}\mathrm{qd},\ (0,\ 1){:}{-}\mathrm{p},\ (2,\ 3){:}{-}\mathrm{q}]_c \Rightarrow [(2,\ 4){:}\mathrm{d},\ (0,\ 1){:}{-}\mathrm{p}]_c$ (Move-1); $[(1,\ 2){:}{=}\mathrm{d}{+}\mathrm{pc}]_s$, $[(2,\ 4){:}\mathrm{d},\ (0,\ 1){:}{-}\mathrm{p}]_c \Rightarrow [(1,\ 4){:}{+}\mathrm{pc},\ (0,\ 1){:}{-}\mathrm{p}]_c$ (Merge-1); $[(1,\ 4){:}{+}\mathrm{pc},\ (0,\ 1){:}{-}\mathrm{p}]_c \Rightarrow [(0,\ 4){:}\mathrm{c}]_c$ (Move-1). □

Since Shieber et al. ([7]) proved that the deductive procedure is sound and complete, establishing the correctness of the recognizer entails showing that the deductive system defined above is sound and complete relative to the intended interpretation of the items. This will be done in the next two sections.

4 Proof of Soundness

The following two lemma's will be helpful for establishing soundness of the deductive system.

Lemma 1a: if τ, τ' and υ, υ' are trees such that *merge*(τ, υ)=[$_<\tau'$, υ'] or *merge*(τ, υ)=[$_>\upsilon'$, τ'], then $\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau)$ and $\mathrm{Y}_n(\upsilon')=\mathrm{Y}_n(\upsilon)$.

Proof: inspection of the definition of *merge* shows that in both cases τ' and τ are identical except for the labels of their heads. According to the definition of narrow yield the label of the head of a tree is of no relevance for determining the narrow yield of that tree. Analogously, $\mathrm{Y}_n(\upsilon')=\mathrm{Y}_n(\upsilon)$. □

Lemma 1b: if τ, τ' and τ_0, τ_0' are trees such that *move*(τ)=[$_>\tau_0'$, τ'], then $\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau)$ and $\mathrm{Y}_n(\tau_0')=\mathrm{Y}_n(\tau_0)$.

Proof: according to the definition of *move*, τ' will be like τ except that feature +y is deleted and a subtree τ_0 has been replaced by a single node with no features. As is the case for *merge*, the different head labels of τ and τ' are irrelevant as narrow yields are concerned. Furthermore, the yield of τ_0 is excluded from $\mathrm{Y}_n(\tau)$ because τ_0 has a feature -f $\in$ *licensees*. The yield of τ_0 is also excluded from $\mathrm{Y}_n(\tau')$ because τ_0 is not a subtree of τ'. Since trees τ and τ' are otherwise the same, $\mathrm{Y}_n(\tau')=\mathrm{Y}_n(\tau)$. Concerning τ_0' and τ_0, these trees differ only by their head label. Hence, $\mathrm{Y}_n(\tau_0')=\mathrm{Y}_n(\tau_0)$. □

Proving soundness of the recognizer amounts to showing that the axioms and the rules of inference of the deductive system are sound. Then it will follow that every derivable item in the deductive system will represent a true grammatical statement under the intended interpretation.

4.1 Soundness of the Axioms

According to the interpretation given in section 3.2, an axiom [(i, j):γ]$_s$ asserts the existence of a tree $\tau \in$ CL(G) with the following properties:

1. Tree τ is a simple tree.
2. The head of τ is labeled by $\gamma\pi\iota$, for some $\pi\iota \in$ P*I*.
3. Besides the head of τ there are no other nodes with syntactic features in τ.
4. The narrow yield of τ is $\mathrm{w}_{i+1} \ldots \mathrm{w}_j$.

It is easy to see that the lexical item in Lex which occasioned the axiom [(i, j):γ]$_s$ has exactly these properties. By definition this lexical item is in $\mathrm{CL}^0(\mathrm{G}) \subseteq \mathrm{CL}(\mathrm{G})$.

4.2 Soundness of the Rules of Inference

There are six rules of inference. Their soundness will be established below.[7]

Merge-1: the items [(p, q):=xγ]$_s$ and [(q, v):x, α_1, …, α_k]$_t$ in the antecedent of the rule Merge-1 assert the existence of trees τ and $\upsilon \in$ CL(G), whose heads are labeled by =xγ and x, respectively. Hence, τ and υ are in the domain of the function *merge*. This function will apply to τ and υ and produce a complex tree ϕ=[$_<\tau'$, υ'] $\in$ CL(G), since τ is a simple tree. The head of ϕ is labeled by γ. Furthermore, $\mathrm{Y}_n(\phi)=\mathrm{Y}_n(\tau')\cdot\mathrm{Y}_n(\upsilon')$, since the head of υ' does not have a feature -f $\in$ *licensees*. By lemma 1a, $\mathrm{Y}_n(\tau')\cdot\mathrm{Y}_n(\upsilon')=\mathrm{Y}_n(\tau)\cdot\mathrm{Y}_n(\upsilon)=\mathrm{w}_{p+1}\ldots\mathrm{w}_q\cdot\mathrm{w}_{q+1}\ldots\mathrm{w}_v=\mathrm{w}_{p+1}\ldots\mathrm{w}_v$. Also, all maximal subtrees properly contained in υ will be included in ϕ with their head labels and narrow yields unchanged, since *merge* does not touch any proper subtrees of υ. As is easy to check, the item [(p,

[7]Since applicability of the structure building functions and the rules of inference does not depend on non-syntactic features, their presence in trees will be ignored in the discussion to follow. Thus, the statement 'the head of tree ϕ is labeled by γ', for example, means: 'the head of tree ϕ is labeled by $\gamma\pi\iota$, for some $\pi\iota \in$ P*I*'.

v):γ, $\alpha_1, \ldots, \alpha_k]_c$ in the consequent of the rule Merge-1 claims the existence of a tree in CL(G) with the properties of ϕ. Thus Merge-1 is sound.

Merge-2: application of the Merge-2 presupposes the existence of trees τ, $\upsilon \in$ CL(G) which will produce another tree ϕ=*merge*(τ, υ)=$[_<\tau', \upsilon'] \in$ CL(G). According to the definition of the function *Label*, the first feature of δ, which is the label of the head of υ', must be a feature -f $\in$ *licensees*. Therefore, by the definition of narrow yield and lemma 1a, $Y_n(\phi)=Y_n(\tau')=Y_n(\tau)=w_{p+1} \ldots w_q$. Obviously, υ', whose head is labeled δ, will be a maximal subtree of ϕ. By lemma 1a, $Y_n(\upsilon')=Y_n(\upsilon)=w_{v+1} \ldots w_w$. Now it is easy to see that the grammatical claim made by the item in the consequent of the rule Merge-2 is justified by $\phi \in$ CL(G).

Merge-3: the argument unfolds in a fashion similar to Merge-1, except that now $\phi=[_>\upsilon', \tau']$, because τ is complex. Consequently, $Y_n(\phi)=Y_n(\upsilon) \cdot Y_n(\tau)=w_{v+1} \ldots w_p \cdot w_{p+1} \ldots w_q = w_v \ldots w_q$. Furthermore, ϕ inherits the narrow yields and head labels of the maximal subtrees in both τ and υ.

Merge-4: this rule of inference is treated analogously to Merge-2, with the provisos mentioned under Merge-3.

Move-1: rule Move-1 will apply provided there is a complex tree $\tau \in$ CL(G) whose head is labeled by +yγ, which contains one leaf labeled by a feature -y and no other leaves whose first feature is -y. Let τ_0 be the maximal subtree in τ projected by the node labeled with the single feature -y.[8] Then τ is in the domain of the function *move*. The result of applying *move* to τ will be a complex tree $\phi=[_>\tau_0', \tau'] \in$ CL(G). The head of ϕ is labeled γ. Moreover, $Y_n(\phi)=Y_n(\tau_0') \cdot Y_n(\tau')=Y_n(\tau_0) \cdot Y_n(\tau)$ by lemma 1b and because the head of τ_0' has an empty label. Since $Y_n(\tau_0)=w_{v+1} \ldots w_p$ and $Y_n(\tau)=w_{p+1} \ldots w_q$, $Y_n(\phi)=w_{v+1} \ldots w_q$. The function *move* does not affect the labels or narrow yields of any of the maximal subtrees properly contained in τ, except for τ_0 (cf. the third case in the proof of lemma 2 in section 5). Now it is easy to see that the item in the consequent of the inference rule Move-1 actually describes $\phi \in$ CL(G).

Move-2: application of rule Move-2 requires the existence in CL(G) of a complex tree τ to which the function *move* will apply to produce a complex tree $\phi=[_>\tau_0', \tau'] \in$ CL(G), whose head is labeled γ, τ_0' and τ' as in the definition of *move*. $Y_n(\phi)=Y_n(\tau')$, since τ_0', whose head is labeled by δ, has a feature -f $\in$ *licensees*, cf. similar situations in Merge-2 and Merge-4. As in Move-1, the narrow yields and labels of maximal subtrees properly contained in τ are left intact, except for τ_0. τ_0 is not a subtree of ϕ, but τ_0' is. The label of the head of τ_0' is δ and τ_0''s narrow yield is $Y_n(\tau_0')=Y_n(\tau_0)$, by lemma 1b. It is easily checked that $\phi \in$ CL(G) has the same properties as the tree claimed to exist by the item in the consequent of Move-2. Hence, Move-2 is sound.

5 Proof of Completeness

This section presents a completeness proof for the recognizer. A recognizer is complete if for every string that is in the language defined by the grammar, there is a derivation of a goal item from the axioms.

First, the following two useful lemma's will be proved.

Lemma 2: if the derivation of tree ϕ in a minimalist grammar G immediately includes tree τ, then for every maximal subtree σ contained in τ, there is a maximal subtree σ' in ϕ such that $Y_n(\sigma)$ is a substring of $Y_n(\sigma')$.

[8] $\tau \neq \tau_0$ because their head labels are different.

Proof: three distinct cases have to be considered: there is an $\upsilon \in$ CL(G) such that ϕ=*merge*(τ, υ), there is an $\upsilon \in$ CL(G) such that ϕ=*merge*(υ, τ), and ϕ=*move*(τ).

In the first case, either $\phi=[_<\tau', \upsilon']$ or $\phi=[_>\upsilon', \tau']$, depending on whether τ is a simple or a complex tree. Furthermore, either σ is a proper subtree of τ, or $\sigma=\tau$. If σ is a proper subtree of τ, then τ' will properly contain σ, as τ' is like τ except that =x is deleted from the label of the head. For the same reason, σ is maximal in τ'. Since ϕ contains τ', σ will be a subtree of ϕ and, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\sigma)$. If $\sigma=\tau$, then $Y_n(\sigma)=Y_n(\tau)=Y_n(\tau')$ by lemma 1a. Also, $Y_n(\tau')$ is a substring of $Y_n(\phi)$ by the definition of narrow yield. Trivially, ϕ is a maximal subtree of ϕ.

In the second case, either $\phi=[_<\upsilon', \tau']$ or $\phi=[_>\tau', \upsilon']$, depending on whether υ is a simple or a complex tree. Again, either σ is a proper subtree of τ, or $\sigma=\tau$. If σ is a proper subtree of τ, then σ will be a proper, maximal subtree of ϕ, as argued for the similar situation in the case above. If $\sigma=\tau$, then $Y_n(\sigma)=Y_n(\tau)=Y_n(\tau')$ by lemma 1a. Tree τ' is a maximal subtree contained in ϕ, and, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\tau')$.

In the third case, $\phi=[_>\tau_0', \tau']$, with τ_0 specified as in the definition of *move*. Either σ is a proper subtree of τ_0, or $\sigma=\tau_0$, or σ is a proper subtree of τ and τ_0 is a proper subtree of σ, or $\sigma=\tau$. If σ is a proper subtree of τ_0, then σ will be a proper, maximal subtree of ϕ, as in the similar situations above. If $\sigma=\tau_0$, then $Y_n(\sigma)=Y_n(\tau_0)=Y_n(\tau_0')$ by lemma 1b. Now, τ_0' is a maximal subtree contained in ϕ, and, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\tau_0')$. If σ is a proper subtree of τ and τ_0 is a proper subtree of σ, then, by the definition of *move*, τ' will contain a tree σ' which is like σ, except that subtree τ_0 is replaced by a single node without features. $Y_n(\sigma)=Y_n(\sigma')$, since the yield of τ_0 is excluded from $Y_n(\sigma)$ because of its -f feature, and the yield of τ_0 is excluded from $Y_n(\sigma')$ because τ_0 is not a subtree of σ'. Hence, trivially, $Y_n(\sigma)$ is a substring of $Y_n(\sigma')$. Moreover, σ' is a maximal subtree of τ' since σ is a proper, maximal subtree of τ. Finally, if $\sigma=\tau$, then $Y_n(\sigma)=Y_n(\tau)=Y_n(\tau')$ by lemma 1b. Furthermore, $Y_n(\tau')$ is a substring of $Y_n(\phi)$ by the definition of narrow yield. Trivially, ϕ is a maximal subtree contained in ϕ. □

Lemma 3: if for a minimalist grammar G, the derivation of a complete tree γ includes τ, then $Y_n(\tau)$ is a substring of $Y(\gamma)$.

Proof: by repeated application of lemma 2, γ will contain a maximal subtree σ' such that $Y_n(\tau)$ is a substring of $Y_n(\sigma')$. Since γ is a complete tree, it does not have any labels with a feature -f $\in$ *licensees*. Hence, by the definition of narrow yield, $Y_n(\sigma')$ is a substring of of $Y_n(\gamma)=Y(\gamma)$, and, consequently, $Y_n(\tau)$ is a substring of $Y(\gamma)$. □

Completeness of the parser will follow as a corollary of lemma 4 below. If w $\in$ L(G), for some minimalist grammar G, then there is a complete tree $\phi \in$ CL(G) such that $Y(\phi)$=w. Since $\phi \in$ CL(G), there must be a k $\in \mathbb{N}$ such that $\phi \in \mathrm{CL}^k$(G). Since ϕ is a complete tree, lemma 4 guarantees that an item corresponding to ϕ will be generated. Obviously, the item will be a goal item, since ϕ is complete and $Y(\phi)$=w.

Lemma 4: for a given minimalist grammar G, if $\phi \in \mathrm{CL}^k$(G), k $\in \mathbb{N}$, and ϕ is included in the derivation of a complete tree or ϕ is a complete tree itself, then an item $[\alpha_0, \alpha_1 \ldots, \alpha_m]_t$ corresponding to ϕ will be generated, $[\alpha_0, \alpha_1 \ldots, \alpha_m]_t$ as defined in section 3.2.

Proof:[9] it will be shown by induction over k that for arbitrary k $\in \mathbb{N}$ and $\phi \in \mathrm{CL}^k$(G) such that ϕ is included in the derivation of a complete tree or ϕ is a complete tree itself, an item corresponding to ϕ will be generated.

[9]As in the discussion of the soundness proof, the presence of non-syntactic features in trees will be ignored.

1. k=0. Then $\phi \in \mathrm{CL}^0(\mathrm{G})$=Lex, and ϕ is covered by one of the axioms.

2. Assume all items corresponding to $\phi \in \mathrm{CL}^k(\mathrm{G})$, ϕ included in the derivation of a complete tree or a complete tree itself, are generated for a particular $k \in \mathbb{N}$. To show: for any $\phi \in \mathrm{CL}^{k+1}(\mathrm{G})$, ϕ included in the derivation of a complete tree or a complete tree itself, a corresponding item will be generated.

Pick an arbitrary $\phi \in \mathrm{CL}^{k+1}(\mathrm{G})$. By definition, $\mathrm{CL}^{k+1}(\mathrm{G})=\mathrm{CL}^k(\mathrm{G}) \cup \{merge(\tau, \upsilon)|(\tau, \upsilon) \in \mathrm{Dom}(merge) \cap \mathrm{CL}^k(\mathrm{G})\times\mathrm{CL}^k(\mathrm{G})\} \cup \{move(\tau)|\tau \in \mathrm{Dom}(move) \cap \mathrm{CL}^k(\mathrm{G})\}$. Hence, three cases can be distinguished: $\phi \in \mathrm{CL}^k(\mathrm{G})$, $\phi \in \{merge(\tau, \upsilon)|(\tau, \upsilon) \in \mathrm{Dom}(merge) \cap \mathrm{CL}^k(\mathrm{G})\times\mathrm{CL}^k(\mathrm{G})\}$, and $\phi \in \{move(\tau)|\tau \in \mathrm{Dom}(move) \cap \mathrm{CL}^k(\mathrm{G})\}$.

In the first case, $\phi \in \mathrm{CL}^k(\mathrm{G})$. Then, by the induction hypothesis, an item corresponding to ϕ will be generated.

In the second case, $\phi \in \{merge(\tau, \upsilon)|(\tau, \upsilon) \in \mathrm{Dom}(merge) \cap \mathrm{CL}^k(\mathrm{G})\times\mathrm{CL}^k(\mathrm{G})\}$. Then there are trees τ, $\upsilon \in \mathrm{CL}^k(\mathrm{G})$ such that $\phi=merge(\tau, \upsilon)$. Since ϕ is included in the derivation of a complete tree or ϕ is a complete tree itself, τ and υ are included in the derivation of a complete tree.[10] Hence, by the induction hypothesis, there are items corresponding to τ and υ. Since $(\tau, \upsilon) \in \mathrm{Dom}(merge)$, the heads of τ and υ are respectively labeled =xγ and xδ, for =x $\in$ *licensees*, x $\in$ *base* and γ, $\delta \in$ Cat*. Tree τ is either a simple tree or a complex tree. With regard to υ, $\delta=\emptyset$ or $\delta\neq\emptyset$. Hence, there are four cases to be dealt with. If τ is a simple tree and $\delta=\emptyset$, i.e., the head of υ consists of the single feature x, then $\phi=[_<\tau', \upsilon']$, and $Y_n(\phi)=Y_n(\tau')\cdot Y_n(\upsilon')$. By lemma 1a, $Y_n(\phi)=Y_n(\tau)\cdot Y_n(\upsilon)$. Suppose ϕ participates in a successful derivation of a complete tree whose yield is the string $w_1 \ldots w_m$.[11] Then, by lemma 3, $Y_n(\tau)\cdot Y_n(\upsilon)$ is a substring of $w_1 \ldots w_m$, that is, $Y_n(\tau)=w_p \ldots w_q$ and $Y_n(\upsilon)=w_{q+1} \ldots w_v$, $1 \leq p \leq q \leq v \leq m$. Hence, the items corresponding to τ and υ will match the antecedents of rule Merge-1. Alternatively, ϕ is a complete tree itself. Assume $Y_n(\phi)=Y(\phi)=w_1 \ldots w_m$. Then it follows immediately that $Y_n(\tau)=w_1 \ldots w_q$ and $Y_n(\upsilon)=w_{q+1} \ldots w_m$, $1 \leq q \leq m$. Again, the items corresponding to τ and υ will match the antecedents of rule Merge-1. In both cases, application of Merge-1 will generate the item corresponding to ϕ. In a similar way it is established that the trees τ and υ for the other three cases will correspond to items that match the antecdents of the rules Merge-1, Merge-3, and Merge-4. Application of these rules will produce an itemn corresponding to tree ϕ.

In the third case, $\phi \in \{move(\tau)|\tau \in \mathrm{Dom}(move) \cap \mathrm{CL}^k(\mathrm{G})\}$. Then there is a $\tau \in \mathrm{CL}^k(\mathrm{G})$ such that $\phi=move(\tau)$. Since ϕ is included in the derivation of a complete tree or ϕ is a complete tree itself, τ is included in the derivation of a complete tree as well. Hence, by the induction hypothesis, there is an item corresponding to τ. Since $\tau \in \mathrm{Dom}(move)$, τ has a feature +y and τ has exactly one maximal subtree that has the feature −y, +y $\in$ *licensors*, −y $\in$ *licensees*. Let τ_0 be the maximal subtree of τ that has feature −y and let the label of its head be −yδ. Then there are two cases to be considered: $\delta=\emptyset$ and $\delta\neq\emptyset$. If $\delta=\emptyset$, then $\phi=[_>\tau_0', \tau']$, and $Y_n(\phi)=Y_n(\tau_0')\cdot Y_n(\tau')=Y_n(\tau_0)\cdot Y_n(\tau)$ by the definition of narrow yield and lemma 1b. As for Merge-1 and Merge-3 above, $Y_n(\tau_0)\cdot Y_n(\tau)$ will be a substring of $w_1 \ldots w_m$, where $w_1 \ldots w_m$ is the yield of a complete tree derived from ϕ or the yield of ϕ itself if ϕ is a complete tree. Therefore, $Y_n(\tau_0)=w_v \ldots w_p$ and $Y_n(\tau)=w_{p+1} \ldots w_q$, $1 \leq v \leq p \leq q \leq m$. Hence, the item corresponding to τ will match the antecedent of rule Move-1. Application of this rule will generate the item corresponding to ϕ. If $\delta\neq\emptyset$, then the item generated for τ will match the antecedent

[10] Since τ is the first argument of *merge*, τ cannot be a complete tree.

[11] Reference to the yield of a complete tree derived from ϕ precludes a general proof of completeness, i.e. a proof that for all $\phi \in \mathrm{CL}^k(\mathrm{G})$, $k \in \mathbb{N}$, a corresponding item $[\alpha_0, \alpha_1 \ldots, \alpha_m]_t$ will be generated.

of rule Move-2, application of which will generate an item corresponding to ϕ. □

6 Complexity Results

For a given minimalist grammar G=(V, Lex, Cat, F) and input string of length n, the number of items is polynomially bounded by the length of the input string. All items are of the form $[(x_0, y_0){:}\gamma_0, (x_1, y_1){:}\gamma_1, \ldots, (x_m, y_m){:}\gamma_m]_t$, as defined in section 3.2. Each part $(x_i, y_i){:}\gamma_i$ of an item, $0 \leq i \leq m$, has $O(n^2)$ possible instantiations, as both x_i and y_i range between 0 and n, 0 and n included. The possible choices of γ_i do not depend on the length of the input string. The number of choices is bounded, however, because the labels occurring in any tree in CL(G) are substrings of the labels of the expressions in Lex. This follows immediately from the the definition of *merge* and *move*. Moreover, Lex is a finite set and the labels assigned by *Label* are finite sequences. Thus, the set of possible labels is bounded by the grammar, and, since the recognizer is sound, the number of possible γ_i's is bounded by the grammar, too. Since an item has at most k+1 parts $(x_i, y_i){:}\gamma_i$, where k=|*licensees*|, the number of items in the chart is bounded by $O(n^{2k+2})$.

As regards the time complexity of the recognizer, step 2.b) of the deduction procedure specified in section 3.1 requires every item on the agenda to be compared with the items already in the chart. Since the number of items in the chart is $O(n^{2k+2})$, this step will take $O(n^{2k+2})$ per item on the agenda. For any item on the agenda but not already in the chart, step 2.c) of the deduction procedure checks whether any rule of inference will apply. Checking applicability of the Merge rules involves looking through all the items in the chart, since all Merge rules have two antecedents. Given an item on the agenda and an item from the chart, actually verifying whether any of the Merge rules applies to these items takes constant time. Thus, the time cost is $O(n^{2k+2})$ per item on the agenda. In order to determine whether one of the two Move rules will apply, the label γ_0 in an item has to be inspected and compared to the other labels γ_i, $1 \leq i \leq m$ in the same item. Since m is bounded by k=|*licensees*|, there is no dependency on the length of the input string. Since steps 2.b) and 2.c) are performed in sequence, the time cost of both steps is bounded by $O(n^{2k+2})$ per item on the agenda, ignoring without loss of generality the fact that step 2.c) is not performed for all items on the agenda. Steps 1. and 3. of the deduction procedure do not exceed this bound. The number of items that will be put on the agenda while recognizing a string is $O(n^{2k+2})$. This is the upperbound on the number of possible items. There will be duplicates in the agenda, but their number is finite and does not depend on n, essentially because the number of axioms and the number of inference rules is finite and all items in the chart are unique. Thus, the overall time complexity of the recognizer is $O(n^{4k+4})$.

7 Conclusions and Future Work

In this paper we have provided a formal definition of a recognizer for minimalist grammars, together with proofs of completeness and soundness and an analysis of its space and time complexity.

There are several issues that deserve further investigation. First of all, the recognizer has to be developed into a parser. This can be done by either extending the items with a field for recording the derivational history of an item, or by devising a method for retrieving derivation trees from the chart. Secondly, we conjecture that the efficiency of the parser can be greatly improved by imposing some order on the chart. In the current recognizer, the entire chart is searched in order to determine whether any one of the Merge rules will apply. The definitions of the Merge rules suggest that grouping the items in the chart according to the first feature of their 'head labels' will allow for a more efficient search. The current recognizer operates in a bottom-up manner: no constituent is recognized until all substrings that make up its yield have been encountered. So, thirdly, it would be interesting to investigate other recognition strategies. Finally, there are still some open questions with regard to the formal power of minimalist grammars in comparison with other grammar formalisms. Careful examination of the complexity of the algorithms for recognizing and parsing these various grammar formalisms might answer some of the questions.

Acknowledgements

Thanks to Ed Stabler and Crit Cremers for inspiring discussions.

References

[1] Boullier, P. 1998. Proposal for a Natural Language Processing Syntactic Backbone. *Research Report N° 3342*, `http://www.inria.fr/RRRT/RR-3342.html`. INRIA-Rocquencourt.

[2] Chomsky, N. 1995. *The Minimalist Program.* MIT Press.

[3] Kayne, R. 1999. A note on Prepositions and Complementizers. In: *A Celebration.* MIT Press.

[4] Koopman, H. and A. Szabolsci. Forthcoming. *Verbal Complexes.* MIT Press.

[5] Michaelis, J. 1998. Derivational Minimalism is Mildly Context-Sensitive. In: *Proceedings, Logical Aspects of Computational Linguistics*, Grenoble.

[6] Seki, H., T. Matsumura, M. Fujii and T. Kasami. 1991. On Multiple Context-Free Grammars. In: *Theoretical Computer Science*, 88.

[7] Shieber, S.M., Y. Shabes and F.C.N. Pereira. 1995. Principles and Implementation of Deductive Parsing. In: *Journal of Logic Programming*, 24.

[8] Stabler, E.P. 1997. Derivational Minimalism. In: *Logical Aspects of Computational Linguistics.* C. Retoré (ed.). Lecture Notes in Artificial Intelligence 1328.

[9] Stabler, E.P. 1999. Remnant Movement and Complexity. In: *Constraints and Resources in Natural Language Syntax and Semantics.* G. Bouma, E. Hinrichs, G.-J. Kruijff, D. Oerhle (eds.). CSLI.

[10] Stabler, E.P. Forthcoming. Performance models for a Derivational Minimalism. In: *Linguistic Form and its Computation*, Final Workshop of SFB340.

[11] Vijay-Shanker, K., D.J. Weir and A.K. Joshi. 1987. Descriptions Produced by Various Grammatical Formalisms. In: *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics.*

[12] Weir, D.J. 1988. *Characterizing Mildly Context-Sensitive Grammar Formalisms.* Ph.D. dissertation, University of Pennsylvania.

A NEURAL NETWORK PARSER THAT HANDLES SPARSE DATA

James Henderson
University of Exeter
School of Engineering and Computer Science
Exeter EX4 4PT, UK
J.B.Henderson@exeter.ac.uk

Abstract

Previous work has demonstrated the viability of a particular neural network architecture, Simple Synchrony Networks, for syntactic parsing [6]. Here we present additional results on the performance of this type of parser, including direct comparisons on the same dataset with a standard statistical parsing method, Probabilistic Context Free Grammars. We focus these experiments on demonstrating one of the main advantages of the SSN parser over the PCFG, handling sparse data. We use smaller datasets than are typically used with statistical methods, resulting in the PCFG finding parses for under half of the test sentences, while the SSN finds parses for all sentences. Even on the PCFG's parsed half, the SSN performs better than the PCFG, as measure by recall and precision on both constituents and a dependency-like measure.

1 Introduction

In many domains neural networks are an effective alternative to statistical methods. This has not been the case for syntactic parsing, but recent work has identified a viable neural network architecture for this problem (SSN) [6], [7]. This alternative parsing technology is potentially of interest because neural networks have different strengths from statistical methods, and thus may be more applicable to some tasks. Like statistical methods, neural networks are robust against noise in the input and errors in the data. But unlike statistical methods, neural networks are particularly good at handling sparse data. In order to compensate for the necessarily limited amount of data available, statistical methods must make strong independence assumptions. These assumptions lead to undesirable biases in the model generated, and may still not guarantee coverage of less frequent cases. Neural networks also require independence assumptions in order to define their input-output format, but these assumptions can be much weaker. Because neural networks learn their own internal representations, neural networks can decide automatically what features to count and how reliable they are for predicting the output.

In this paper we empirically investigate the ability of SSNs to handle sparse data, relative to the statistical method Probabilistic Context Free Grammars (PCFGs) and a statistical version of the SSN parser. To test this ability we use a small dataset relative to those typically used with statistical methods. We train all the models on the same dataset and compare their results, both in terms of coverage and performance on the covered portion. The statistical version of the SSN parser has good coverage, but its performance is worse than both the other two models. The PCFG covers under half of the test sentences. The SSN produces a parse for all of the sentences, while still achieving better performance than that of the PCFG on the PCFG's parsed sentences, as measure by recall and precision on both constituents and a dependency-like measure.

2 Simple Synchrony Networks

The neural network architecture used in this paper has previously been shown to be a viable parsing technology based on both initial empirical results [6] and the linguistic characteristics of the basic computational model [5]. Their appropriateness for application to syntactic parsing is a result of their ability to learn generalisations over structural constituents. This generalisation ability is a result of using a neural network method for representing entities, called Temporal Synchrony Variable Binding (TSVB) [12]. In Simple Synchrony Networks (SSNs) this method is used to extend a standard neural network architecture for processing sequences, Simple Recurrent Networks (SRNs) [3]. SRNs can learn generalisations over positions in an input sequence (and thus can handle unbounded input sequences), which has made them of interest in natural language processing. However, their output at any given time in the input sequence is an unstructured vector, thus making them inappropriate for recovering the syntactic structure of an input sentence. By using TSVB to represent the constituents in a syntactic structure, SRNs can be extended to also learn generalisations over structural constituents. The linguistic relevance of this class of generalisations is what accounts for the fact that SSNs generalise from training set to testing set in an appropriate way, as demonstrated in section 4. In this section we briefly outline the SSN architecture.

2.1 Representing Constituents

The problem of representing constituents in a neural network is an instance of a broader problem, representing entities in general. The difficulty is that each entity can have multiple features and the network needs to represent which features go with which entity. One proposal is to use pulsing units and represent the binding between the features of each entity using the synchrony of the units' activation pulses. This has been proposed on both biological grounds [13] and computational grounds [12]. Following the computational work we will call this method Temporal Synchrony Variable Binding. We use TSVB's pulsing units to represent information about syntactic constituents.

TSVB can be applied to a learning task by combining it with a standard neural network architecture. Because we need the network to process sequences as well as constituents, we use an architecture that combines TSVB with Simple Recurrent Networks. In addition to a layer of input units and a layer of output units, SRNs have a layer of memory units which store the internal state of the network from the previous position in the sequence. This allows SRNs to learn their own internal representation of the previous input history. To extend SRNs with TSVB, pulsing units need to be added. These units represent inputs, outputs, and internal state about individual constituents. In order to retain the SRN's ability to represent information about the situation as a whole, the architecture also has normal nonpulsing units. These units also provide a means for information about one constituent to be passed to other constituents.

Pulsing units can be added to SRNs in a variety of ways. One proposed class of such networks is called Simple Synchrony Networks (SSNs) [7]. The architectural restriction which defines SSNs is appropriate for the parser used here because information about only one constituent is input at any one time, namely the constituent headed by the current input word. This input simply specifies the current word. To accommodate SSN's restrictions we simply need to provide the same information through nonpulsing input units as we provide to this one constituent through the pulsing input units [6]. In this way each constituent gets information about its head word from the pulsing input units and gets information about every other word from the nonpulsing input units.

In this paper we use the simplest form of SSN, called type A in [7]. This architecture only has a single internal state layer and a single memory layer, both consisting of pulsing units. We use 100 units in each of these layers, which is a moderately large size, resulting in a ability to approximate a wide variety of functions.

2.2 Learning Generalisations over Constituents

The most important feature of any learning architecture is how it generalises from training data to testing data. SRNs are popular for sequence processing because they inherently generalise over sequence positions. Because inputs are fed to an SRN one at a time, and the same trained parameters (the link weights) apply at every time, information learned at one sequence position will inherently be generalised to other sequence positions. This generalisation ability manifests itself in the fact that SRNs can handle arbitrarily long sequences. The same argument applies to TSVB and constituents. Using synchronous pulses to represent constituents amounts to cycling through the set of constituents and processing them one at a time. Because the same trained parameters are applied at every time, information learned for one constituent will inherently be generalised to other constituents. This generalisation ability manifests itself in the fact that these networks can handle arbitrarily many constituents.

3 A SSN Parser

The Simple Synchrony Network architecture gives us the basic generalisation abilities that we need for syntactic parsing, but the specific way that a SSN parser generalises from training set to testing set also depends on the specific input-output format that we ask it to learn. The main challenge in defining an input-output format is that a syntactic structure consists of a set of relationships between constituents, whereas SSNs only provide us with outputs about individual constituents. More precisely, there are $O(n^2)$ possible relationships between constituents and only $O(n)$ outputs at any given time. The solution is simply to output the syntactic structure incrementally. By making use of the $O(n)$ positions in the input sentence[1] to produce output, the network can produce the necessary $O(n^2)$ outputs over the course of the parse. In this section we will specify the nature of this incremental output in more detail, and give specifics about the format of the input to the network. In each case, the importance of these choices is how they effect the way that the network will generalise.

3.1 The Model of Constituency

Before discussing the specific input-output pattern, it is necessary to specify what exactly we mean by "constituent". This is a crucial decision, since the constituent is the unit over which generalisations are learned. One answer would be to simply use the definition of constituency that is implicit in whatever corpus you have available. It would be possible for us to do this, but at the cost of a more complicated output format. Also, the import of constituents as the unit of generalisations may not be the one intended by the corpus writer. Instead we choose to take a lexicalised approach. Each constituent is associated with a word in the input, which is intended to be the head of that constituent. The constituent structure can then be thought of as a set of relationships between each constituent's

[1]Here we are assuming that the number of constituents is linear in the number of words. This is uncontroversial, being implied by any lexicalised or dependency-based grammar.

head word and the other words and constituents in the structure. This perspective is closely related to Dependency Grammar [8]. It is also closely related to Lexicalized Tree Adjoining Grammar [11] and the head-driven statistical parsing model by Collins [2], but in these cases our constituent needs to be mapped to the entire projection of a lexical head.

Unfortunately the corpus we will use in the below experiments does not label heads for constituents. As an approximation, because we know we are using a head-initial language, we will use the first word which is a direct child of a constituent as the constituent's head. If a constituent in the original corpus does not have any words as direct children, then that constituent is collapsed into one of its child constituents, as discussed further in section 4.1. Thus the corpus used in the below experiments is slightly flatter than most corpora. In particular it often does not distinguish between an S and its VP, the two constituents having been collapsed.

Using a lexicalised definition of constituency not only makes the unit of generalisation more linguistically appropriate, it has the added advantage that it provides a fixed mapping between the constituents in the network's output and the constituents in the corpus. Two constituents are the same if they have the same word as their head. Using this mapping it is easy to define a fixed input-output pattern for the network to learn during training, which is necessary for supervised learning algorithms such as the one used here. However, we should emphasise that it would be possible to define a different input-output format for an SSN parser which could use any corpus's definition of constituency.

3.2 The Independence Assumptions

As mentioned above, the independence assumptions made by a neural network model are embodied in the definition of its input-output format. If there is no way for information to flow from a given input to a given output, then the model is assuming that that output is independent of that input. For the SSN parser used here these assumptions are due to the incremental nature of the network's output.

As discussed above, the SSN parser solves the problem of outputting relationships between constituents by outputting them incrementally. In particular, this SSN parser outputs structural relationships maximally incrementally. Words are input to the parser one at a time, and with each word a new constituent is introduced to act as the constituent which the word heads, if needed.[2] A structural relationship is output as soon as both things in the relationship have been introduced. This incremental output is illustrated in figure 1. When the parse is complete the accumulation of all the outputs fully specifies the parse, as illustrated in figure 1.

The syntactic structure of a sentence can be fully specified by specifying the set of parent-child relationships in it. These come in two types, one where the parent is a constituent but the child is a word, and one where both the parent and the child are constituent. When the child is a word, then because we assume that the first word child of a constituent is its head, the parent constituent must have been introduced before or at the same time as the word is input. Thus as soon as a word is input, the network can output which constituent is the parent of that word. This is done with a single output unit, called the *parent* output, which pulses in synchrony with the parent of the current input word. For example when 'John' is input in figure 1 the new constituent, *c1*, is specified as the

[2]In the actual experiments below we use part-of-speech tags as inputs, not the actual words. We are using "words" here for expository convenience and clarity.

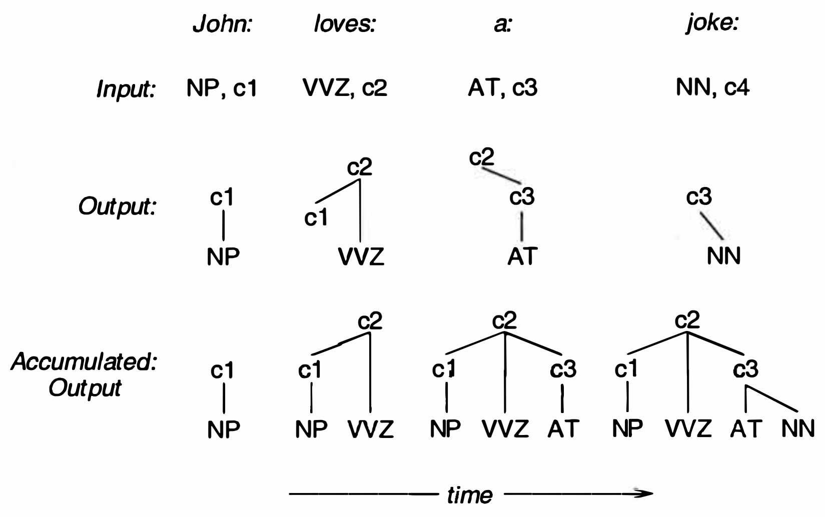


Figure 1: An example of the SSN parser's output.

parent, while when 'joke' is input the previous constituent, *c3*, is specified as the parent. When both the parent and child are constituents, then the relationships is output as soon as the second of the two constituents is input. If the second constituent is the child, then this output is specified using the *grandparent* output unit, which pulses in synchrony with the parent of the constituent which is headed by the current input word. For example when 'a' is input in figure 1 the previous constituent, *c2*, is specified as the grandparent. If the second constituent is the parent, then this output is specified using the *sibling* output unit, which pulses in synchrony with the child of the constituent which is headed by the current input word. For example when 'loves' is input in figure 1 the previous constituent, *c1*, is specified as a sibling. Given the previous assumptions, these three output units are all that is necessary to specify the syntactic structure of the sentence.

So far we have only discussed the output pattern which we would like the network to produce (the target output), but a network will actually output real values, not zeros and ones. To interpret this output we look at all the possible parents for a given child, and pick the one whose output activation is the highest. First we incrementally choose the parents for each word, based on the *parent* output unit's activations. This not only determines all parent-child relationships involving words, it also determines which of the introduced constituents have head words and thus are included in the output structure. For example in figure 1 constituent *c4* is not included in the output structure. Only these headed constituents candidates to be in relationships with other constituents or later words. Second we choose the best parent for these headed constituents, based on both the *grandparent* output unit's activations at the time of the constituent's head (for leftward attachments) and the *sibling* output unit's activations in synchrony with the constituent (for rightward attachments). This determines all the parent-child relationships between two constituents.[3] While the resulting set of parent-child relationships may not form a tree, this representation of the structure is sufficient to determine which words are included in each constituent, and thus to calculate recall and precision on constituents.

This maximally incremental output format implies some fairly strong independence assumptions. This is particularly true for the parents of words. No output after a word can influence which constituent is interpreted as its parent, and thereby this decision is assumed to be independent of later

[3]There is one distinguished constituent, introduced with a sentence-initialising input symbol, which does not require a parent. This acts as the root of the parse tree, in the same way as the start symbol of CFGs.

input words. For the parents of constituents there is a slight complication in that a later *sibling* output can override a *grandparent* output that is produced at the time of the constituent's head word. For example, in the sentence "John knows Mary left", the network may produce a fairly high *grandparent* output value for attaching 'Mary' to 'knows', but when 'left' is input an even higher *sibling* output value can override this and result in 'Mary' being attached to 'left'. However it is not possible for later outputs to affect a decision which does not involve a later word's constituent. For example, in the sentence "John saw the policeman with Mary", the decision of whether 'with' should be attached to 'saw' or 'the policeman' is not effected by any output at the time when 'Mary' is introduced, and thus the unsuitability of 'Mary' as an instrument of the seeing cannot be used. Thereby decisions between two possible parents for a constituent are assumed to be independent of any input words after the last head word of the three constituents involved.

Finally, in addition to outputs about structural relationships, the parser also outputs the labels of the constituents. This output is produced at the time when the constituent's head word is input, using a bank of output units, one for each label. Thus the decision of what label to give a constituent is assumed to be independent of all words after the constituent's head word. This assumption is mostly for convenience, since any output about an individual constituent could be delayed until the end of the sentence.[4]

3.3 The Soft Biases

Because all the words are input to the representation of each constituent, in theory any earlier input can effect a later output, and thus they are not being assumed to be independent. However, in practice there are ways to make the network pay more attention to some inputs than to others. In particular, a recurrent network such as an SSN will learn more about the dependencies between an input and an output if they are close together in time. The immediate effect of this is a form of recency bias; the most recent input words will effect an output more than earlier input words. To make use of this we provide a *new constituent* input pattern, which is correlated with being an output for a short time afterwards. We also provide a *last parent* input unit, which is the disambiguated *parent* output from the previous input word. This is also correlated with being an output for a short time afterwards. Finally, we bias the network to pay particular attention to the head word for each constituent by providing the head word as an input at every time during the life of a constituent. Thus an output at a given input word for a given constituent pays particular attention to the input word and the head of the constituent, with previous input words providing an influence proportional to their recency. This input format has been devised in part on the basis of the author's linguistic knowledge and in part on the basis of experimental results.[5]

4 Generalising from Sparse Data

To test the ability of Simple Synchrony Networks to handle sparse data we train the SSN parser described in the previous section on a relatively small set of sentences and then test how well it

[4]Because in the experiments below we are actually inputting part-of-speech tags and there are a fairly small number of possible labels, labelling of constituents is actually not difficult and thus this independence assumption is not a problem.

[5]All development decisions such as these have been made on the basis of the cross validation set used below. The testing set was not used until the final results reported in section 4.3 were produced. The one exception was a set of results collected before training had completed for the submitted version of this paper.

generalises to a set of previously unseen sentences. Because the training set is small, the testing sentences will contain many constituents and constituent contexts which the network has never seen before. Thus the network cannot simply choose the constituents which it has seen the most times in the training set, because in most cases it will have never seen any of the constituents which it needs to choose between. To handle this sparseness of training data the network must learn which of the inputs about a constituent are important, as well as how they correlate with the outputs. The advantage of SSNs is that they automatically learn the relative importance of different inputs as part of their training process.

To provide a comparison with the SSN parser we also apply a standard statistical method to the same data sets. We estimate the probabilities for two Probabilistic Context Free Grammars, one using just the network's training sentences and another using both the training sentences and the cross validation sentences. These PCFGs are then both tested on the network's testing sentences. PCFGs deal with the problem of sparse data by ignoring everything about a constituent other than its label. The strength of this independence assumption depends on how many constituent labels the corpus has, and thus how much information the labels convey. Because we are dealing with small training sets, we only use a small number of labels. Even so, the PCFGs have only seen enough CFG rules in the training sets to produce parses for about half of the test sentences. The problem with such statistical approaches is that information is either counted (i.e. increasing the number of labels) or ignored (i.e. decreasing the number of labels), and there is no middle ground.[6]

In addition to the PCFGs, we train a statistical model based on the output format of the SSN parser. This test is to control for the possibility that it is the linguistic assumptions discussed in the previous section which are responsible for the SSN parser's performance, and not the SSN architecture. Rather than estimating the probabilities for CFG rules like in PCFGs, this statistical model estimates the probabilities of the same structural relationships used in the output of the SSN parser (parent, grandparent, and sibling, plus label-head relationships). Thus we will call this model the Probabilistic Structural Relationships (PSR) model. The PSR model also makes all the independence assumptions made by the SSN parser, but in order to deal with the sparse data it must also make additional independence assumptions. Every structural relationship is dependent on the word involved in the relationship and the head word of the constituent involved in the relationship (and whether they are the same), but they are independent of all other words. This assumption imposes a hard bias that parallels the main soft biases provided by the SSN parser's input format. This independence assumption is strong enough to provide us with sufficient statistics given our training data, but still captures to the extent possible the relevant information for estimating the structural relationship probabilities. The PSR model is closely related to dependency-based statistical models, such as that in [2].

4.1 A Small Corpus

Work on statistical parsing typically uses a very large corpora of preparsed sentences (for example more than a million words in [1], [2], and [9]). Such corpora are very expensive to produce, and are currently only available for English. In addition, using a very large training corpus helps hide

[6]It should be noted that we are using the term "statistical method" here in a rather narrow sense, intending to reflect common practice in parsing technology. Indeed, neural networks themselves are a statistical method in the broader sense. We do not intend to imply that there are no other methods for addressing the problem of sparse data. For example, Maximum Entropy is a framework for deciding how strongly to believe different sources of information and has been applied to parsing [9], although with a much larger training set than that used here.

inadequacies in the parser's ability to generalise, because the larger the training corpus the better results can be obtained by simply memorising the common cases.[7] Here we use a training set of only 26,480 words, plus a cross validation set of 4365 words (30,845 words total). By using a small training set we are placing greater emphasis on the ability of the parser to generalise to novel cases in a linguistically appropriate way, and to do so robustly. In other words, we are testing the parser on its ability to deal with sparse data.

We use the Susanne[8] corpus as our source of preparsed sentences. The Susanne corpus consists of a subset of the Brown corpus, preparsed according to the Susanne classification scheme described in [10], and we make use of the "press reportage" subset of this. These parses have been converted to a format appropriate for this investigation, as described in the rest of this section.[9]

As is commonly done for PCFGs, we do not use words as the input to the parser, but instead use part-of-speech tags. The tags in the Susanne scheme are a detailed extension of the tags used in the Lancaster-Leeds Treebank (see [4]), but we use the simpler Lancaster-Leeds scheme. Each tag is a two or three letter sequence, for example 'John' would be encoded 'NP', the articles 'a' and 'the' are encoded 'AT', and verbs such as 'is' encoded 'VBZ'. There are 105 tags in total.

The parse structure in the Susanne scheme is also more complicated than is needed for our purposes. Firstly, the meta-sentence level structure has been discarded, leaving only the structures of individual sentences. Secondly, the 'ghost' markers have been removed. These elements are used to represent long distance dependencies, but they are not needed here because they do not effect the boundaries of the constituents. Third, as was discussed above, we simplify the nonterminal labels so as to help the PCFG deal with the small training set. We only use the first letter of each label, resulting in 15 nonterminal labels (including a new start symbol). Finally, as was discussed in section 3.1, some constituents need to be collapsed with one of their child constituents so that every constituent has at least one terminal child. There are very few constructions in the Susanne corpus that violate this constraint, but one of them is very common, namely the S-VP division. The head word of the S (the verb) is within the VP, and thus the S often occurs without any terminals as immediate children. In these cases, we collapse the S and VP into a single constituent, giving it the label S. The same is done for other such constructions. As discussed above, this change is not linguistically unmotivated.

The total set of converted sentences was divided into three disjoint subsets, one for training, one for cross validation, and one for testing. The division was done randomly, with the objective of producing cross validation and testing sets which are each about an eighth of the total set. No restrictions were placed on sentence length in any set. The training set has 26480 words, 15411 constituents, and 1079 sentences, the cross validation set has 4365 words, 2523 constituents, and 186 sentences, and the testing set has 4304 words, 2500 constituents, and 181 sentences.

4.2 Training the Models

The SSN parser was trained using standard training techniques extended for pulsing units [7]. Neural network training is an iterative process, in which the network is run on training examples and then modified so as to make less error the next time it sees those examples. This process can be continued until no more changes are made, but to avoid over-fitting it is better to check the performance of the

[7]Given this fact, it is unfortunate that much work on statistical parsing does not even report the number of words in their training set.

[8]We acknowledge the roles of the Economic and Social Research Council (UK) as sponsor and the University of Sussex as grant holder in providing the Susanne corpus used in the experiments described in this paper.

[9]We would like to thank Peter Lane for performing most of this conversion.

network on a cross validation set and stop training when this error reaches a minimum. This is why we have split the corpus into three datasets, one for training, one for cross validation, and one for testing. This technique also allows multiple versions of the network to be trained and then evaluated using the cross validation set, without ever using the testing set until a single network has been chosen. This is the technique which we have used in developing the specific network design reported in this paper. The resulting network trained for 325 passes through the training set before reaching a maximum of the average between its recall and precision on constituents in the cross validation set.

Estimating the parameters of a PCFG is straightforward. All the sequences of child labels that occur in the corpus for each parent label need to be extracted, counted, and normalised in accordance with the conditional probabilities required by the model. Because this process does not require a cross validation set, we create two PCFGs, one using only the network's training set and the other using both the training set and the cross validation set. The first provides a more direct comparison with the SSN parser's ability to deal with small training sets, but the second provides a better indication of how the two methods compare in practice.

Estimating the PSR model is also straightforward. All the head tag bigrams associated with each structural relationship (or label-head tag bigrams) are extracted, counted, and normalised in accordance with the probability model.[10] As with the second PCFG, we use the network's training set plus its cross validation set to estimate the probabilities.

In addition to embodying the same linguistic assumptions as the SSN parser, the PSR model has the advantage that it has a finite space of possible parameters (namely one probability per tag bigram for each relationship). Because a PCFG places no bound on the number of children that a parent constituent can have, a PCFG has an infinite space of possible parameters (namely one probability for each of the infinite number of possible rules). This makes it difficult to apply smoothing to a PCFG, to avoid the problem of assigning zero probability to rules that did not occur in the training set. Thus we have not applied smoothing to the PCFGs, contributing to the bad test set coverage discussed in the next section. However the PSR's finite space of parameters makes it simple to apply smoothing to the PSR model. Thus we apply a smoothing method to estimate the parameters for a second PSR model; before normalising we add half to all the counts so that none of them are zero.

4.3 Testing Results

Once all development and training had been completed, the SSN parser, the two PCFGs, and the two PSRs were tested on the data in the testing set. For the PCFGs and the PSRs the most probable parse according to the model was taken as the output of the parser.[11] The results of this testing are shown in table 1, where PCFG 1 is the PCFG that was produced using only the training set, PCFG 2 is the PCFG produced using both the training set and the cross validation set, PSR is the unsmoothed PSR, and PSR Sm is the smoothed PSR.

The first thing to notice about the testing results is that both PCFGs only found parses for about half of the sentences. For the unparsed sentences the PCFGs had not found enough rules in their training sets to construct a tree that spans the entire sentence, and thus there is no straightforward

[10]Briefly, in the probability model each structural relationship probability is conditional on the constituents' heads being correct, and the chain rule plus the independence assumptions from section 4 are used to combine these probabilities into the probability of the entire parse.

[11]We would like to thank Jean-Cedric Chappelier and the LIA-DI at EPFL, Lausanne, Switzerland for providing the tools used to train and test the PCFG.

	Sentences		Constituents		Parent-child	
	Parsed	Correct	Recall	Precision	Recall	Precision
SRN	100%	14.4%	65.1%	65.0%	83.0%	82.9%
PCFG 1	45.3%	3.3%	24.9%	54.0%	32.2%	72.4%
PCFG 2	50.8%	3.3%	29.2%	53.7%	38.2%	73.3%
PSR	90.6%	2.8%	37.0%	40.3%	63.1%	65.1%
PSR Sm	100%	2.8%	35.9%	36.8%	58.8%	59.4%

Table 1: Testing results.

		Sentences	Constituents		Parent-child	
		Correct	Recall	Precision	Recall	Precision
Parsed by PCFG 1	SRN	19.5%	66.6%	67.6%	83.7%	84.0%
	PCFG 1	7.3%	58.1%	54.0%	74.3%	72.4%
Parsed by PCFG 2	SRN	17.4%	65.4%	66.4%	83.4%	83.8%
	PCFG 2	6.5%	57.5%	53.7%	75.2%	73.3%

Table 2: Testing results on the sentences parsed by PCFG 1 and PCFG 2, respectively.

way to choose the most probable parse.[12] In contrast the SSN parser is able to make a guess for every sentence. This is a result of the definition of the SSN parser, as with the smoothed PSR, but unlike both PSR models the SSN parser produces good guesses. The first evidence of the quality of the SSN parser's guesses is that they are exactly correct more that 4 times as often as for any of the PCFGs or PSRs.

The remaining four columns in table 1 give the performance of each parser in terms of recall (percentage of desired which are output) and precision (percentage of output which are desired) on both constituents and parent-child relationships. An output constituent is the same as a desired constituent if they contain the same words and have the same label. This measure is a common one for comparing parsers. Parent-child relationships are the result of interpreting the parse as a form of dependency structure. Each parent-child relationship in the parse is interpreted as a dependency from the head word of the child to the head word of the parent. Two such relationships are the same if their words are the same. This measure is more closely related to the output of the SSN parser and the PSR model, and may be more appropriate for some applications.

Given that both PCFGs produce no parse for about half of the sentences, it is no surprise that the SSN parser achieves about twice the recall of both PCFGs on both constituents and parent-child relationships (although it is interesting that the SRN actually performs more than twice as well). Thus we also compute these performance figures for the subset of sentences which are parsed by each PCFG, as discussed below. However restricting attention to the parsed subset will not change the PCFGs' precision figures. Just as the recall figures are particularly low, the precision figures are improved due to the fact that the PCFGs are not outputting anything for those sentences which are particularly hard for them (i.e. the sentences they cannot parse). Even so, the SSN does about 10% better than either PCFG on both constituent precision and parent-child precision.

[12]It would be possible to use methods for choosing partial parses, and thus improve the recall results in table 1 (at the cost of precision). Instead we choose to report results on the parsed subset, since this is sufficient to make our point.

Table 2 shows the performance of the PCFGs and SSN on the subset of test sentences parsed by each PCFG. Note that these figures are biased in favour of the PCFGs, since we are excluding only those sentences which are difficult for each PCFG, as determined by the results on the testing set itself. Even so, the SSN outperforms each PCFG under every measure. In fact, under every measure the SSN's performance on the entire testing set is better than the PCFGs' performance on the subset of sentences which they parse.

Given the large difference between the linguistic assumptions embodied in the PCFGs and the SSN parser, we need to address the possibility that the better performance of the SRN parser on this corpus is due to its linguistic assumptions and not due to the SSN architecture. The poor performance of both PSR models clearly demonstrates this. The PSR model was designed to follow the linguistic assumptions embodied in the SSN parser as closely as possible, only imposing additional independence assumptions to the extent that they were required to get sufficient counts for estimating the model's probabilities. Nonetheless, both PSR models do much worse than the PCFGs, even on the parent-child relationships, which are directly related to the parameters of the PSR model. The only exception is comparing against the recall results of the PCFG models including their unparsed sentences, but the recall results of the PCFG models on the parsed subsets indicates that this is simple an artifact of the low coverage of the PCFGs. Thus the better performance of the SSN parser cannot be due to its linguistic assumptions alone. It must be due to the SSN architecture's ability to handle sparse data without the need to impose strong independence assumptions.

One final thing to notice about these results is that there is not a big difference between the results of the SRN parser on the full testing set and the results on the subsets which are parsed by each PCFG. There is a small improvement, reflecting the fact that the PCFGs fail on the more difficult sentences. However the lack of any big improvement is a further demonstration that the SRN is not simply returning parses for every sentence because it is defined in such a way that it must do so. The SSN is making good guesses, even on the difficult sentences. This demonstrates the robustness of the SSN parser in the face of sparse training data and the resulting novelty of testing cases.

5 Conclusion

The good performance of the Simple Synchrony Network parser despite being trained on a small training set demonstrates that SSN parsers are good at handling sparse data. In comparison with Probabilistic Context Free Grammars trained on the same data, the SSN parser not only returns parses for twice as many sentences, its performance on the full testing set is even better than the performance of the PCFGs on the subset of sentences which they parse.

By demonstrating SSNs' ability to handle sparse data we have in fact shown that SSNs generalise from training data to testing data in a linguistically appropriate way. The poor performance of the PSR model shows that this is not simply due to clever linguistic assumptions embodied in the particular parser used. This generalisation performance is due to SSNs' ability to generalise across constituents as well as across sequence positions, plus the ability of neural networks in general to learn what input features are important as well as what they imply about the output. The resulting robustness makes SSNs appropriate for a wide variety of parsing applications, particularly when a small amount of data is available or there is a large amount of variability in the input.

References

[1] Eugene Charniak. Statistical techniques for natural language parsing. *AI Magazine*, 1997.

[2] Michael Collins. *Head-Driven Statistical Models for Natural Language Parsing.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1999.

[3] Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

[4] R. Garside, G. Leech, and G. Sampson (eds). *The Computational Analysis of English: a corpus-based approach.* Longman Group UK Limited, 1987.

[5] James Henderson. *Description Based Parsing in a Connectionist Network.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1994. Technical Report MS-CIS-94-46.

[6] James Henderson and Peter Lane. A connectionist architecture for learning to parse. In *Proceedings of COLING-ACL*, pages 531–537, Montreal, Quebec, Canada, 1998.

[7] Peter Lane and James Henderson. Simple synchrony networks: Learning to parse natural language with temporal synchrony variable binding. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 615–620, Skovde, Sweden, 1998.

[8] I. Melčuk. *Dependency Syntax: Theory and Practice.* SUNY Press, 1988.

[9] Adwait Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34:151–175, 1999.

[10] Geoffrey Sampson. *English for the Computer.* Oxford University Press, Oxford, UK, 1995.

[11] Yves Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* PhD thesis, University of Pennsylvania, Philadelphia, PA, 1990.

[12] Lokendra Shastri and Venkat Ajjanagadde. From simple associations to systematic reasoning: A connectionist representation of rules, variables, and dynamic bindings using temporal synchrony. *Behavioral and Brain Sciences*, 16:417–451, 1993.

[13] C. von der Malsburg. The correlation theory of brain function. Technical Report 81-2, Max-Planck-Institute for Biophysical Chemistry, Gottingen, 1981.

A Context-Free Approximation of Head-Driven Phrase Structure Grammar

Bernd Kiefer and Hans-Ulrich Krieger
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{kiefer,krieger}@dfki.de

Abstract
We present a context-free approximation of unification-based grammars, such as HPSG or PATR-II. The theoretical underpinning is established through a least fixpoint construction over a certain monotonic function. In order to reach a finite fixpoint, the concrete implementation can be parameterized in several ways, either by specifying a finite iteration depth, by using different restrictors, or by making the symbols of the CFG more complex adding annotations à la GPSG. We also present several methods that speed up the approximation process and help to limit the size of the resulting CF grammar.

1 Introduction

This paper presents a context-free approximation of unification-based grammars, such as HPSG or PATR-II. The basic idea of our approach is to generalize in a first step the set of all lexicon entries. The resulting structures form equivalence classes, since they abstract from word-specific information, such as FORM or STEM. The abstraction is specified by means of a restrictor (Shieber 1985), the so-called *lexicon restrictor*. The grammar rules are then instantiated via unification, using the abstracted lexicon entries and resulting in derivation trees of depth 1. We apply the *rule restrictor* to each resulting feature structure, which removes all information contained only in the daughters of the rule. Additionally, the restriction gets rid of information that will either lead to infinite growth of the feature structures during derivation or that does not restrict the search space. Why this is crucial is explained in section 4. The restricted feature structures then serve as the basis for the next instantiation step. Again, this gives us feature structures encoding a derivation, and again we are applying the rule restrictor. We proceed with the iteration, until we reach a fixpoint. In parallel, we also generate annotated context-free rules as have been made popular in linguistics by Harman 1963 and are a cornerstone in GPSG (Gazdar et al. 1985). The final result of this process is the set of CF productions.

Although several approaches have been proposed to limit the size of grammatical descriptions or to delay their applicability (cf. Shieber 1985, Kasper and Krieger 1996, Kiefer et al. 1999), it is practically exciting to extract grammars in a weaker formalism that either preserve the generated strings (weak equivalence) or that approximate them through a superset. Kasper et al. 1995 achieve a compilation of a large fragment of HPSG into lexicalized feature-based TAG. This paper even moves to a weaker formalism, viz., context-free grammars.

Approximating an HPSG through a CFG is interesting for the following reason. Assuming that we have a CFG that comes close to an HPSG, we can use the CFG as a cheap filter (running time complexity is $O(n^3)$). The main idea is to use the CFG first and then let the HPSG deterministically replay the derivations of the CFG. The important point here is that one can correlate every CF production with the corresponding HPSG rule. Kasper et al. 1996 describe such an approach for word graph parsing which employs only the relatively unspecific CF backbone of an HPSG-like grammar. Diagne et al. 1995 replaces the CF backbone through a restriction of the original HPSG. This grammar, however, is still an unification-based grammar, since it employs coreference constraints.

The structure of the paper is as follows. In the next section, we recapitulate some basic HPSG terminology. We argue why it is hard to extract a restrictive CFG from a given HPSG due to lexicalization. We also introduce other concepts which we will need later: restrictors and root nodes. After that, section 3 present the theoretical foundation of our method: approximation as fixpoint construction of a certain monotonic function T. Section 4 presents the basic algorithm which exactly implements T. In order to overcome some deficiencies of the basic algorithm, both in terms of the running time and the resulting CFG, we describe useful optimizations in section 5. In section 6, we apply our method to four grammars and discuss the outcome of the approximation. We conclude this article by indicating our next steps.

2 Basic Inventory

Unification-based theories of grammar allow for an elegant integration of different levels of linguistic descriptions in the common framework of typed feature structures (see Carpenter 1992 for an introduction). In Head-Driven Phrase Structure Grammar this assumption is embodied in the fundamental concept of a *sign*; see Pollard and Sag 1994. A sign is a structure incorporating information from all levels of linguistic analysis, such as phonology, syntax, and semantics. This structure specifies interactions between these levels by means of coreferences, indicating the sharing of information. It also describes how the levels constrain each other mutually.

Such a concept of linguistic description is attractive for several reasons: 1. it supports the use of common formalisms and data structures on all linguistic levels, 2. it provides declarative and reversible interface specifications between these levels, 3. all information is simultaneously available, and 4. no procedural interaction between linguistic modules needs to be set up.

Similar approaches, especially for the syntax-semantics interface, have been suggested for all major kinds of unification-based theories, such as LFG or CUG. A crucial property of HPSG is that it is *lexicalized*, meaning that the rules (called rule schemata) are very general (and only a few of them exist) and that the lexicon entries (actually the lexical types) are relatively specific. HPSG shares this feature with other approaches, such as TAG and CG.

Let us now clarify some terminology which is used throughout this paper. In this paper, we interchange the terms *rules*, *rule schemata*, and *ID schemata* in HPSG. The set of all *rules* is depicted by $\mathcal{R}$. An example of a rule is the head-complement schema. When we say *lexicon entry*, we mean a feature structure that is subsumed by *word*. The collection of all *lexicon entries* constitutes the lexicon $\mathcal{L}$. $ar(r)$ denotes the number of daughters of a given rule $r \in \mathcal{R}$ and is called the *arity* of r.

We also need the notion of a *restrictor*, as originally introduced by Shieber 1985. A restrictor is an automaton describing the paths in a feature structure that will remain after *restriction* (the deletion operation). Since HPSG requires all relevant information to be contained in the SYNSEM feature of the mother structure, the unnecessary daughters only increase the size of the overall structure without constraining the search space. Due to the Locality Principle of HPSG, they can therefore be legally removed in fully instantiated items (see Kiefer et al. 1999). Other portions of a feature structure will also be deleted during the iteration process. In section 4, we will describe why this is necessary. We employ two different restrictors in our approach: the *lexicon restrictor* L and the *rule restrictor* R.

Finally, we introduce the term *root node*. These are constraints specifying well-formedness conditions for other feature structures to be legal utterances/phrases (e.g., empty SUBCAT list). Root nodes are also feature structures and checking well-formedness is achieved via unification. The set of all root nodes is depicted by $\mathcal{N}$.

As we said in the introduction, we not merely produce CF rules employing flat (non-)terminal symbols, but instead enrich them with other information as is known from GPSG. This information directly comes from the feature structure and a user is requested to state this information in terms of an ordered set of paths $\mathcal{P}$, the so-called *relevant paths*. The values under these paths are exactly the annotations associated with the (non-)terminal symbols. Usually, one select those paths which restrict HPSG derivations the most, hence a resulting CF grammar becomes more restrictive. Kiefer et al. 1999 describe how such paths can be obtained with respect to a training corpus. This set of relevant paths is used both for the mother as well as for every daughter of an instantiated rule.

In the following, we make some minor simplifications to the original HPSG framework. (1) We do not distinguish between rules and principles. Instead, we assume that each rule has already inherited the principles which can be applied to it via unification. Such a strategy can often be found in implemented systems. (2) We do not pay attention to LP constraints as they are used in HPSG. The idea here is to hard-wire them in the rules, using the specialized attribute ARGS (see section 6 for a description). This technique is commonly used in many systems. (3) We do not take into account relational or set constraints, such as set membership or set difference. This information is clearly missing in the CFG. We note here that crucial relational constraints such as *append* as used in the subcategorization principle of HPSG must be expressed differently; in case of *append* either by employing the well-known *difference list* representation (good!) or by using a recursive type constraint à la Aït-Kaci. Again, many systems such as $\mathcal{TDL}$ or LKB use exactly the diff list representation. For instance, the English Verb*mobil* grammar (see section 6.4) as well as CSLI's LinGO grammar circumvent *append* in favor of diff lists.

3 Approximation as Fixpoint Construction

The basic inventory we have introduced in the last section now comes into play in order to formalize our idea as stated in the introduction. Let $\mathcal{L}$ be the set of all lexicon entries, $\mathcal{R}$ the set of all rules (rule schemata), L the lexicon restrictor, and R the rule restrictor. We can view a rule $r \in \mathcal{R}$, with $ar(r) = n$ as an n-ary function, mapping n-tuples of feature structures into a single feature structure:

$$r : \mathcal{FS}^n \longmapsto \mathcal{FS} \tag{1}$$

By instantiating only a single daughter, we obtain again a function (*currying*), i.e., $r(fs) = r'$ such that $ar(r') = n - 1$ ($fs \in \mathcal{FS}$). Similarly, we can view the lexicon and the rule restrictor as a (unary) function:

$$L/R : \mathcal{FS} \longmapsto \mathcal{FS} \tag{2}$$

During each iteration step i, we (inductively) obtain the following set T_i of feature structures:

$$T_0 := \bigcup_{l \in \mathcal{L}} L(l) \tag{3}$$

$$T_{i+1} := \bigcup_{n=0}^{i} \bigcup_{r \in \mathcal{R}} \bigcup_{t \in T_n^{ar(r)}} R(r(t)) \tag{4}$$

(3) corresponds to the generalization process of the lexicon, whereas (4) exactly describes the $i + 1$ step of the iteration.

More formally in terms of the (upward) ordinal power of a monotonic mapping T (see Lloyd 1987), we define

$$T(S) := S \cup \bigcup_{r \in \mathcal{R}} \bigcup_{s \in S^{ar(r)}} R(r(s)) \tag{5}$$

Mathematically speaking, T itself operates on the power set of our domain of feature structures $\mathcal{FS}$, mapping approximations into finer approximations, i.e.,

$$T : \wp(\mathcal{FS}) \longmapsto \wp(\mathcal{FS}) \tag{6}$$

The idea here is that in the limit, T will exactly enumerate those feature structures which are instantiations of underspecified rule schemata and which have undergone the application of the rule restrictor R. Thus we are interested in sets of feature structures $S \in \wp(\mathcal{FS})$, where

$$T(S) = S \tag{7}$$

since in this case, S has been saturated, and so we can stop our iteration process via T. In this case, we call S a *fixpoint* of T.

Clearly, T is *monotonic*, since for all $S, S' \in \wp(\mathcal{FS})$ with $S \subseteq S'$, we have $T(S) \subseteq T(S')$. In order to show $T(S) \subseteq T(S')$, we use the definition of T:

$$S \cup \bigcup_{r \in \mathcal{R}} \bigcup_{s \in S^{ar(r)}} R(r(s)) \subseteq S' \cup \bigcup_{r \in \mathcal{R}} \bigcup_{s' \in S'^{ar(r)}} R(r(s'))$$

Since $S \subseteq S'$ (assumption), we have

$$\bigcup_{r \in \mathcal{R}} \bigcup_{s \in S^{ar(r)}} R(r(s)) \subseteq \bigcup_{r \in \mathcal{R}} \bigcup_{s' \in S'^{ar(r)}} R(r(s'))$$

Because $S \subseteq S'$, there must exist an S'' s.t. $S \cup S'' = S'$, hence we finally have for every $r \in \mathcal{R}$

$$\bigcup_{s \in S^{ar(r)}} R(r(s)) \subseteq \bigcup_{s \in S^{ar(r)}} R(r(s)) \cup \bigcup_{s'' \in S''^{ar(r)}} R(r(s'')) \subseteq \bigcup_{t \in (S \cup S'')^{ar(r)}} R(r(t)) = \bigcup_{s' \in S'^{ar(r)}} R(r(s'))$$

The approximated context-free grammar $\mathcal{G}$ then is defined as the *least fixpoint* of T:

$$\mathcal{G} := \mathit{lfp}(T) \tag{8}$$

Alternatively, one can define a fixpoint over the sets of annotated CF productions $\langle P_i \rangle_{i \geq 0}$ which are created in parallel with the construction of $\langle T_i \rangle_{i \geq 0}$. Since elements from T_i are much more specific than from P_i, (7) should of course guarantee a much more specific CF grammar. From a practical point of view, however, terminating the iteration process when $P_i = P_{i+1}$ might suffice in most cases, due to the following observation. When we chose a wrong restrictor R during our experiments, we often experimented a dramatic increase of T_i, but a standstill of P_i. The reason for this were growing feature values in elements from T_i which however do not occur as annotations in rules from P_i.

It should now be clear that a finite fixpoint can only be reached if such growing values are eliminated. Furthermore, the more features are deleted by the restrictor, the more general the CFG will be and the sooner the fixpoint will be reached. Since the approximated CFG is defined as the least fixpoint of T, a pre-specified finite iteration depth usually does neither lead to a subset nor a superset of the original HPSG. But even a pre-specified finite iteration depth together with a restrictor that only deletes the daughters makes perfect sense, since in this case the CFG should generate the same sentences as the HPSG up to a certain number of embeddings.

4 The Basic Algorithm

We will now describe the naïve approximation algorithm which can be seen as a direct implementation of the definitional equations (3) and (4), together with the fixpoint termination criterion given by (7). The construction of (new) annotated context-free rules from successful instantiations is done in parallel with the computation of T_i during iteration step i. Several optimizations are discussed later in section 5 that improve the efficiency of the algorithm, but are uninteresting at this stage of presentation.

We start with the description of the top-level function *HPSG2CFG* which initiates the approximation process. In addition to the parameters already presented, we also use a global variable n, the iteration depth, and a local variable P that accumulates annotated context-free rules.

We begin the approximation by first abstracting from the lexicon entries $\mathcal{L}$ with the help of the lexicon restrictor L (line 6 of the algorithm). This constitutes our initial set T_0 (line 7). We note here that an abstracted lexicon entry is not merely added using set union but instead through a more complex operation, depicted by $\cup_{\sqsubseteq}$. We will describe this operation in section 5.

In case that an abstracted entry l is compatible with one of the root nodes from $\mathcal{N}$ (line 8), a start production is generated from l (employing the relevant paths $\mathcal{P}$) and added to the set of productions P (line 9). Again, adding a production does not reduce to a simple set union, but to a more sophisticated operation. We will describe $\cup_{\rightarrow}$ in section 5. Finally, we start the true iteration calling *Iterate* with the necessary parameters.

```
1  $HPSG2CFG(\mathcal{R}, \mathcal{L}, \mathcal{N}, \mathcal{P}, R, L, n) :\Longleftrightarrow$
2   local $T_0, P$;
3   $T_0 := \emptyset$;
4   $P := \emptyset$;
5   for each $l \in \mathcal{L}$
6     $l := L(l)$;
7     $T_0 := T_0 \cup_{\sqsubseteq} \{l\}$;
8     when $Is\text{-}Saturated(l, \mathcal{N})$
9       $P := P \cup_{\rightarrow} \{S \rightarrow Make\text{-}Symbol(l, \mathcal{P})\}$;
10  $Iterate(\mathcal{R}, \mathcal{P}, \mathcal{N}, R, T_0, P, n)$.
```

Iterate first checks whether we have reached a pre-defined iteration depth n specified originally in *HPSG2CFG* (line 13 of the algorithm). If this is the case, we immediately return the annotated CF rules computed so far (line 14). Otherwise, we decrease the depth, since we are going to start a new iteration (line 15). Usually the iteration depth is ∞, hence line 13 will never become true.

After that the instantiation of the rule schemata with rule/lexicon-restricted elements from the previous iteration T_i begins (line 17–23). The instantiation is performed by *Fill-Daughters* which takes into account a single rule r and T_i, returning successful instantiations (line 18). It is explained in section 5.

For each successful instantiation t which represents a feature structure tree of depth 1, we create an annotated CF rule using *Make-Production*. As was the case for *Make-Symbol*, *Make-Production* uses the relevant paths $\mathcal{P}$ for extracting the right annotations (line 19). It also inspects the daughters of t to compute the proper right-hand side of the new CF rule. Again, we are using the special set union operation $\cup_{\rightarrow}$ here.

After this, we apply the rule restrictor to t, since at this point we are allowed to delete the daughters (line 20). Furthermore, the rule restrictor should be carefully tuned (if possible) to get rid of paths whose values would otherwise grow infinitely through the approximation. Critical areas include the semantics (roughly speaking: information under feature CONTENT in HPSG) which is collected through a derivation but does not shrink as is usually the case for valence information. Now to guarantee a *finite* fixpoint (and we are interested in this), such information must be deleted in order to enforce a moderate growth of T_{i+1}. We then add the restricted t to T_{i+1} (line 21) and check whether it is saturated as specified by one of the root nodes from $\mathcal{N}$; if so, we generate a start production for t (lines 22–23).

We finally come to the point where we compare the (restricted) feature structures from the previous iteration T_i with the new ones from T_{i+1}. If both sets contain the same elements, we clearly have reached a fixpoint. In this case we immediately terminate with the set of annotated CF rules P; otherwise, we proceed with the iteration (lines 24–26).

```
11 $Iterate(\mathcal{R},\mathcal{P},\mathcal{N},R,T_i,P,n) :\Longleftrightarrow$
12   local $T_{i+1}$;
13   when $n = 0$
14     return $P$;
15   $n := n - 1$;
16   $T_{i+1} := T_i$;
17   for each $r \in \mathcal{R}$
18     for each $t \in Fill\text{-}Daughters(r,T_i)$ do
19       $P := P \cup_{\rightarrow} \{Make\text{-}Production(t,\mathcal{P})\}$;
20       $t := R(t)$;
21       $T_{i+1} := T_{i+1} \cup_{\sqsubseteq} \{t\}$;
22       when $Is\text{-}Saturated(t,\mathcal{N})$
23         $P := P \cup_{\rightarrow} \{S \rightarrow Make\text{-}Symbol(t,\mathcal{P})\}$;
24   if $T_i = T_{i+1}$
25     then return $P$
26     else $Iterate(\mathcal{R},\mathcal{P},\mathcal{N},R,T_{i+1},P,n)$.
```

5 Implementation Issues and Optimizations

In this section we describe several techniques that speed up the iteration and that help to reduce the size of the resulting annotated context-free grammar.

5.1 Speeding Up the Algorithm

In order to make the basic algorithm work for large-size grammars, we modified it in several ways. The most obvious optimization applies to the function *Fill-Daughters*, where the number of unifications is reduced by avoiding re-computation of combinations of daughters and rules that already have been checked. To do this in a simple way, we split T_i into $T_i \setminus T_{i-1}$ and T_{i-1} and fill a rule with only those permutations of daughters which contain at least one element of $T_i \setminus T_{i-1}$. This guarantees checking of only those configurations which were enabled by the last iteration.

To further reduce the number of unifications, the algorithm could keep partly-filled rules which may be re-used in later iterations. Whether this pays off depends on the number of these items and the cost for copying and memory management vs. the cost for unification.

We also use techniques originally developed to speed up parsing, namely the so-called *rule filter* and the *quick-check method* (see Kiefer et al. 1999 for a detailed description of these techniques). The rule filter precomputes the applicability of rules into each other and thus is able to predict a failing unification using simple and fast table lookup. The quick-check method exploits the fact that unification fails more often at certain points in feature structures than at others. In an off-line stage, we parse a test corpus using a special unifier that records all failures instead of bailing out after the first in order to determine the most prominent failure points. These points constitute the quick-check vector. When executing a unification during approximation, those points are efficiently accessed and checked prior to the rest of the structure.

As it was mentioned in section 4, instead of using set union we use the more elaborate operation $\cup_{\sqsubseteq}$ when adding new feature structure nodes to T_{i+1}. In fact, we add a new node only if it is not subsumed by some node already in the set. To do this efficiently, the quick-check vectors described above are employed here: before performing full feature structure subsumption, we pairwise check the elements of the vectors using type subsumption and only if this succeeds do a full subsumption test. Extending feature structure subsumption by quick-check subsumption definitely pays of: between 95–99% of all failing subsumptions can be detected early. If we add a new node, we also remove all those nodes in T_{i+1} that are subsumed by the new node in order to keep the set small. This does not change the language of the resulting CF grammar because a more general node can be put into at least those daughter positions which can be filled by the more specific one. Consequently, for each production that employs the more specific node, there will be a (possibly) more general production employing the more general node in the same daughter positions.

A further, although orthogonal method to speed up the subsumption check is to compute a hash key from a quick-check vector that preselects sets of feature structures which are likely to be subsumed. I.e., the quick-check vector here serves as a means to partition feature structures from T_{i+1}. The collection of all vectors form a hierarchy (DAG) which is incrementally build up during the iteration process.

The last two methods accelerate different aspects of the subsumption check: the former technique quickly rejects failing subsumptions, whereas the latter determines likely-subsuming structures, i.e., it is worth to have them both.

5.2 Reducing CF Grammar Size

Subsumption can be extended to annotated symbols and CF productions in an obvious way, namely as componentwise subsumption on types and (non-)terminal symbols. We can then remove all those productions which are subsumed by another one from the set of productions P. We indicated this in the algorithm in section 4 by using the special operation $\cup_{\rightarrow}$. Since applicability of the more general production implies the applicability of the specialized one, the language induced by the reduced grammar remains the same.

Rule folding is another method that can drastically decrease the number of CF productions. Assume that we find a set of annotated CF rules that only differ in one slot. Assume further that the set of values for the slot is equal to the set of all possible values for this slot. Hence we can replace these rules by a single rule in which the slot is assigned the most general value. Rule folding also leads to a grammar inducing the same language. Clearly, the analogue to rule folding can also be implemented in the feature structure domain, but is much more complicated.

Since the annotated CF grammar is still a CFG, we can furthermore remove those symbols which will never contribute to a reading. These symbols and the productions using them are called *useless* and there exist fast decidable algorithms that given a grammar produce a weakly equivalent one which does not contain useless productions (e.g., Hopcroft and Ullman 1979, pp. 87).

5.3 Computing the Productions Afterwards

Although the algorithm from section 4 produces annotated CF productions in parallel with instantiated rule schemata (lines 9, 19, and 23), it might be more efficient to generate them afterwards when reaching the fixpoint (or a predefined iteration depth). Since (5) tells us that (more general) elements from iteration step i are always present in step j, $j \geq i$ (remember, T is monotonic), it suffices to generate CF productions simply by instantiating the rule schemata a final time with elements from the last iteration step . Such a strategy can be more efficient than the parallel generation, due to the fact that we do not have to check for rule subsumption and rule folding (recall that elements in T_i are pairwise incomparable).

6 Examples and Results

In our system, grammar rules are encoded as pure feature structures using a special list-valued feature ARGS to define the right-hand side, while the other top-level features belong to the left-hand side. The rules given in the following figures use this encoding too. During the iteration process, this ARGS feature is the only feature that we remove with the rule restrictor in the next three examples. Otherwise, feature structures would grow in every iteration, due to the rising depth of rule applications and consequently the derivation structure. The fourth example describes the transformation of the large English HPSG that is used in Verb*mobil*.

6.1 A Grammar for $a^n b^n$

The first grammar licenses the language $\{a^n b^n \mid n > 0\}$ and is shown together with the final result of the approximation algorithm (reached after four iterations) in figure 1. The set of relevant paths $\mathcal{P}$ is simply given by the CAT feature.

Rules		Lexicon Entries
Rule1	$\begin{bmatrix} \texttt{CAT S} \\ \texttt{ARGS}\langle [\texttt{CAT A}], [\texttt{CAT B}] \rangle \end{bmatrix}$	
Rule2	$\begin{bmatrix} \texttt{CAT GET-B} \\ \texttt{ARGS}\langle [\texttt{CAT A}], [\texttt{CAT S}] \rangle \end{bmatrix}$	$[\texttt{CAT A}]$ $[\texttt{CAT B}]$
Rule3	$\begin{bmatrix} \texttt{CAT S} \\ \texttt{ARGS}\langle [\texttt{CAT GET-B}], [\texttt{CAT B}] \rangle \end{bmatrix}$	

```
s[_] --> Rule1[s]                          s[_] --> Rule3[s]
Rule1[s] --> lex-entry[a] lex-entry[b]     Rule3[s] --> Rule2[get-b] lex-entry[b]
Rule2[get-b] --> lex-entry[a] Rule1[s]     Rule2[get-b] --> lex-entry[a] Rule3[s]
```

Figure 1: Example grammar 1. `s[_]` denotes the start symbol.

6.2 A Grammar With Coreferences

Our second example employs coreferences to show the variation of the CAT feature in the annotated CF rules between the mother and the two daughters. After three iterations, the algorithm terminates with six feature structure nodes and 18 context-free productions (see figure 2).

Rule	Lexicon Entries
$\begin{bmatrix} \texttt{CAT } \boxed{1} \\ \texttt{ARGS}\left\langle \left[\texttt{CAT}\boxed{1}\right], \left[\texttt{CAT}\boxed{1}\right] \right\rangle \end{bmatrix}$	$[\texttt{CAT A}]$ $[\texttt{CAT B}]$ $[\texttt{CAT C}]$

```
Rule[a] --> lex-entry[a] lex-entry[a]   S[_] --> Rule[a]
Rule[b] --> lex-entry[b] lex-entry[b]   S[_] --> Rule[b]
Rule[c] --> lex-entry[c] lex-entry[c]   S[_] --> Rule[c]
Rule[a] --> lex-entry[a] Rule[a]        S[_] --> lex-entry[a]
Rule[b] --> lex-entry[b] Rule[b]        S[_] --> lex-entry[b]
Rule[c] --> lex-entry[c] Rule[c]        S[_] --> lex-entry[c]
Rule[a] --> Rule[a] lex-entry[a]        Rule[a] --> Rule[a] Rule[a]
Rule[b] --> Rule[b] lex-entry[b]        Rule[b] --> Rule[b] Rule[b]
Rule[c] --> Rule[c] lex-entry[c]        Rule[c] --> Rule[c] Rule[c]
```

Figure 2: Coreference variation in grammar 2.

6.3 Shieber's Toy Grammar

The third example is the feature structure equivalent of Shieber's second sample grammar Shieber 1986, pp. 71–76. This grammar uses two underspecified rules for verb phrase construction and is therefore a test case much more in the direction of HPSG than the first two examples (see figure 3). Overall, the grammar consists of three rules, together with 13 lexicon entries. Two of the lexicon entries can be collapsed into a single category.

We made two tests with different path sets, one containing only agreement, the other containing also subcategorization information. After four iterations, both processes stopped with 31 feature structure nodes ($= T_5$). Without subcategorization information, we got 20 context-free productions. However by using also the type and agreement information of subcategorized elements, we obtained 33 productions. It should be noted that only the second set of productions correctly distinguishes

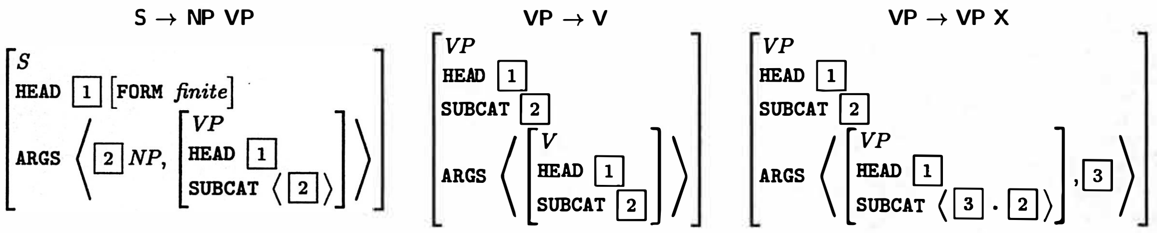

S → NP VP
VP → V
VP → VP X
S
HEAD 1 [FORM finite]
ARGS ⟨ 2 NP, [VP HEAD 1 SUBCAT ⟨2⟩] ⟩
VP
HEAD 1
SUBCAT 2
ARGS ⟨ [V HEAD 1 SUBCAT 2] ⟩
VP
HEAD 1
SUBCAT 2
ARGS ⟨ [VP HEAD 1 SUBCAT ⟨3 . 2⟩], 3 ⟩

Figure 3: Underspecified rules in Shieber's grammar.

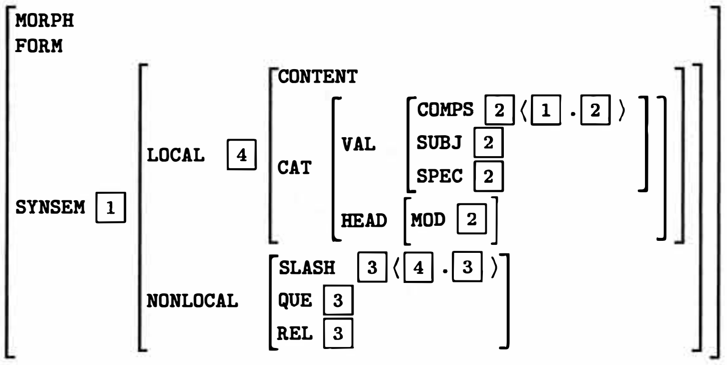


Figure 4: The (slightly simplified) lexicon restrictor used in our experiments.

VP nonterminals subcategorizing for NPs with specific agreement values. Clearly, a more specific CF grammar results in more productions plus more nonterminal symbols.

6.4 CSLI's English Verbmobil Grammar

Our final example employs the English Verb*mobil* grammar, developed at CSLI, Stanford. The grammar consists of 42 rule schemata, 7,473 types, and a lexicon of 4,919 stems from which 10,967 full forms are derived. The lexicon restrictor for the English grammar as shown in figure 4 maps these entries onto 422 lexical abstractions. This restrictor tells us which parts of a feature structure have to be deleted—this is the kind of restrictor which we are usually going to use. We call this a *negative* restrictor, contrary to the *positive* restrictors used in the PATR-II system that specify those parts of a feature structure which will survive after restricting it. Since a restrictor could have reentrance points, one can even define a *recursive* (or cyclic) restrictor to foresee recursive embeddings as is the case in HPSG. The rule restrictor looks quite similar, cutting off additional feature-value pairs, such as DTRS/ARGS.

We made four experiments, using 5, 10, 29, and finally the full 42 rule schemata of the English grammar. Finite fixpoints were reached after at most 12 iteration steps; see figure 5. As we discussed in the last section, it might be the case that more specific feature structures were removed in favor of more general ones. Exactly this happened during the iterations (figure 5). For example, during the approximation of the full grammar, we obtained 8,388 feature structures after iteration step 3, but ended up with 5,522 in step 12. Since the full grammar licensed more constructions than the other three, we guessed that a more general grammar could perhaps result in even less feature structures, due to the generalization process. Indeed, this has happened when comparing the full grammar (5,522) with the grammar consisting of only 29 rules (5,664).

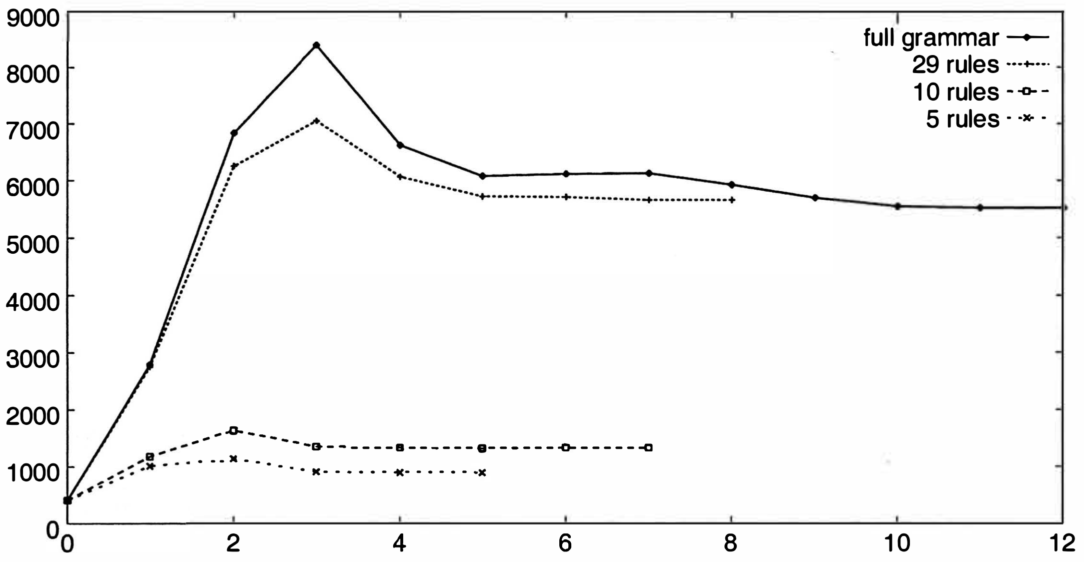


Figure 5: Number of feature structures computed for the four grammars, using the restrictor from figure 4. Finite fixpoints were reached after 5, 7, 8, and 12 iteration steps, resp.

The computation of the fixpoint for the full grammar took about 45 CPU hours on a 300MHz SUN Ultrasparc 2 with Solaris 2.5/Franz Allegro Common Lisp. During the iteration, 22,834,257 full top-level unifications and 71,666,481 subsumptions were performed. The quick-check method, used both during unification and subsumption, leads to a filter rate of 96.1% and 98.8%, resp. This translates into an enormous number of 341,906,443 and 5,018,258,054 quickly failing unification and subsumption operations.

The full grammar consists of 42 rules (14 unary and 28 binary rules). As we indicated in section 5.3, it is possible to compute the CF productions after reaching the fixpoint, viewing full feature structures from the fixpoint as complex CF symbols (one can assign a unique integer to every feature structure). We did this and obtained the following results (a full discussion, including an evaluation of the CF grammar can be found in Kiefer and Krieger 2000). Given the 5,522 feature structures from the fixpoint which are exactly the (non-)terminal symbols of the CF grammar, the 42 rules might lead to $14 \times 5,522 + 28 \times 5,522 \times 5,522 = 853,866,860$ CF productions in the worst case. Our method produces 2,157,445 productions, only 0.25% of all possible ones.

7 Summary

We have presented a context-free approximation of unification-based grammars, such as HPSG or PATR-II. The application of our method to the large-scale English Verb*mobil* grammar has been finished with promising results. We plan to evaluate our method on the German and Japanese HPSGs used in Verb*mobil*. The idea also seems to work with minor modifications for approximating HPSG by Tree Substitution Grammars, TAGs, or other kinds of lexicalized grammars. The substantial difference here comes from the interpretation of the restrictor and the function *Fill-Daughters*. This is currently under evaluation. We also plan to compare the outcome of our approach with the results reported in Torisawa et al. 2000 who also achieved a conversion of the LinGO grammar into a CFG. In section 5.1, we outlined two orthogonal techniques that speed up different aspects of feature structure subsumption, viz., quickly rejecting failing subsumptions and quickly determining likely-subsuming feature structures. These techniques might be worthwhile in other cases where subsumption is a time-critical operation (e.g., during packing of feature structures).

Acknowledgments

We would like to thank our colleagues Tilman Becker, Dan Flickinger, Günter Neumann, and Stephan Oepen for fruitful discussions and two of our reviewers for moral support. Especially Stephan has helped us a lot with the English grammar. The paper has also benefited from a former collaboration with John Carroll and Rob Malouf. Thanks guys. This research was supported by the German Federal Ministry for Education, Science, Research and Technology under grant no. 01 IV 701 V0.

References

Carpenter, B. 1992. *The Logic of Typed Feature Structures.* Tracts in Theoretical Computer Science. Cambridge: Cambridge University Press.

Diagne, A. K., W. Kasper, and H.-U. Krieger. 1995. Distributed Parsing With HPSG Grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies, IWPT-95*, 79–86.

Gazdar, G., E. Klein, G. Pullum, and I. Sag. 1985. *Generalized Phrase Structure Grammar.* Cambridge, MA: Harvard University Press.

Harman, G. 1963. Generative Grammars Without Transformation Rules: A Defence of Phrase Structure. *Language* 39:597–616.

Hopcroft, J. E., and J. D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley.

Kasper, R. T., B. Kiefer, K. Netter, and K. Vijay-Shanker. 1995. Compilation of HPSG to TAG. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, ACL-95*, 92–99.

Kasper, W., and H.-U. Krieger. 1996. Modularizing Codescriptive Grammars for Efficient Parsing. In *Proceedings of the 16th International Conference on Computational Linguistics, COLING-96*, 628–633.

Kasper, W., H.-U. Krieger, J. Spilker, and H. Weber. 1996. From Word Hypotheses to Logical Form: An Efficient Interleaved Approach. In *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, ed. D. Gibbon, 77–88. Berlin: Mouton de Gruyter.

Kiefer, B., and H.-U. Krieger. 2000. Practical Experiments with an HPSG-to-CFG Approximation. Research report.

Kiefer, B., H.-U. Krieger, J. Carroll, and R. Malouf. 1999. A Bag of Useful Techniques for Efficient and Robust Parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, ACL-99*, 473–480.

Lloyd, J. 1987. *Foundations of Logic Programming.* Springer. 2nd edition.

Pollard, C., and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar.* Studies in Contemporary Linguistics. Chicago: University of Chicago Press.

Shieber, S. M. 1985. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, ACL-85*, 145–152.

Shieber, S. M. 1986. *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes, Number 4. Stanford: Center for the Study of Language and Information.

Torisawa, K., K. Nishida, Y. Miyao, and J. Tsujii. 2000. An HPSG Parser with CFG Filtering. *Natural Language Engineering.*

OPTIMAL AMBIGUITY PACKING IN CONTEXT-FREE PARSERS WITH INTERLEAVED UNIFICATION

Alon Lavie
Language Technologies Inst.
Carnegie Mellon University
Pittsburgh, PA 15213
alavie@cs.cmu.edu

Carolyn Penstein Rosé
Learning Res. and Dev. Center
University of Pittsburgh
Pittsburgh, PA 15213
rosecp@pitt.edu

Abstract

Ambiguity packing is a well known technique for enhancing the efficiency of context-free parsers. However, in the case of unification-augmented context-free parsers where parsing is interleaved with feature unification, the propagation of feature structures imposes difficulties on the ability of the parser to effectively perform ambiguity packing. We demonstrate that a clever heuristic for prioritizing the execution order of grammar rules and parsing actions can achieve a high level of ambiguity packing that is provably optimal. We present empirical evaluations of the proposed technique, performed with both a Generalized LR parser and a chart parser, that demonstrate its effectiveness.

1 Introduction

Efficient parsing algorithms for purely context-free grammars have long been known. Most natural language applications, however, require a far more linguistically detailed level of analysis that cannot be supported in a natural way by pure context-free grammars. Unification-based grammar formalisms (such as HPSG), on the other hand, are linguistically well founded, but are difficult to parse efficiently. Unification-augmented context-free grammar formalisms have thus become popular approaches where parsing can be based on an efficient context-free algorithm enhanced to produce linguistically rich representations. Whereas in some formalisms, such as the ANLT Grammar Formalism [2] [3], a context-free "backbone" is compiled from a pure unification grammar, in other formalisms [19] the grammar itself is written as a collection of context-free phrase structure rules augmented with unification constraints that apply to feature structures that are associated with the grammar constituents. When parsing with such grammar formalisms, the goal of the parser is to produce both a *c-structure* – a constituent phrase structure that satisfies the context-free grammar – and an *f-structure* – a feature structure that satisfies the functional unification constraints associated with phrasal constituents.

Regardless of the specific context-free parsing algorithm used, the most common computational approach to performing the dual task of deriving both c-structure and f-structure is an interleaved method, where the unification functional constraints associated with a context-free rule are applied whenever the parser completes a constituent according to the rule. If a "bottom-up" parsing approach is used, this process usually involves applying the unification constraints that augment a grammar rule at the point where the right-hand side (RHS) constituents of the rule have all been identified and are then reduced to the left-hand side (LHS) constituent in the rule. The functional unification constraints are applied to the already existing feature structures of the RHS constituents, resulting in a new feature structure that is associated with the LHS constituent. In the case that any of the specified functional constraints cannot be satisfied, unification fails, and the completed left-hand side (LHS) constituent of the rule is pruned out from further consideration.

The computational cost involved in the construction and manipulation of f-structures during parsing can be substantial. In fact, in some unification-augmented formalisms, parsing is no longer guaranteed to be polynomial time, and in certain cases may even not terminate (when infinitely many f-structures can be associated by the grammar with a given substring of the input) [17]. Additionally, as pointed out by Maxwell and Kaplan [12], interleaved pruning is not the only possible computational strategy for applying the two types of constraints. However, not only is interleaved pruning the most common approach used, but certain grammars, in which the context-free backbone in itself is cyclic, actually rely on interleaved f-structure computation to "break" the cycle and ensure termination. Thus, in this paper, we only consider parsers that use the interleaved method of computation.

Ambiguity packing is a well known technique for enhancing the efficiency of context-free parsers. However, as noted by Briscoe and Carroll [2] [3], when parsing with unification-augmented CFGs, the propagation of feature structures imposes difficulties on the ability of the parser to effectively perform ambiguity packing. In this paper, we demonstrate that a clever heuristic for prioritizing the execution order of grammar rules and parser actions can significantly reduce the problem and achieve high levels of ambiguity packing. We prove that our ordering strategy is in fact optimal in the sense that no other ordering strategy that is based only on the context-free backbone of the grammar can achieve better ambiguity packing. We then present empirical evaluations of the proposed technique, implemented for both a chart parser and a generalized LR parser, that demonstrate that rule prioritization using our heuristic can achieve very significant gains in parser efficiency.

We start out (Section 2) by describing the general technique of ambiguity packing, its utility in parsing, and the problem of applying ambiguity packing in context-free parsers with interleaved unification. Section 3 then describes our heuristic for prioritizing the order in which rules are applied during parsing, in order to ensure effective ambiguity packing. We then analyze the proposed heuristic and prove that it is indeed optimal. In Section 4 we present evaluations performed with both a GLR parser and a chart parser, which demonstrate the effectiveness of our rule prioritization technique. Finally, our conclusions and future directions are discussed in Section 5.

2 Ambiguity Packing in Context Free Parsing

2.1 Ambiguity Packing

Many grammars developed for parsing natural language are highly ambiguous. It is often the case that the number of different parses allowed by a grammar increases rapidly as a function of the length (number of words) of the input, in some cases even exponentially. In order to maintain feasible runtime complexity (usually cubic in the length of the input), many context-free parsing algorithms use a shared parse node representation and rely on performing *Local Ambiguity Packing*. A local ambiguity is a case in which a portion of the input sentence can be parsed and reduced to a particular grammar category (non-terminal) in multiple ways. Local ambiguity packing allows storing these multiple sub-parses in a single common data structure, indexed by a *single* pointer. Any constituents further up in the parse tree can then refer to the set of sub-parses via this single pointer, instead of referring to each of the sub-analyses individually. As a result, an exponential number of parse trees can be succinctly represented, and parsing can still be performed in polynomial time.

Obviously, in order to achieve optimal parsing efficiency, the parsing algorithm must identify all possible local ambiguities and pack them together. Certain context-free parsing algorithms are inherently better at this task than others. Tabular parsing algorithms such as CKY [21] by design synchronize processing in a way that supports easy identification of local ambiguities. On the other hand, as has been pointed out by Billot and Lang [1], the Generalized LR Parsing algorithm (GLR) [18] is not capable of performing full ambiguity packing, due to the fact that stacks that end in different states must be kept distinct. There has been some debate in the parsing community regarding the relative efficiency of GLR and chart parsers, with conflicting evidence in both directions [16], [20], [2], [14]. The relative effectiveness of performing ambiguity packing has not received full attention in this debate, and may in fact account for some of the conflicting evidence[1].

2.2 The Problem: Ambiguity Packing in CFG Parsing with Interleaved Unification

Most context-free parsing algorithms, including GLR and chart parsing, are under-specified with respect to the order in which various parsing actions must be performed. In particular, when pursuing multiple ambiguous analyses, the parsing actions for different analyses may be arbitrarily interleaved. In a chart parser, for example, this non-determinism is manifested via the choice of which key (or inactive edge) among the set of keys waiting in the agenda should next be used for extending active edges. In the case of a GLR parser the non-determinism appears in the choice of which among a set of possible rule reductions should be picked next.

The particular order in which parsing actions are performed determines when exactly alternative analyses of local ambiguities are created and detected. With certain orderings, a new local ambiguity for a constituent may be detected after the constituent has already been further processed (and incorporated into constituents higher up in the parse tree). When parsing with a pure CFG, this does not affect the ability of the parser to perform local ambiguity packing. When a new local ambiguity is detected, it can simply be packed into the previously created node. Any pointers to the packed node will now also point to the new sub-analysis packed with the node. However, when parsing with unification-augmented CFGs, this is not the case. Since feature structures corresponding to the various analyses are propagated up the parse tree to parent nodes, a new local ambiguity requires the creation of a new feature structure that must then be propagated up to any existing parent nodes in the parse tree. In effect, this requires re-executing any parsing actions in which the packed node was involved. In many implementations (for example, the Generalized LR parser/Compiler [19]), to avoid this rather complex task, ambiguity packing is not performed if it is determined that the previous parse node has already been further processed. Instead, the parser creates a new parse node for the newly discovered local ambiguity and processes the new node separately. As a result, the parser's overall local ambiguity packing is less than optimal, with a significant effect on the performance of the parser.

It should be noted that effective local ambiguity packing when parsing with unification-augmented CFGs requires not only a compact representation for the packed c-structure node, but also an effective representation of the associated f-structures. The issue of how to effectively pack ambiguous

[1]For example, van Noord [20] compares left corner and chart parsers that do apply ambiguity packing with a GLR parser without ambiguity packing.

f-structures has received quite a bit of attention in the literature over the last decade [4] [11] [12] [13] [6]. While the two issues are related, we focus here on how to achieve optimal packing of c-structures, so that unification operations do not have to be re-executed due to the later detection of an additional local ambiguity. While there is much to be gained from improved packing on the f-structure level, one should note that effective ambiguity packing on the c-structure level is a necessary pre-condition for efficient parsing, regardless of how well f-structures are packed.

3 The Rule Prioritization Heuristic

In order to ensure effective ambiguity packing in unification-augmented context-free parsers, the situation in which a new local ambiguity is detected after the previous ambiguity has already been further processed must be avoided as much as possible. Ideally, we would like to further process a constituent only after all local ambiguities for the constituent have been detected and packed together into the appropriate node. Our goal was thus to find a computationally inexpensive heuristic that can determine the optimal ordering, or at least closely approximate it. The heuristic that we describe in this section uses only the context-free backbone of the grammar and the spans of constituents in determining what grammar rule (or parsing action) should be "fired" next. Later in the section we prove that this heuristic does in fact achieve the best amount of packing possible, given the information available. In practice, it is easy and fast to compute, and it results in very substantial parser efficiency improvements, as demonstrated by the evaluations presented in the following section. The heuristic was initially developed for the GLR parser, parsing with a unification-augmented CFG. It was then modified to handle the corresponding ordering problem in a chart parser.

3.1 The GLR Ordering Heuristic

In the context of the GLR parser, the heuristic is activated at the point where a decision has to be made between multiple applicable grammar rule reductions. The following example demonstrates the problem and how we would like to address it. Let us assume we have just executed a rule reduction by `[rule0: A --> B C]` that created a parse node for a constituent `A` that spans words $4-7$ of the input. Assume we now have to choose between the following possible rule reductions:

1. Reduce by `[rule1: D --> A]` that would create a constituent `D` spanning words $4-7$ (using the previously just created constituent `A`).
2. Reduce by `[rule2: A --> E F]` that would create a constituent `A` spanning words $4-7$.
3. Reduce by `[rule3: G --> B A]` that would create a constituent `G` spanning words $3-7$ (using the previously just created constituent `A`).

Which rule reduction should be picked next? Since the last rule reduction created a constituent `A` spanning $4-7$, by picking `[rule2]` we can create another constituent `A` spanning $4-7$, that can then potentially be packed with the previous `A`. If we pick `[rule1]` on the other hand, we would further process the previously created constituent `A`. When `[rule2]` is then executed later, the new `A` can no longer be packed with the previous `A`. Thus, it is best to choose to perform the `[rule2]` reduction first.

The first criterion we can use in a heuristic aimed at selecting the desired rule is the span of the constituent that would result from applying the rule. Obviously, we wish to delay applying rules such

```
Input: a set of applicable grammar rule reductions.
Output: A selected grammar rule reduction to be performed next.

Selection Heuristic:

(1) For each potential grammar rule reduction, determine the span
    (start and end positions) and category of the constituent that
    would result from the rule reduction.
(2) Select the rule reduction that is rightmost - has the largest
    start position.
(3) If there is more than one rightmost rule reduction, pick a rule
    reduction that results in a category that is ''least'' according
    to the ''>*'' partial-order.
```

Figure 1: Rule Reduction Selection Heuristic for GLR Parser

as `[rule3]` which process the first `A` until all other possible `A`'s of the same span have been found and packed. Ignoring for the moment the complications arising from unary rules (such as `[rule1: D --> A]` above) and epsilon rules (which consume no input), selecting a rule that is "rightmost" - creating a constituent with the greatest possible starting position, will in fact achieve this goal. This is due to the fact that in the absence of unary and epsilon rules, every constituent must cover some substring of the input, and thus every RHS constituent in a grammar rule must start at a different input position. Thus, the "rightmost" criterion supports the goal of delaying the construction of constituents that extend "further to the left" until all ambiguities of the previous span have been processed, and all potential local ambiguities have been packed.

In the presence of unary grammar rules, however, the "rightmost" criterion is insufficient. Looking again at our example above, both `[rule1]` and `[rule2]` are "rightmost", since both create a constituent that starts in position 4. However, `[rule1]` would further process the existing `A` constituent before `[rule2]` detects a local ambiguity. The problem here is that the application of unary rules does not consume an additional input token. We therefore require a more sophisticated measure that can model the dependencies between constituents in the presence of unary rules. To do this, we use the context-free backbone of the grammar, and define the following partial-order relation "$\geq$" between constituent non-terminals:

- For every unary rule `A --> B` in the grammar G, $A \geq B$.
- We compute "$\geq^*$" - the transitive closure of "$\geq$"

We can now extend our heuristic to use the partial-order information. The idea is not to perform a unary reduction resulting in a constituent `B` when there is an alternative "rightmost" reduction that produces a constituent `A` where $B \geq^* A$. The resulting GLR parser heuristic can be seen in Figure 1.

The partial-order "$\geq^*$" can easily be pre-compiled from the grammar. Note that it is theoretically possible that for particular non-terminals `A` and `B`, both $A \geq^* B$ and $B \geq^* A$. This implies that the context-free backbone of the grammar contains a cycle. With unification-augmented CFGs, this is indeed possible, since the unification equations augmenting the grammar rules may resolve the cycle.

In such cases, the above heuristic may encounter situations where there is no unique rule that is minimal according to the partial-order. If this occurs, we pick one of the "least" categories randomly. This may in fact result in sub-optimal packing, but only full execution of the unification operations can in fact correctly decide which rule should be picked in such cases. In practice, the computational cost of such a test would most likely far exceed its benefits.

3.2 Handling Epsilon Rules

The case that the grammar contains epsilon rules requires some additional attention. In this case, the premise that every RHS constituent of a grammar rule starts at a different position (i.e. consumes input) no longer holds. Consequently, the "rightmost" criterion no longer ensures that a rule that includes a constituent A on the RHS will not fire until all possible local ambiguities of A have been processed and packed. It is useful to note, however, that the problem is in fact similar to that of unary rules, and can be treated quite similarly as well. We first find all "nullable" non-terminals in the grammar G - the set of all non-terminals that can derive the empty string. A well known algorithm for detecting nullable non-terminals can be found in Hopcroft and Ullman [5]. We denote by $A \in EP(G)$ that A is nullable. We now modify the partial-order relation defined above to handle the case of nullable categories. We define "$\geq_\epsilon^*$" as follows:

1. if $A \geq^* B$ then $A \geq_\epsilon^* B$
2. for every rule $A \rightarrow B_1\ B_2\ \cdots B_k$ in G, check if $B_i \in EP(G)$ (for $1 \leq i \leq k$). If the rule is such that at most *one* $B_i \not\in EP(G)$, then for all $1 \leq i \leq k$, $A \gneq B_i$
3. we compute the transitive closure of $\geq_\epsilon^*$

We then replace the "$\geq^*$" relation in our ordering heuristic with the extended "$\geq_\epsilon^*$" relation. Intuitively, the idea is to identify grammar rules that are not unary, but which when "fired" will not construct a constituent that extends further than the RHS constituents. This is the case whenever the LHS constituent derives the empty string, or else all but one of the RHS constituents derive the empty string. We extend the partial-order so that the RHS constituents in such rules are "lesser" in the order than the LHS constituent. This ensures that such rules will not be selected until all local ambiguities of the RHS constituents have been processed and packed.

3.3 Proof of Optimality

We now argue that the complete ordering heuristic that uses the "rightmost" and "least" criteria as defined above is in fact optimal in the sense that it achieves the maximal ambiguity packing possible given only the context-free backbone of the grammar as available information. Let us assume to the contrary that the heuristic does not result in optimal packing. This implies that a constituent A of span (i, j) was created, that a rule that contains A on the RHS was selected by the heuristic, executed, and created a LHS constituent B, and that subsequently, another rule with a LHS of A was executed, creating a new A of span (i, j) that can no longer be packed with the previous A. B must also be of span (i, j), since otherwise it could not have been chosen due to the "rightmost" criterion. At the time the rule creating B was selected by the heuristic, B must have satisfied the "least" criterion. If the second A is created as a result of processing the first A via a series of rule applications, then the grammar is in fact cyclic. As mentioned earlier, while our heuristic is not guaranteed to be optimal in such cases, any better heuristic would require using f-structure unification information. So we assume that the second A constituent was created via a series of rule applications that did not involve the first A. We look at the sequence of constituents $X_1, X_2, \cdots A$ created in the series of rule applications that resulted in the second A after the first A had already been created. All must have span (i, j), since otherwise the second A cannot have a span of (i, j). At least one of these rule applications must have been a possible candidate at the point in time that the rule using the first A and creating B was chosen and executed. However, by the definition of $\geq_\epsilon^*$, for each of the X_i, $A \geq_\epsilon^* X_i$, and since B is created by applying a rule that uses the first A, we also have that $B \geq_\epsilon^* A$. By the transitive closure property of $\geq_\epsilon^*$ we thus have that for all of the X_i, $B \geq_\epsilon^* X_i$. Thus, at the point in time where the rule creating B was fired, B was *not* minimal according to the "least" criterion, and we derive a contradiction. We therefore conclude that the heuristic is in fact optimal.

3.4 The Chart Parser Ordering Heuristic

In order to achieve effective ambiguity packing in the case of the chart parser, we require that the parser process the input strictly left-to-right, in order to synchronize the creation and processing of constituents with similar spans. While chart parsers in general do not require such ordered processing, many chart-based parsers are in fact left-to-right, and this is usually not a burdensome restriction. Our rule reordering heuristic can then be added to the parsing algorithm in a straightforward way. In the case of the chart parser, rather than directly reordering reductions, the reordering heuristic modifies the order in which active edges are extended. The idea is that the most efficient way to extend active edges is to ensure that active edges like $[A \rightarrow \ldots \bullet C \ldots]$ are not extended over inactive edges of category C until all possible inactive edges of category C starting in the appropriate position in the chart have been inserted and packed. The same criteria of "rightmost" and "least" are applied in deciding which active edge should be selected next for extension.

4 Empirical Evaluations and Discussion

To evaluate the effectiveness of our rule selection heuristic we conducted empirical test runs with both a GLR parser and a chart parser. Both parsers are designed to support the same LFG style unification grammar framework. The two parsers and the grammar formalism are briefly described below. Both parsers also have robust versions that are designed to overcome input that is not completely grammatical due to disfluencies or lack of grammar coverage. The robust parsers can skip over words or segments of the input that cannot be incorporated into a grammatical structure. When operated in robust mode, the skipping of words and segments introduces a significantly greater level of local ambiguity. In many cases, a portion of the input sentence may be reduced to a non-terminal symbol in many different ways, when considering different subsets of the input that may be skipped. Thus, efficient runtime performance of the robust versions of the parsers is even more dependent on effective local ambiguity packing. We therefore conducted evaluations with the two parsers in both robust and non-robust modes, in order to quantify the effect of the rule selection heuristic under both scenarios.

All of the described experiments were conducted on a common test set of 520 sentences from the JANUS English Scheduling Task [10], using a general English syntactic grammar developed at Carnegie Mellon University. The grammar has 412 rules and 71 non-terminals, and produces a full predicate-argument structure analysis in the form of a feature structure. For the GLR parser, the

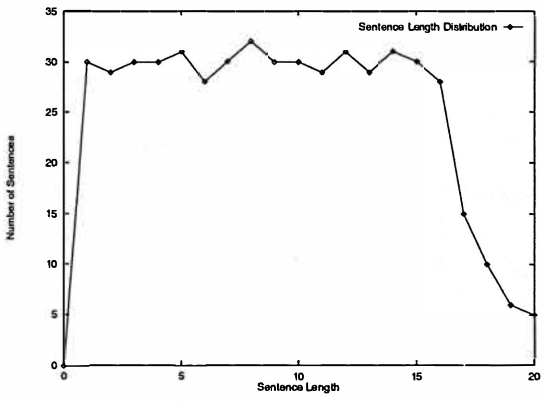


Figure 2: **Distribution of evaluation set sentence lengths**

grammar compiles into an SLR parsing table with 628 states and 8822 parsing actions. Figure 2 shows the distribution of sentence lengths in the evaluation set.

4.1 The GLR Parser

The Generalized LR Parser/Compiler [19] is a unification based practical natural language analysis system that was designed around the GLR parsing algorithm at the Center for Machine Translation at Carnegie Mellon University. The system supports grammatical specification in an LFG framework, that consists of context-free grammar rules augmented with feature bundles that are associated with the non-terminals of the rules. Feature structure computation is, for the most part, specified and implemented via unification operations. This allows the grammar to constrain the applicability of context-free rules. A reduction by a context-free rule succeeds only if the associated feature structure unification is successful as well. The Generalized LR Parser/Compiler is implemented in Common Lisp, and has been used as the analysis component of several different projects at the Center for Machine Translation at CMU in the course of the last decade.

GLR* [7],[9],[8], the robust version of the parser, was constructed as an extended version of the unification-based Generalized LR Parser/Compiler. The parser skips parts of the utterance that it cannot incorporate into a well-formed sentence structure. Thus it is well-suited to domains in which non-grammaticality is common. The parser conducts a search for the maximal subset of the original input that is covered by the grammar. This is done using a beam search heuristic that limits the combinations of skipped words considered by the parser, and ensures that it operates within feasible time and space bounds. The GLR* parser also includes several tools designed to address the particular difficulties of parsing spontaneous speech. These include a statistical disambiguation module and a collection of parse evaluation measures which are combined into an integrated heuristic for evaluating and ranking the parses produced by the parser.

4.2 The LCFlex Parser

The LCFlex parser [15] is a recently developed robust left corner chart parser designed to incorporate the flexibility of GLR* within a more efficient parsing architecture. Its left corner chart parsing algorithm is similar to that described by Carroll in [3]. Thus, it makes use both of bottom-up predictions based on newly created inactive edges as well as top-down predictions based on active

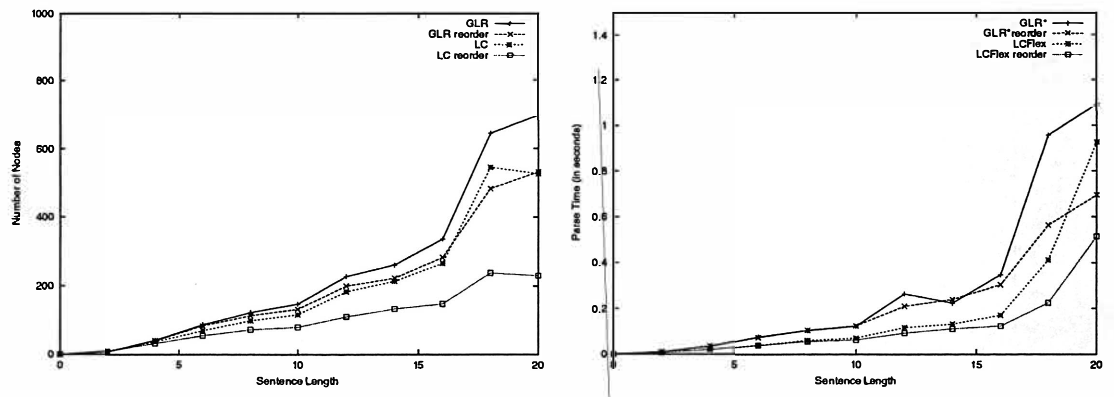


Figure 3: **Non-robust Parsers: number of parse nodes and parse time as a function of sentence length**

edges bordering on those edges. It utilizes the same grammar formalism used in GLR*, described above. The context free backbone within the GLR grammar formalism allows for efficient left corner predictions using a pre-compiled left corner prediction table, such as that described by Van Noord in [20]. It incorporates similar statistical disambiguation and scoring techniques to those used in GLR*, as described in [8].

4.3 Evaluation Results

We first ran both the GLR and the LC parsers in non-robust mode (with no word skipping allowed), once without the rule reordering heuristic and once with the heuristic. Figure 3 (left) shows a plot of the average number of created parse nodes as a function of sentence length. For both parsers, the total number of nodes created when rule reordering is applied significantly decreases, especially with longer sentences. This is a direct result of the increased level of ambiguity packing performed by the parsers when rule reordering is applied. For the LC parser, the savings are quite dramatic. For example, for sentences of length 12, the average relative reduction in number of parse nodes with reordering was about 12% for the GLR parser and about 40% for the LC parser. As can be expected, this relative reduction rate appears to grow as a function of sentence length, due to increasing levels of ambiguity. Figure 3 (right) shows a plot of the average parse times as a function of sentence length. As can be seen, the LC parser is for the most part faster than the GLR parser, regardless of whether or not rule reordering is applied. However, for both parsers, the improved ambiguity packing with rule reordering results in a significant reduction in parsing time which increases with sentence length, due to increasing levels of ambiguity. For sentences of length 12, the time savings were about 21% for both parsers.

We then re-ran the experiment with the robust versions of the two parsers. The GLR* parser was run with a search beam of 30, which was determined in various previous experiments to be a reasonable setting for achieving good parsing results within feasible parsing times. The LCFLEX parser was then run in a way that simulated the same skipping behavior of GLR* (i.e. the exact same word skipping possibilities were considered by the parser). Figure 4 (left) shows a plot of the average number of created parse nodes as a function of sentence length. Once again, the total number of nodes created when rule reordering is applied significantly decreased. For GLR*, the reduction in number of nodes

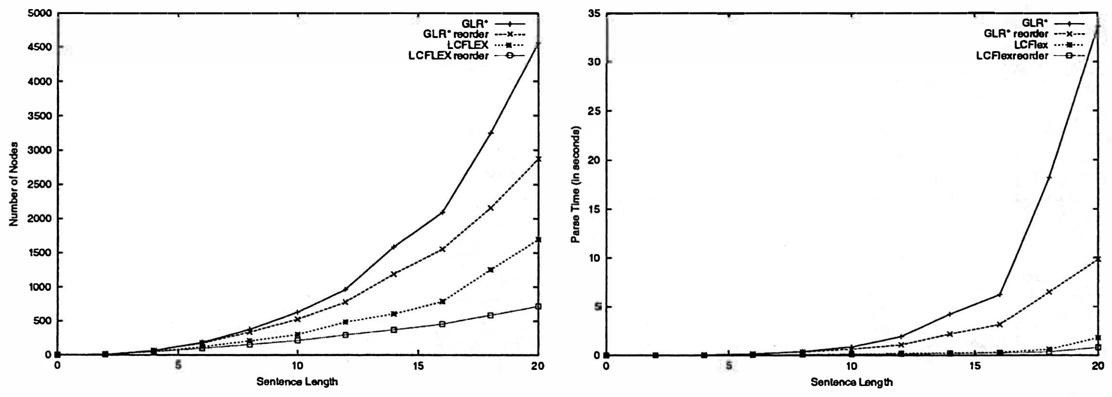


Figure 4: **Robust Parsers: number of parse nodes and parse time as a function of sentence length**

was more significant than the non-robust case, while for LCFLEX it appears to be about the same. For example, for sentences of length 12, the average relative reduction in number of parse nodes with reordering was about 19% for the GLR* parser and about 39% for the LCFLEX parser. Figure 4 (right) shows a plot of the average parse times as a function of sentence length. As can be seen, in robust mode, the LCFLEX parser is much faster than GLR*. As expected, runtimes when rule reordering is applied significantly decreased. For GLR*, the decrease is more pronounced than in the none-robust experiment, while for LCFLEX, it was about the same. For sentences of length 12, the time savings were about 44% for GLR* and about 21% for LCFLEX.

5 Conclusions and Future Directions

The efficiency of context-free parsers for parsing natural languages crucially depends on effective ambiguity packing. As demonstrated by our experimental results presented in this paper, when parsing with CFGs with interleaved unification, it is vital to execute parsing actions in an order that allows detecting local ambiguities in a synchronous way, before their constituents are further processed. Our grammar rule prioritization heuristic orders the parsing actions in a way that achieves the best possible ambiguity packing using only the context-free backbone of the grammar and the constituent spans as input to the heuristic. Our evaluations demonstrated that this indeed results in a very substantial improvement in parser efficiency, particularly so when incorporated into robust versions of the parsers, where local ambiguities abound.

The focus of most of our current and planned future research is on efficient parsing within the context of the robust LCFLEX parser. In particular, we are interested in investigating the interdependence between the parser's robustness capabilities, its search strategy, and its disambiguation and parse selection heuristics. Effective ambiguity packing will continue to be an important factor in this investigation. We plan to further explore strategies other than interleaved pruning for efficiently applying both phrasal and functional constraints, while using the existing grammar framework and broad-coverage grammars that are at our disposal.

References

[1] S. Billot and B. Lang. The Structure of Shared Forests in Ambiguous Parsing. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics (ACL'89)*, Vancouver, BC, Canada, June 1989.

[2] T. Briscoe and J. Carroll. Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. *Computational Linguistics*, 19(1):25–59, 1993.

[3] J. A. Carroll. *Practical Unification-Based Parsing of Natural Language*. PhD thesis, University of Cambridge, Cambridge, UK, October 1993. Computer Laboratory Technical Report 314.

[4] A. Eisele and J. Dorre. Unification of Disjunctive Feature Descriptions. In *Proceedings of the 26th Annual Meeting of the ACL (ACL-88)*, Buffalo, NY, 1988.

[5] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation*, chapter 4.4, page 90. Addison-Wesley, 1979.

[6] B. Kiefer, H.-U. Krieger, J. Carroll, and Malouf R. A Bag of Useful Techniques for Efficient and Robust Parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pages 473–480, College Park, MD, Jun 1999.

[7] A. Lavie. An Integrated Heuristic Scheme for Partial Parse Evaluation. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics (ACL-94)*, pages 316–318, Las Cruces, New Mexico, June 1994.

[8] A. Lavie. *GLR*: A Robust Grammar-Focused parser for Spontaneously Spoken Language*. PhD thesis, School of Computer Science, Carnegie Mellon University, May 1996. Technical Report CMU-CS-96-126.

[9] A. Lavie. GLR*: A Robust Parser for Spontaneously Spoken Language. In *Proceedings of ESSLLI-96 Workshop on Robust Parsing*, 1996.

[10] A Lavie, D. Gates, M. Gavalda, L. Mayfield, A. Waibel, and L. Levin. Multi-lingual Translation of Spontaneously Spoken Language in a Limited Domain. In *Proceedings of International Conference on Computational Linguistics (COLING'96)*, pages 442–447, Copenhagen, Denmark, 1996.

[11] J. T. Maxwell and R. M. Kaplan. A Method for Disjunctive Constraint Satisfaction. In M. Tomita, editor, *Current Issues in Parsing Technology*. Kluwer Academic Press, 1991.

[12] J. T. Maxwell and R. M. Kaplan. The Interface between Phrasal and Functional Constraints. *Computational Linguistics*, 19(4):571–590, December 1993.

[13] Y. Miyao. Packing of Feature Structures for Efficient Unification of Disjunctive Feature Structures. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics (ACL'99)*, pages 579–584, College Park, MD, Jun 1999.

[14] M-J. Nederhof and G. Satta. Efficient Tabular LR Parsing. In *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, June 1996.

[15] C. P. Rosé and A. Lavie. Balancing Robustness and Efficiency in Unification-augmented Context-Free Parsers for Large Practical Applications. In van Noord and Junqua, editors, *Robustness in Language and Speech Technology*, ELSNET. Kluwer Academic Press. To appear.

[16] Y. Schabes. Polynomial Time and Space Shift-Reduce Parsing of Arbitrary Context-free Grammars. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics (ACL'91)*, June 1991.

[17] S. M. Shieber. *An Introduction to Unification-based Approaches to Grammar*. Number 4 in CSLI Lecture Notes. CSLI Stanford Univerity, Stanford, CA, 1986.

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

[19] M. Tomita. The Generalized LR Parser/Compiler - Version 8.4. In *Proceedings of International Conference on Computational Linguistics (COLING'90)*, pages 59–63, Helsinki, Finland, 1990.

[20] G. van Noord. An Efficient Implementation of the Head-Corner Parser. *Computational Linguistics*, 23(3):425–456, 1997.

[21] D.H Younger. Recognition and Parsing of Context-Free Languages in Time n^3. *Information and Control*, 10(2):189–208, 1967.

EXTENDED PARTIAL PARSING FOR LEXICALIZED TREE GRAMMARS

Patrice Lopez
DFKI GmbH
Stuhlsatzenhausweg 3
D-66123 Saarbrcken
lopez@dfki.de

Abstract
Existing parsing algorithms for Lexicalized Tree Grammars (LTG) formalisms (LTAG, TIG, DTG, ...) are adaptations of algorithms initially dedicated to Context Free Grammars (CFG). They do not really take into account the fact that we do not use context free rules but partial parsing trees that we try to combine. Moreover the lexicalization raises up the important problem of multiplication of structures, a problem which does not exist in CFG. This paper presents parsing techniques for LTG taking into account these two fundamental features. Our approach focuses on robust and pratical purposes. Our parsing algorithm results in more extended partial parsing when the global parsing fails and in an interesting average complexity compared with others bottom-up algorithms.

1 Introduction

Lexicalized Tree Adjoining Grammars (LTAG) [Joshi and Schabes, 1992] have given rise to a lot of interests for the modeling of syntax, in particular thanks to its three "key" properties: the principle of Extended Domain of Locality (EDL), the lexicalization and the representation of recursive phenomena using the operation of adjunction. Since these properties alow LTAG to capture semantic dependencies in elementary trees, the parsing result (the derivation trees) is closed to semantic dependency trees which is a relevant structure for post-parsing processes [Candito and Kahane, 1998].

This paper focuses on robust chart parsing with Lexicalized Tree Grammars. We do not consider here probabilistic approximation but only complete or partial structures obtained with valid rule applications. The classical parsing algorithms for LTAG, for instance CKY-like algorithm [Vijay-Shanker, 1987], Head-Corner [van Noord, 1994] or Earley-like algorithm [Schabes, 1994] focus on the parsing of complete grammatical utterances. We argue that an algorithm dedicated to the parsing of a Lexicalized Tree Grammar can take into account more efficiently EDL and lexicalization in order to (1) obtain beneficial extended partial results if the global parsing fails, (2) decrease the average complexity of the analysis obtained with bottum-up parsers.

The constraints expressed with lexicalized elementary trees are richer than the constraints captured with a set of rewriting context free rules. The *extended domain of locality* is the fact that an elementary tree encodes directly a syntactical substructure view as a partial parsed tree. It allows to define constraints in more than one level of the parsing tree as compared to context free rules and permits to use atomic features. One way to exploit this property during parsing is to consider for instance co-anchors which are often neglected in existing parsing algorithm. Co-anchors encode cooccurrence information which are significant for parsing. Co-anchors are frequently used in the French LTAG grammar for prepositional complements and idioms [Abeillé et al., 1999]. They could also be massively used in elementary trees obtained with Explanation Based Learning (EBL) methods used to speed up parsing.

The *lexicalization* imposes that each elementary tree contains at least one lexical terminal symbol called the *anchor*. This constraint permits to represent in each elementary tree one of the syntactical contexts of a lexical entry. This property is usually exploited during a pre-parsing process which consists in the selection of the sub-set of elementary trees that can anchor at least one of the words of the input sentence. But such a property results also in a lot of duplication of the same sub-structures in the grammar which does not exist in a CFG. Existing parsers usually ignore this drawback and result in a lot of redundant computations.

Considering the syntactic level, the two main differences between parsing a sentence using a formal grammar and a grammar dedicated to natural language are the necessity to take into account the ambiguity of the language and the potential "out of grammar" words and structures. Addressing the problem of ambiguities, efficient results preserving all valid rule applications has been obtained using tabular parsing technics: The result stored in a chart is a shared parse forest [Lang, 1991]. From the point of view of robustness, it is important to be able to implement local analyses, i.e. to try to extract a maximum of information from the utterance (at least all potential constituents) even if the complete parsing failed. Prediction usually speeds up a parser. However as explained for example in [Magerman and Weir, 1992], Earley-style prediction can improve the average case performance but it has serious impacts on robust processing of incomplete and ungrammatical sentences which are very common in natural language systems. The same limit can be observed for LR-style parsing results as [Nederhof, 1998].

As a consequence, we present here a new tabular parsing algorithm for LTAG called *connection driven parsing*. This incremental bottom-up algorithm could be applied to other kind of LTG. Classical top-down predictions could be used to improve the parsing but they would decrease robustness.

2 Connection driven parsing technics

2.1 Lexicalized Tree Grammars

A Tree Adjoining Grammar is specified as a quintuplet $G = (\Sigma, NT, I, A, S)$, where Σ is a set of terminal symbols, NT the disjoint set of nonterminals including the start symbol S, and where I and A are two finite sets of trees called respectively *initial trees* and *auxiliary trees*. The set $I \cup A$ gathers the *elementary trees*. We consider the standard definition of completely Lexicalized TAG in which each tree contains at least a leaf node, called the *anchor*, which corresponds to a word. For the complexity results we note N the maximum number of nodes of an elementary tree, G the size of the set $I \cup A$ and n the lenght of the input string to parse.

We note a substitution node with its category and the mark $\downarrow$, internal nodes without mark, root nodes with $^{\triangle}$ and anchors with the lexical mark $\Diamond$. Finally, we note $*_l$ (resp. $*_r$) the foot node of a left (resp. right) auxiliary tree and $*$ the foot node of a wrapping auxiliary tree. In order to make explanations easier, we will not consider non-adjunction constraints on nodes. A Tree Inserted Grammar (TIG) is a LTG which does not include any wrapping auxiliary tree and supposes that this kind of tree never appears dynamically during the parsing. Using a TIG the worst case complexity of the parsing is decreased to (n^3) as all derivations are equivalent to Context Free derivations [Schabes and Waters, 1995].

2.2 Limits of CF algorithms invariant

One of the most important caracteristic of a parser for natural language is its ability to deliver the richest possible partial parsing results when the complete parsing fails. Diagnostic, repairing and interpretation are much easier when we can exploit informative partial derivation trees. When we use a chart parsing algorithm, the partial results are given by the items which represent a chunk of well recognized words. The nature of a partial result is given by the invariant of the algorithm which caracterizes the properties of each produced item.

In CFG algorithms, the invariants are based on subtree recognition: Each item represents one position in a elementary tree and a portion of the string to parse. The classical invariants impose that the sub-tree dominated by the dotted node associated to an item is completely parsed [Shieber et al., 1995]. Non predictive rules combine complete subtrees into larger ones preserving this invariant.

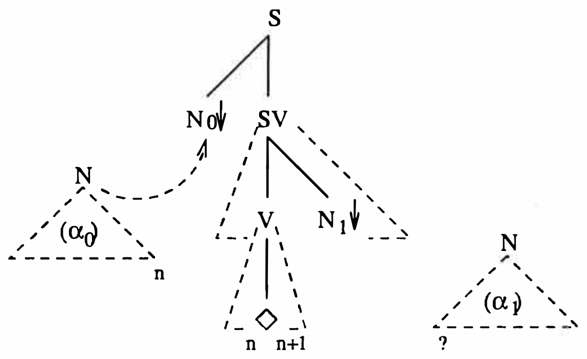


Figure 1: Bottom-up tree walk and subtree invariant.

In figure 1, we indicate different subtrees which have to be completely recognized to allow the processing of the parsing of the elementary tree. Now we suppose that the utterance to be parsed is agrammatical because of an unexpected component on the right (at position marked with a ?). The subtree α_1 on the right is not adjacent to the anchor of the elementary and so the attachment on substitution node N_1 is impossible. The subtree dominated by the node SV can not be complete and the parsing process of this elementary tree must stop, even if the subtree α_0 can be combined by adjacency to the susbtitution node N_0. Considering a bidirectional parsing as a CKY type or Head Driven type [Lavelli and Satta, 1991] [van Noord, 1994] when an unexpected phenomena occurs the classical invariant stops the parsing process on both sides of the current recognized subtree even when it is possible to continue the parsing on one side, resulting in poorer partial results.

Our first proposition is to focus the parsing on recognized islands and not on recognized subtree and to allow real bidirectional extensions. Since tabular technics impose adjacency of items to combine, we will see that we can take into account the topology of the trees to decrease the average complexity with a new level of granularity for a linearized tree called *connected route.*

2.3 Finite States Automata representation of an elementary tree

The existing representations of the parsing process of a elementary tree are dotted tree [Vijay-Shanker, 1987] and dotted rules [Nederhof, 1997]. In both case this representation is constrained by their locality. We propose an alternative representation that allows to express contraints of significant parts of the tree.

The linearization of a tree can be represented with a Finite State Automaton (FSA) as in figure 2. Every tree traversal (left-to-right, bidirectional from an anchor, ...) can be performed on this automaton. The dotted trees used for example in [Schabes, 1994] or [Shieber et al., 1995] are equivalent to the states of these automata.

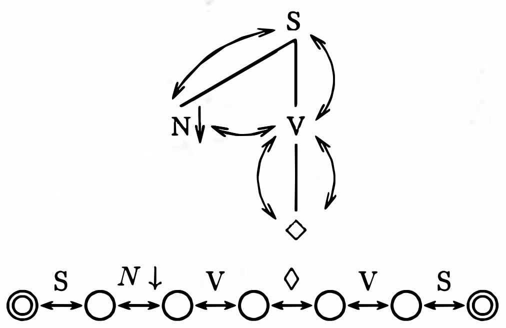


Figure 2: Simple FSA representing an elementary tree for the normal form of an intransive verb.

We consider the following definitions and notations :

- Each automaton transition is annotated with a category of node, each non-leaf node appears twice in the list of transition framing the nodes which it dominates. In order to simplify our explanation the transition is shown by the annotated category.

- Transitions can be bidirectionnal in order to be able to start a bidirectionnal tree walk of a tree starting from any state.

- Considering a direction of transition (left-to-right, right-to-left) the FSA becomes acyclic.

2.4 Parsing invariant and island representation

A set of FSA corresponds to a global representation of the grammar, for the parsing we use a classical local representation called *item*. An item is defined as a 7-tuple of the following form :

```
state:  (  left index, right index,
           left state, right state,
           foot left index,
           foot right index, star state)
```

The two first indices are the limits on the input string of the island (an anchor or consecutive anchors) corresponding to the item. During the initialization, we build an item for each anchor (or co-anchor) present in the input string. An item also stores two states of the same FSA corresponding to the maximal extension of the island on the left and on the right, and *only* if necessary two additional indices for the position of the foot node of a wrapping auxiliary tree and the state *star* corresponding to the node where the current wrapping adjunction have been predicted.

This representation maintains the following invariant: An item of the form $(p, q, \sigma_L, \sigma_R)$ specifies the fact that the linearized tree represented by a FSA Δ is completely parsed between the states σ_L and σ_R of Δ and between the indices p and q. No other attachment on the tree can happen on the nodes located between the anchors p and q-1.

2.5 Connected routes

Considering an automaton representing the linearization of an elementary tree, we can define a connected route as a part of this automaton corresponding to the list of nodes crossed successively until reaching a substitution, a foot node or a root node (included transition) or an anchor (excluded transition). Connected route is an intermediate level of granularity when representing a linearized tree: each elementary (or a derived tree) can be represented as a list of connected routes. Since they can represent frontiers of the constituants, considering connected routes during the parsing permits to take into account the topology of the elementary trees and to locate significative nodes for an attachment.

The main advantage of considering connected routes during the parsing is the following: The connected routes located between two consecutive anchors become useless, because there is no place for any operation, and they can be directly eliminated without considering the states they contain. Such elimination will be performed during the parsing each time anchors get close to each other, so each time that an attachment is performed because, by using classical tabular techniques and linearized trees, the parsing is driven by the connection of anchors. This elimination operation is called *co-anchor reduction* and permits to consider less states (so nodes) during the parsing and so less hypotheses to test.

We use the following additional simplified notations and primitives :

- The connected route passing through the state σ_d is noted Γ_d.
- $next(\Gamma)$ (resp. $previous(\Gamma)$) gives the first state of the connected route after (resp. before) Γ according to a left-to-right automaton walk.
- $next(N)$ (resp. $previous(N)$) gives the state after (resp. before) the transition N.
- $head(\Gamma)$ (resp. $tail(\Gamma)$) gives the first right (resp. left) transition of the leftmost (resp. rightmost) state of the connected route Γ.

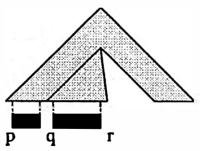
2.6 Inference rules system

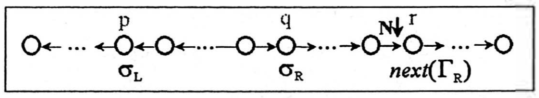
The derivation process can be viewed as inference rules which use and introduce items. The inference rules [Schabes, 1994] have the following meaning: If $item_1$ and $item_2$ are present in the chart and if the conditions are fulfilled then add $item_3$ in the chart *if necessary*:

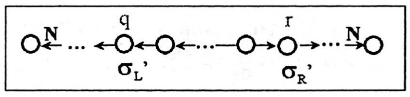
$$\frac{item_1 \qquad item_2}{\text{add} \quad item_3} \quad (\text{ conditions })$$

We present in tables 1 to 5 the different rules of the parser. We illustrate each rule with an abstract tree combination diagram and the state positions on the automata involved in the rule. The first item can be considered as the functor of the rule, for each items we examine successively the nodes on the left and right connected routes. We just have to test then if another adjacent item fulfills the requirements to be the operand of one of the rules. For the bidirectional parsing we just implement the symetric rules. Each rule extends the position on the linearized elementary tree instead of climbing up the *spine*[1] of the elementary tree as in [Lavelli and Satta, 1991] or [van Noord, 1994]. Even if the parsing is stopped on one side of the spine, our algorithm allows to reach the root node on the other side (of course if no agrammaticality occurs on this second side) when the classical algorithms would completely stop the parsing on the both sides.

Rules for substitution

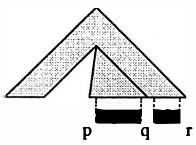


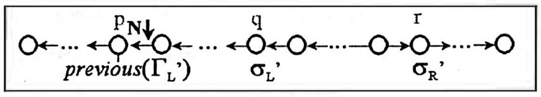


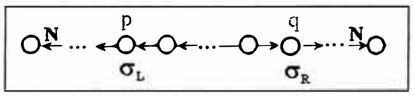


Substitution case 1:

$$\frac{(p,q,\sigma_L,\sigma_R) \qquad (q,r,\sigma_L{}',\sigma_R{}')}{(p,r,\sigma_L,next(\Gamma_R))} \quad \left(\begin{array}{lll} tail(\Gamma_R) = N\downarrow & (\sigma_L{}' \wedge \sigma_R{}') \in \Delta & \Delta \in I \\ head(\Gamma_L{}') = N^{\Delta} & tail(\Gamma_R{}') = N^{\Delta} & \end{array} \right)$$







Substitution case 2:

$$\frac{(p,q,\sigma_L,\sigma_R) \qquad (q,r,\sigma_L{}',\sigma_R{}')}{(p,r,previous(\Gamma_L{}'),\sigma_R{}')} \quad \left(\begin{array}{lll} head(\Gamma_L{}') = N\downarrow & (\sigma_L \wedge \sigma_R) \in \Delta & \Delta \in I \\ head(\Gamma_L) = N^{\Delta} & tail(\Gamma_R) = N^{\Delta} & \end{array} \right)$$

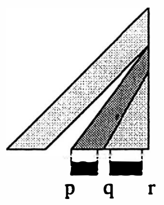
Table 1: Inference rules for substitutions.

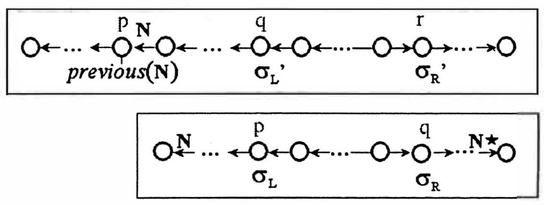
Substitution, left and *right adjunctions* are Context Free derivations: These operations only have to consider adjacent substrings. *Co-anchor reduction* rule allows to combine two islands which belong to the same tree. *End of connected route* rule permits to reach positions at the end of current left and right connected route on the automata in order to improve factorization of tabulated items. If we only consider these operations we can parse a TIG with a worst case complexity in $O(n^3)$ because a wrapping auxiliary tree is never built during the parsing.

The parsing of LTAG must deal with non-continuous strings occuring in *wrapping adjunctions*: A wrapping auxiliary tree can match a pair of non-adjacent strings, one on the left of the foot and one on the right of the foot. The parsing of this operation proceeds in two steps: First the foot node initialization which corresponds to the prediction of an adjunction, then the complete recognition of the sub tree dominated by the internal node where the adjunction occurs. The second step deals with 2 additional indices by items to represent the position of the foot node. Two additional cases are necessary but not represented here because of space limitation: the first one initializes first the root node of the auxiliary tree and then extends the island until the foot node. The last one is triggered only by wrapping auxiliary trees which have no anchor at one side of the spine. It initializes at all possible indices the position of the foot node at the side where there is no anchor. The complete inference rule system can be found in [Lopez, 1999a]. Parsing LTAG increases the worst case time complexity from $O(n^3)$ to $O(n^7)$. This worst case complexity can be decreased to $O(n^6)$ by using additional techniques as suggested in [van Noord, 1994].

[1]The spine of an elementary tree is the path between the main anchor and the root node of the tree.

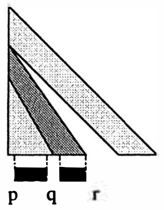
Rules for context free adjunction

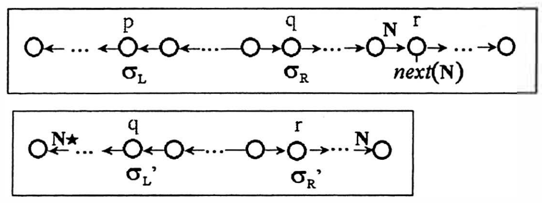




Left adjunction:

$$\frac{(p,q,\sigma_L,\sigma_R) \qquad (q,r,{\sigma_L}',{\sigma_R}')}{(p,r,\mathit{previous}(N),{\sigma_R}')} \quad \begin{pmatrix} \exists(N^{\triangle} \vee N) \in {\Gamma_L}' & (\sigma_L \wedge \sigma_R) \in \Delta \\ \mathit{head}(\Gamma_L) = N^{\triangle} & \mathit{tail}(\Gamma_R) = N{*}_l \end{pmatrix} \quad \Delta \in A$$





Right adjunction:

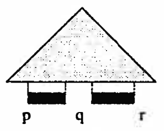
$$\frac{(p,q,\sigma_L,\sigma_R) \qquad (q,r,{\sigma_L}',{\sigma_R}')}{(p,r,\sigma_L,\mathit{next}(N)} \quad \begin{pmatrix} \exists(N^{\triangle} \vee N) \in \Gamma_R & ({\sigma_L}' \wedge {\sigma_R}') \in \Delta \\ \mathit{head}({\Gamma_L}') = N{*}_r & \mathit{tail}({\Gamma_R}') = N^{\triangle} \end{pmatrix} \quad \Delta \in A$$

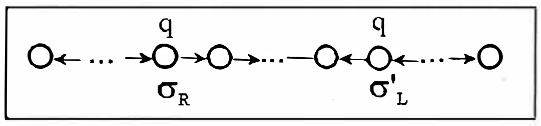
Table 2: Inference rules for context free adjunctions.

2.7 Chart parsing

Items are stored in a chart type data structure, indexed by their indices and combined according to their adjacency, which allows the factorization of items of the chart (tabular techniques). No item can be added that already exists in the chart. We use classical item history that prevents to add in the chart already existing items. The same technique is used to select possible operand items for a rule exploiting mutual exclusion between items corresponding to concurrent lexical hypothesis (same anchors or co-anchors). It allows to obtain a global coherence of an item considering its corresponding tree. We also have used a *subsomption test*, as described in [Satta and Stock, 1989] and suggested for LTAG in [Lavelli and Satta, 1991], in order to limit redundant items due to the bidirectional expansion. Parsing is complete when all possible rules have been executed.

Coanchors reduction rule

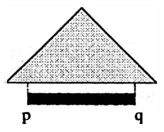


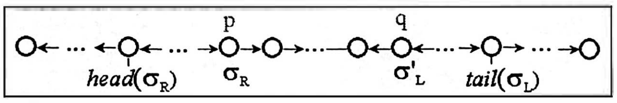


$$\frac{(p,q,\sigma_L,\sigma_R) \qquad (q,r,{\sigma_L}',{\sigma_R}')}{(p,r,\sigma_L,{\sigma_R}')} \quad (\ \Gamma_R = {\Gamma_L}'\ )$$

Table 3: Inference rule for co-anchors reduction.

End of connected route rule

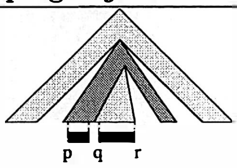




$$\frac{(p,q,\sigma_L,\sigma_R)}{(p,q,head(\sigma_L),tail(\sigma_R))}$$

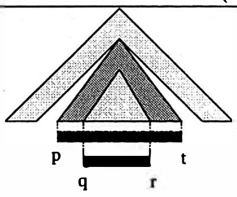
Table 4: Inference rule for end of connected route.

Wrapping adjunction case 1



Foot node initialization:

$$\frac{(p,q,\sigma_L,\sigma_R) \qquad (q,r,\sigma_L{}',\sigma_R{}')}{(p,r,\sigma_L,next(\Gamma_R),q,r,N_0)} \quad \left(\begin{array}{c} \exists N_0/(N_0 \in \Gamma'_L) \wedge (N_0 \in \Gamma'_R) \quad (\sigma_L \wedge \sigma_R) \in \Delta \\ \Delta \in A \quad tail(\Gamma_R) = N* \quad head(\Gamma_L) = N^{\triangle} \end{array} \right)$$



Wrapping adjunction:

$$\frac{(p,t,\sigma_L,\sigma_R,q,r,N_0) \qquad (q,r,\sigma_L{}',\sigma_R{}',u,v)}{(p,t,previous(N_0),next(N_0),u,v)} \quad \left(\exists N_0/(N_0 \in \Gamma'_L) \wedge (N_0 \in \Gamma'_R) \right)$$

Table 5: Inference rules for wrapping adjunction case 1.

3 Capting CF factorizations: LTG sharing

Lexicalization raises the problem of multiplication of the same substructures which can be serious. In CFG the same rule can be used for all possible parsing trees which contain the corresponding substructure, but in LTG this substructure is duplicated. Considering classical linguistics choices, polystructures (for example *to speak to...*, *speak about...*, *to speak to ... about...*) are very common. The corresponding elementary trees must share common substructures and therefore do not cost as much as an independant elementary tree for each. As chart parsing can reduce the complexity in n exploiting factorisation of chart items by their positions on the string and on the elementary trees, we can reduce the average complexity in G using compaction of common substructures because such common structures of elementery trees imply common actions of the parser. For example [Nederhof, 1998] adresses this problem for LR parsing algorithm.

[Evans and Weir, 1998] shares different substructures of elementary trees using FSA and classical minimalization techniques. As presented in [Evans and Weir, 1997], the authors use automata corresponding to one particular traversal of the trees. We use similar techniques to share here linearized structures between different elementary trees. The main differences are that, since it represents elementary trees for any kind of tree walk, this FSA does not impose a specific strategy during the parsing and makes it possible to exploit the granularity of connected route.

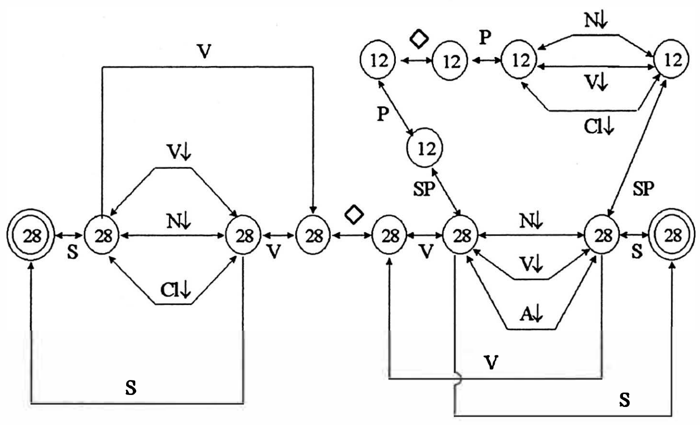


Figure 3: FSA representing 28 elementary trees corresponding to some intransive and transitive contexts.

Figure 3 gives an example of a minimalized FSA which allows to share 28 elementary trees. The number in a state indicates how many trees pass through that state. Table 6 gives the resulting compaction statistics.

automaton	no. of trees	no. of states	no. of transitions	trees per state
divided	28	273	245	1
shared	28	13	23	21.84

Table 6: An example of Lexicalized Tree Grammar compaction

When FSA are shared in a single one, each state contains identifiers of the elementary trees which pass through it and each item the list of elementary tree identifiers valid for the item's positions. To test conditions of a rule we must consider every possible transitions paths according to connected routes and the shared FSA. The resulting item of a rule is valid for a subset of identifiers of the elementary trees passing throught the both position states. The “uncompaction” can be done when we enumerate the derivations.

4 Implementation and results

The algorithm has been implemented in Java and is integrated in a graphical environment system dedicated to grammar design and parsing tests. The parsing workbench allows to test a grammar on corpus and to compare different parsing algorithms for LTAG. The connection driven parsing algorithm and the graphical workbench will be freely avalaible with an open-source licence by the end of 1999. The connection driven parser includes derivation enumeration from the shared parse forest, features unification but not yet sharing of automata.

We present in table 7 statistics for the chart parsing of a corpora of 861 utterances in French of transcripted spontaneous spoken language with a Sun Ultra 1. We compared two sets of rules: One for the bottum-up connected driven parsing and one for a generalized CKY algorithm for LTAG with predictions as suggested by [Vijay-Shanker and Weir, 1993], both on the same data, with the same chart implementation and without agenda. The LTAG grammar for the sublanguage corresponds to a syntactical lexicon of 819 entries and a set of 85 non-instancied elementary trees.

corpus	% complete parses	average no of parses/utter.
Gocad	78.31	2.06

Table 7: Global results of the complete parsing

A partial result corresponds here to the maximal extension of an island, so to an item which is not the origin of any other item. Table 8 shows that the average length of partial recognized substrings is higher with our techniques than with this other bottum-up strategy, preserving a running time better than the predictive CKY. This result means that our algorithm has checked more possible attachments and has delivered more extented partial results.

algorithm	average time /utter. (ms)	average island extension
predictive CKY	87	2.55
connection driven	58	2.79

Table 8: Comparaison between bottum-up parsers for partial parses

The difference between these two running times can be explained by the consideration of connected routes which results in less items to tabulate and by the initialization of foot node. In a predictive CKY-like algorithm for LTAG, the foot nodes are initialized each time that an internal node (with the same category label) is recognized independently to other positions in the auxiliary trees they belong to. This initialization is more determinist in our algorithm since a foot node is initialized when a left or right extension has reached this node.

Conclusion

We have presented a new tabular algorithm dedicated to LTG which really takes into account the fact that we manipulate partial parsing trees and not CF rules. Lexicalized partial parsing trees are richer structures than CF rules and allow to obtain more extended partial results which are relevant for natural language robust parsing. A complementary compaction of the LTG allows the factorization of sub-structures parsing.

This algorithm tries to compromise a chunk parsing and the complete parsing of an utterance, it satisfies the following properties at any moment:

- The maximal extension of islands according to adjacency and local constraints located on the left and right connected route of this island.
- The global correction of each partial results.

The theorical worst case complexity is in $O(N^2G^2n^6)$, but we argue that in practice, when dealing with natural language, decreasing the average complexity is more important than considering a worst case complexity which conditions almost never happen.

When no connected parse can span the whole sentence, our algorithm results not only in more extended partial derivation trees but also consists in representations of islands and both their right and left connected routes. We can then exploit these results for parsing repairs: The adjacency of two islands which can not be combined according to a given LTAG often localizes an agrammaticality. The corresponding chart items can trigger additional repairing rules and flexible controls in a two-step strategy as shown in [Lopez, 1999b].

References

[Abeillé et al., 1999] Abeillé, A., Candito, M.-H., and Kinyon, A. (1999). FTAG: current status and parsing scheme. In *VEXTAL, Venice, Italy.*

[Candito and Kahane, 1998] Candito, M.-H. and Kahane, S. (1998). Defining DTG derivations to get semantic graphs. In *Fourth International Workshop on TAG+ (TAG+4), Philadelphia.*

[Evans and Weir, 1997] Evans, R. and Weir, D. (1997). Automaton-based Parsing for Lexicalized Grammars. In *Fifth International Workshop on Parsing Technologies*, Cambridge, Mass.

[Evans and Weir, 1998] Evans, R. and Weir, D. (1998). A structure-sharing parser for lexicalized grammars. In *COLING-ALC*, Montréal, Canada.

[Joshi and Schabes, 1992] Joshi, A. K. and Schabes, Y. (1992). Tree Adjoining Grammars and lexicalized grammars. In Nivat, M. and Podelski, A., editors, *Tree automata and languages.* Elsevier Science.

[Lang, 1991] Lang, B. (1991). Towards a Uniform Formal Framework for Parsing. In Tomita, M., editor, *Current Issues in Parsing Technology.* Kluwer Academic Publishers.

[Lavelli and Satta, 1991] Lavelli, A. and Satta, G. (1991). Bidirectional parsing of lexicalized tree adjoining grammars. In *Fifth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Berlin, Germany.

[Lopez, 1999a] Lopez, P. (1999a). *Parsing of spoken language for man-machine dialogue with Lexicalized Tree Grammars (in French).* PhD thesis, Université Henry Poincaré Nancy 1.

[Lopez, 1999b] Lopez, P. (1999b). Repairing strategies for Lexicalized Tree Grammars. In *Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Bergen, Norway.

[Magerman and Weir, 1992] Magerman, D. M. and Weir, C. (1992). Efficiency, Robustness, and Accuracy in Picky Chart Parsing. In *ACL'92.*

[Nederhof, 1997] Nederhof, M.-J. (1997). Solving the correct-prefix property for ltags. In *Fifth Meeting on Mathematics of Language (MOL5), Schloss Dagstuhl, Germany.*

[Nederhof, 1998] Nederhof, M.-J. (1998). An alternative LR algorithm for TAGs. In *COLING'98*, Montréal, Quebec.

[Satta and Stock, 1989] Satta, G. and Stock, O. (1989). Formal properties and implementation of bidirectional charts. In *Eleventh International Joint Conference on Artificial Intelligence (IJCAI), Detroit, MI*, pages 1480–1485.

[Schabes, 1994] Schabes, Y. (1994). Left to Right Parsing of Lexicalized Tree Adjoining Grammars. *Computational Intelligence*, 10:506–524.

[Schabes and Waters, 1995] Schabes, Y. and Waters, R. C. (1995). Tree insertion Grammar: A Cubic-Time, Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the Trees Produced. *Computational Intelligence*, 21:479–514.

[Shieber et al., 1995] Shieber, S., Schabes, Y., and Pereira, F. (1995). Principles and Implementation of Deductive Parsing. *Journal of Logic Programming*, 24:3–36.

[van Noord, 1994] van Noord, G. (1994). Head Corner Parsing for TAG. *Computational Intelligence*, 10:525–534.

[Vijay-Shanker, 1987] Vijay-Shanker, K. (1987). *A Study of Tree Adjoining Grammar*. PhD thesis, University of Pennsylvania, Philadelphia.

[Vijay-Shanker and Weir, 1993] Vijay-Shanker, K. and Weir, D. J. (1993). Parsing some constrained grammar formalisms. *Computational Linguistics*, 19:591–636.

Improved Left-Corner Chart Parsing for Large Context-Free Grammars

Robert C. Moore
Microsoft Research
One Microsoft Way
Redmond, Washington 98052, USA
bobmoore@microsoft.com

Abstract

We develop an improved form of left-corner chart parsing for large context-free grammars, introducing improvements that result in significant speed-ups more compared to previously-known variants of left-corner parsing. We also compare our method to several other major parsing approaches, and find that our improved left-corner parsing method outperforms each of these across a range of grammars. Finally, we also describe a new technique for minimizing the extra information needed to efficiently recover parses from the data structures built in the course of parsing.

1 Introduction

Parsing algorithms for contex-free grammars (CFGs) are generally recognized as the backbone of virtually all approaches to parsing natural-language. Even in systems that use a grammar formalism more complex than CFGs (e.g., unification grammar), the parsing method is usually an extension of one of the well-known CFG parsing algorithms. Moreover, recent developments have once again made direct parsing of CFGs more relevant to natural-language processing, including the recent explosion of interest in parsing with stochastic CFGs or related formalisms, and the fact that commercial speech recognition systems are now available (from Nuance Communications and Microsoft) that accept CFGs as language models for constraining recognition.

These applications of context-free parsing share the common trait that the grammars involved can be expected to be very large. A "treebank grammar" extracted from the sections of the Penn Treebank commonly used for training stochastic parsers contains over 15,000 rules, and we also have a CFG containing over 24,000 rules, compiled from a task-specific unification grammar for use as a speech-recognition language model. Grammars such as these stress established approaches to context-free parsing in ways and to an extent not encountered with smaller grammars.

In this work we develop an improved form of left-corner chart parsing for large context-free grammars. We introduce improvements that result in speed-ups averaging 38% or more compared to previously-known variants of left-corner parsing. We also compare our method to several other major parsing approaches: Cocke-Kasami-Younger (CKY), Earley/Graham-Harrison-Ruzzo (E/GHR), and generalized LR (GLR) parsing. Our improved left-corner parsing method outperforms each of these by an average of at least 50%. Finally, we also describe a new technique for minimizing the extra information needed to efficiently recover parses from the data structures built in the course of parsing.

2 Evaluating Parsing Algorithms

In this work we are interested in algorithms for finding all possible parses for a given input. We measure the efficiency of the algorithms in building a complete chart (or comparable structure) for the input, where the chart includes information sufficient to recover every parse without additional searching.[1] We take CPU time to be the primary measure of performance. Implementation-independent measures, such as number of chart edges generated, are sometimes preferred in order to factor out the effects of different platforms and implementation methods, but only time measurement provides a practical way of evaluating some algorithmic details. For example, one of our major improvements to left-corner parsing simply transposes the order of performing two independent filtering checks, resulting in speed ups of up to 67%, while producing exactly the same chart edges as the previous method. To ensure comparability of time measurements, we have re-implemented all the algorithms considered, in Perl 5,[2] on as similar a basis as possible.

One caveat about this evaluation should be noted. None of the algorithms were implemented with general support for empty categories, due to the fact that none of the large, independently motivated grammars we had access to contained empty categories. We did, however make use of a grammar transformation (left factoring) that can produce empty categories, but only as the right-most daughter of a rule with at least two daughters. For the algorithms we wanted to test with this form of grammar, we added limited support for empty categories specifically in this position.

3 Terminology and Notation

Nonterminals, which we will sometimes refer to as categories, will be designated by "low order" upper-case letters (A, B, etc.); and terminals will be designated by lower-case letters. We will use the notation a_i to indicate the ith terminal symbol in the input string. We will use "high order" upper-case letters (X, Y, Z) to denote single symbols that could be either terminals or nonterminals, and Greek letters to denote (possibly empty) sequences of terminals and/or nonterminals. For a grammar rule $A \rightarrow B_1 \ldots B_n$ we will refer to A as the mother of the rule and to $B_1 \ldots B_n$ as the daughters of the rule. We will assume that there is a single nonterminal category S that subsumes all sentences allowed by the grammar.

All the algorithms considered here build a collection of data structures representing segments of the input partially or completely analyzed as a phrase of some category in the grammar, which we will refer to as a "chart". We will use the term "item" to mean an instance of a grammar rule with an indication of how many of the daughters have been recognized in the input. Items will be represented as dotted rules, such as $A \rightarrow B_1.B_2$. An "incomplete item" will be an item with at least one daughter to the right of the dot, indicating that at least one more daughter remains to be recognized before the entire rule is matched; and a "complete item" will be an item with no daughters to the right of the dot, indicating that the entire rule has been matched.

We will use the terms "incomplete edge" or "complete edge" to mean an incomplete item or complete item, plus two input positions indicating the segment of the input covered by the daughters that have

[1]Formally, we require that for any m up to the total number parses of the input, we can extract from the chart m parses of a string of length n in time proportional to $m \cdot n$.

[2]We take advantage of Perl 5's ability to arbitrarily nest hash tables and linked lists to produce efficient implementations of the data structures required by the algorithms. In particular, the multi-dimensional arrays required by many of the algorithms are given a sparse-matrix implementation in terms of multiply-nested Perl hash tables.

already been recognized. These will be written as (e.g.) $\langle A \rightarrow B_1 B_2 . B_3, i, j \rangle$, which would mean that the sequence $B_1 B_2$ has been recognized starting at position i and ending at position j, and has been hypothesized as part of a longer sequence ending in B_3, which is classified a phrase of category A. Positions in the input will be numbered starting at 0, so the ith terminal of an input string spans position $i-1$ to i. We will refer to items and edges none of whose daughters have yet been recognized as "initial".

4 Test Grammars

For testing context-free parsing algorithms, we have selected three CFGs that are independently motivated by analyses of natural-language corpora or actual applications of natural language processing. The CT grammar[3] was compiled into a CFG from a task-specific unification grammar written for CommandTalk (Moore et al., 1997), a spoken-language interface to a military simulation system. The ATIS grammar was extracted from an internally generated treebank of the DARPA ATIS3 training sentences. The PT grammar was extracted from the Penn Treebank.[4] We employ a standard test set for each of the three grammars. The test set for the CT grammar is a set of sentences made up by the system designers to test the functionality of the system, and the test set for the ATIS grammar is a randomly selected subset of the DARPA ATIS3 development test set. The test set for the PT grammar is a set of sentences randomly generated from a probabilistic version of the grammar, with the probabilities based on the frequency of the bracketings occuring in the training data, and then filtered for length to make it possible to conduct experiments in a reasonable amount of time, given the high degree of ambiguity of the grammar.

The terminals of the grammars are preterminal lexical categories rather than words. Preterminals were generated automatically, by grouping together all the words that could occur in exactly the same contexts in all grammar rules, to eliminate lexical ambiguity.

	CT Grammar	ATIS Grammar	PT Grammar
Rules	24,456	4,592	15,039
Nonterminals	3,946	192	38
Terminals	1,032	357	47
# Test Sentences	162	98	30
Average Length	8.3	11.4	5.7
# Grammatical	150	70	30
Average # Parses	5.4	940	7.2×10^{27}

Table 1: Grammars and test sets for parser evaluations

Some statistics on the grammars and test sets are contained in Table 1. Note that for the CT and ATIS sets, not all sentences are within the corresponding grammars. The most striking difference among the three grammars is the degree of ambiguity. The CT grammar has relatively low ambiguity, the ATIS grammar may be considered highly ambiguous, and the PT grammar can only be called massively ambiguous.

[3]Courtesy of John Dowding, SRI International
[4]Courtesy of Eugene Charniak, Brown University

5 Left-Corner Parsing Algorithms and Refinements

Left-corner (LC) parsing—more specifically, left-corner parsing with top-down filtering—originated as a method for deterministically parsing a restricted class of CFGs. It is often attributed to Rosenkrantz and Lewis (1970), who may have first used the term "left-corner parsing" in print. Griffiths and Petrick (1965), however, previously described an LC parsing algorithm under the name "selective bottom-to-top" (SBT) parsing, which they assert to be an abstraction of previously described algorithms.

The origins of LC parsing for general CFGs (other than by naive backtracking) are even murkier. Pratt's (1975) algorithm is sometimes considered to be a generalized LC method, but it is perhaps better described as CKY parsing with top-down filtering added. Kay's (1980) method for undirected bottom-up chart parsing is clearly left-corner parsing without top-down filtering, but in adding top-down filtering to obtain directed bottom-up chart parsing, he changed the method significantly. The BUP parser of Matsumoto et al. (1983) appears to be the first clearly described LC parser capable of parsing general CFGs in polynomial time.[5]

LC parsing depends on the left-corner relation for the grammar, where X is recursively defined to be a left corner of A if $X = A$, or the grammar contains a rule of the form $B \rightarrow X\alpha$, where B is a left corner of A. This relation is normally precompiled and indexed so that any pair of symbols can be checked in essentially constant time.

Although LC parsing was originally defined as a stack-based method, implementing it in terms of a chart enables polynomial time complexity to be achieved by the use of dynamic programming; which simply means that if the same chart edge is derived in more than one way, only one copy is retained for further processing. A chart-based LC parsing algorithm can be defined by the following set of rules for populating the chart:

1. For every grammar rule with S as its mother, $S \rightarrow \alpha$, add $\langle S \rightarrow .\alpha, 0, 0\rangle$ to the chart.

2. For every pair of edges of the form $\langle A \rightarrow \alpha.X\beta, i, k\rangle$ and $\langle X \rightarrow \gamma., k, j\rangle$ in the chart, add $\langle A \rightarrow \alpha X.\beta, i, j\rangle$ to the chart.

3. For every edge of the form $\langle A \rightarrow \alpha.a_i\beta, i, k\rangle$ in the chart, where a_i is the ith terminal in the input, add $\langle A \rightarrow \alpha a_i.\beta, i, j\rangle$ to the chart.

4. For every pair of edges of the form $\langle A \rightarrow \alpha.C\beta, i, k\rangle$ and $\langle X \rightarrow \gamma., k, j\rangle$ in the chart and every grammar rule with X as its left-most daughter, of the form $B \rightarrow X\delta$, if B is a left corner of C, add $\langle B \rightarrow X.\delta, k, j\rangle$ to the chart.

5. For every edge of the form $\langle A \rightarrow \alpha.C\beta, i, k\rangle$, where a_i is the ith terminal in the input, and every grammar rule with a_i as its left-most daughter, of the form $B \rightarrow a_i\delta$, if B is a left corner of C, add $\langle B \rightarrow X.\delta, k, j\rangle$ to the chart.

An input string is successfully parsed as a sentence if the chart contains an edge of the form $\langle S \rightarrow \alpha., 0, n\rangle$ when the algorithm terminates.

Rules 1–3 are shared with other parsing algorithms, notably E/GHR, but rules 4 and 5 are distinctive to LC parsing. If naively implemented, however, they can lead to unnecessary duplication of work. Rules 4 and 5 state that for every triple consisting of an incomplete edge, a complete edge or input terminal, and a grammar rule, meeting certain conditions, a new edge should be added to the chart.

[5]Cyclic grammars and empty categories were not supported, however.

Inspection reveals, however, that the form of the edge to be added depends on only the complete edge or input terminal and the grammar rule, not the incomplete edge. Thus if this parsing rule is applied separately for each triple, the same new edge may be proposed repeatedly if several incomplete edges combine with a given complete edge or input terminal and grammar rule to form triples satisfying the required conditions. A number of implementations of generalized LC parsing have suffered from this problem, including the BUP parser, the left-corner parser of the SRI Core Language Engine (Moore and Alshawi, 1991), and Schabes's (1991) table-driven predictive shift-reduce parser.

However, if parsing is performed strictly left-to-right, so that every incomplete edge ending at k has already been computed before any left-corner checks are performed for new edges proposed from complete edges or input terminals starting at k, there is a solution that can be seen by rephrasing rules 4 and 5 follows:

4a. For every edge of the form $\langle X \rightarrow \gamma., k, j\rangle$ in the chart and every grammar rule with X as its left-most daughter, of the form $B \rightarrow X\delta$, if there is an incomplete edge in the chart ending at k, $\langle A \rightarrow \alpha.C\beta, i, k\rangle$, such that B is a left corner of C, add $\langle B \rightarrow X.\delta, k, j\rangle$ to the chart.

5a. For every input terminal a_i and every grammar rule with a_i as its left-most daughter, of the form $B \rightarrow a_i\delta$, if there is an incomplete edge in the chart ending at k, $\langle A \rightarrow \alpha.C\beta, i, k\rangle$, such that B is a left corner of C, add $\langle B \rightarrow a_i.\delta, k, j\rangle$ to the chart.

This formulation suggests driving the parser by proposing a new edge from every grammar rule exactly once for each complete edge or input terminal corresponding to the rule's left-most daughter, and then checking whether some previous incomplete edge licenses it via left-corner filtering. If implemented by nested iteration, this still requires as many nested loops as the naive method; but the inner-most loop does much less work, and it can be aborted as soon as one previous incomplete edge has been found to satisfy the left-corner check. Wirén (1987) seems to have been the first to explicitly propose performing left-corner filtering in an LC parser in this way. Nederhof (1993) proposes essentially the same solution, but formulated in terms of a graph-structured stack of the sort generally associated with GLR parsing.

Several additional optimizations can be added to this basic schema. Wirén adds bottom-up filtering (Wirén uses the term "selectivity", following Griffiths and Petrick (1965)) of incomplete edges based on the next terminal in the input. That is, no incomplete edge of the form $\langle A \rightarrow \alpha.X\beta, i, k\rangle$ is added to the chart unless a_k is a left corner of X. Nederhof proposes that, rather than iterate over all the incomplete edges ending at k each time a left-corner check is performed, compute just once for each input position a set of nonterminal predictions, consisting of the symbols immediately to the right of the dot in the incomplete edges ending at that position, and iterate over that set for each left-corner check at the position.[6] With this optimization, it is no longer necessary to add initial edges to the chart at position 0 for rules of the form $S \rightarrow \alpha$. If P_i denotes the set of predictions for position i, we simply let $P_0 = \{S\}$.

Another optimization from the recent literature is due to Leermakers (1992), who observes that in Earley's algorithm the daughters to the *left* of the dot in an item play no role in the parsing algorithm; thus the representation of items can ignore the daughters to the left of the dot, resulting in fewer distinct edges to be considered. This observation is equally true for LC parsing. Thus, instead of $A \rightarrow B_1B_2.B_3$, we will write simply $A \rightarrow .B_3$. Note that with this optimization, $A \rightarrow .$ becomes

[6]Nederhof proposes several other optimizations, which we evaluated and found not to repay their overhead.

the notation for an item all of whose daughters have been recognized; the only information it contains being just the mother of the rule. We will therefore write complete edges simply as $\langle A, i, j\rangle$, rather than $\langle A \rightarrow ., i, j\rangle$. We can also unify the treatment of terminal symbols in the input with complete edges in the chart by adding a complete edge $\langle a_i, i-1, i\rangle$ to the chart for every input terminal a_i.[7]

Taking all these optimizations together, we can define an optimized LC parsing algorithm by the following set of parsing rules:

1. Let $P_0 = \{S\}$.

2. For every input position $j > 0$, let $P_j = \{B \mid$ there is an incomplete edge in the chart ending at j, of the form $\langle A \rightarrow .B\alpha, i, j\rangle\}$.

3. For every input terminal a_i, add $\langle a_i, i-1, i\rangle$ to the chart.

4. For every pair of edges $\langle A \rightarrow .XY\alpha, i, k\rangle$ and $\langle X, k, j\rangle$ in the chart, if a_j is a left corner of Y, add $\langle A \rightarrow .Y\alpha, i, j\rangle$ to the chart.

5. For every pair of edges $\langle A \rightarrow .X, i, k\rangle$ and $\langle X, k, j\rangle$ in the chart, add $\langle A, i, j\rangle$ to the chart.

6. For every edge $\langle X, k, j\rangle$ in the chart and every grammar rule with X as its left-most daughter, of the form $A \rightarrow XY\alpha$, if there is a $B \in P_k$ such that A is a left corner of B, and a_j is a left corner of Y, add $\langle A \rightarrow .Y\alpha, k, j\rangle$ to the chart.

7. For every edge $\langle X, k, j\rangle$ in the chart and every grammar rule with X as its only daughter, of the form $A \rightarrow X$, if there is a $B \in P_k$ such that A is a left corner of B, add $\langle A, k, j\rangle$ to the chart.

Note that in Rule 6, the top-down left-corner check on the mother of the proposed incomplete edge and the bottom-up left-corner check on the symbol immediately to the right of the dot in the proposed incomplete edge are independent of each other, and therefore could be performed in either order. Wirén, the only author we have found who includes both, is vague on the ordering of these checks. For each proposed edge, however, the bottom-up check requires examining an entry in the left-corner table for each of the elements of the prediction list, until a check succeeds or the list is exhausted; while the bottom up check requires examining only a single entry in the left-corner table for the next terminal of the input. It therefore seems likely to be more efficient to do the bottom-up check before the top-down check, since the top-down check need not be performed if the bottom-up check fails. To test this hypothesis, we have done two implementations of the algorithm: LC_1, which performs the top-down check first, and LC_2, which performs the bottom-up check first.

Shann (1991) uses a different method of top-down filtering in an LC parser. Shann expands the list of predictions created by rules 1 and 2 to include all the left-corners of the predictions. He does this by precomputing the proper left corners of all nonterminal categories and adding to the list of predictions all the left-corners of the original members of the list. Then top-down filtering consists of simply checking whether the mother of a proposed incomplete edge is on the corresponding prediction list. Graham, Harrison, and Ruzzo (1980) attribute this type of top-down filtering to Cocke and Schwartz, so we will refer to it as "Cocke-Schwartz filtering". Since our original form of filtering uses the left-corner relation directly, we will call it "left-corner filtering".

[7]Many chart parsers unify the treatment of input terminals and complete edges in this way, by ignoring daughters to the left of the dot, but only for complete edges. The Leermakers optimization permits a unified treatment of incomplete edges, complete edges, and input terminals.

We have implemented Cocke-Schwartz filtering as described by Shann, except that for efficiency in both forming and checking the sets of predictions, we use hash tables rather than lists. The resulting algorithm, which we will call LC_3, can be stated as follows:

1. Let $P_0 = \{\text{all left corners of } S\}$.[8]

2. For every input position $j > 0$, let $P_j = \{\text{all left corners of } B \mid \text{there is an incomplete edge in the chart ending at } j, \text{ of the form } \langle A \rightarrow .B\alpha, i, j\rangle\}$.

3. For every input terminal a_i, add $\langle a_i, i-1, i\rangle$ to the chart.

4. For every pair of edges $\langle A \rightarrow .XY\alpha, i, k\rangle$ and $\langle X, k, j\rangle$ in the chart, if a_j is a left corner of Y, add $\langle A \rightarrow .Y\alpha, i, j\rangle$ to the chart.

5. For every pair of edges $\langle A \rightarrow .X, i, k\rangle$ and $\langle X, k, j\rangle$ in the chart, add $\langle A, i, j\rangle$ to the chart.

6. For every edge $\langle X, k, j\rangle$ in the chart and every grammar rule with X as its left-most daughter, of the form $A \rightarrow XY\alpha$, if $A \in P_k$, and a_j is a left corner of Y, add $\langle A \rightarrow .Y\alpha, k, j\rangle$ to the chart.

7. For every edge $\langle X, k, j\rangle$ in the chart and every grammar rule with X as its only daughter, of the form $A \rightarrow X$, if $A \in P_k$, add $\langle A, k, j\rangle$ to the chart.

There is one simple refinement, not mentioned by Shann, that we can add to this algorithm. Since we already have the information needed to perform bottom-up filtering, we can apply bottom-up filtering to building the prediction sets, omitting any left-corner of an existing prediction that is incompatible with the next terminal of the input. This will certainly save space, and may save time as well, depending on the relative costs of adding a nonterminal to the prediction set compared to performing the bottom-up left-corner check. Our modification of LC parsing with Cocke-Schwartz filtering to include this refinement is implemented as LC_4.

	CT Grammar	ATIS Grammar	PT Grammar
LC_1	4.3	15.6	45.0
LC_2	3.4	11.9	43.0
LC_3	3.1	11.6	41.8
LC_4	2.7	11.8	42.3

Table 2: LC parsing algorithm performance comparisons

The results of running algorithms LC_1–LC_4 appear in Table 2. The numbers are CPU time in seconds required by the parser to completely process the standard test set associated with each grammar.[9] LC_2, which performs the bottom-up left-corner check on proposed incomplete edges before top-down left-corner check, is faster on all three grammars than LC_1, which performs the checks in the reverse order—substantially so on the CT and ATIS grammars. Comparing LC_3 with LC_4—which both use Cocke-Schwartz filtering, but differ as to whether the prediction sets are bottom-up filtered—the results are less clear. LC_4, which does filter the predictions, is noticably faster on the CT grammar, while LC_3 which does not filter predictions is slightly faster, but not significanly so, on the ATIS grammar and PT grammar. Finally, both parsers that use Cocke-Schwartz filtering are faster on all grammars than either of the parsers that use left-corner filtering.

[8]Recall that by our definition, the left-corner relation is reflexive so S will be included.

[9]All timings in this report are for execution on a Dell 610 workstation with Pentium III Xeon 550 MHz processors running Windows 2000.

6 Grammar Transformations

One other issue remains to be addressed in our examination of LC parsing. It is a common observation about left-to-right parsing, that if two grammar rules share a common left prefix, e.g., $A \rightarrow BC$ and $A \rightarrow BD$, many parsing algorithms will duplicate work for the two rules until reaching the point where they differ. A simple solution often proposed to address the problem is to "left factor" the grammar. Left factoring applies the following grammar transformation repeatedly, until it is no longer applicable:

> For each nonterminal A, let α be the longest nonempty sequence such that there is more than one grammar rule of the form $A \rightarrow \alpha\beta$. Replace the set of rules $A \rightarrow \alpha\beta_1, \ldots, A \rightarrow \alpha\beta_n$ with $A \rightarrow \alpha A', A' \rightarrow \beta_1, \ldots, A' \rightarrow \beta_n$, where A' is a new nonterminal symbol.

Left factoring has been explored in the context of generalized LC parsing by Nederhof (1994), who refers to LC parsing with left factoring as PLR parsing. Shann (1991) also applies left factoring directly in the representation of the rules he uses in his LC parser, e.g. $A \rightarrow B(C, D)$.

One complication associated with left factoring is that if the daughters of one rule are a proper prefix of the daughters of another rule, then empty rules will be introduced into the grammar, even if there were none originally. For example $A \rightarrow BC$ and $A \rightarrow BCD$ will be replaced by $A \rightarrow BCA', A' \rightarrow D, A' \rightarrow \epsilon$. To explore the cost of this additional complication we compare full left factoring with the following restricted form of left factoring;

> For each nonterminal A, let α be the longest nonempty sequence such that there is more than one grammar rule of the form $A \rightarrow \alpha\beta$, for some nonempty string β. Replace the set of rules $A \rightarrow \alpha\beta_1, \ldots, A \rightarrow \alpha\beta_n$ with $A \rightarrow \alpha A', A' \rightarrow \beta_1, \ldots, A' \rightarrow \beta_n$, where A' is a new nonterminal symbol.

The requirement that β always be nonempty blocks the introduction of empty productions, so with this transformation $A \rightarrow BC$ and $A \rightarrow BCD$ will be replaced by $A \rightarrow BA', A' \rightarrow CD, A' \rightarrow C$.

Left factoring is not the only transformation that can be used to address the problem of common rule prefixes. Left factoring applies only to sets of rules with a common mother category, but as an essentially bottom-up method, generalized LC parsing does most of its work before the mother of a rule is determined. There is another grammar transformation that seems better suited to LC parsing, introduced by Griffiths and Petrick (1965), but apparently neglected since:

> For each grammar symbol X, let α be the longest nonempty sequence such that there is more than one grammar rule of the form $A \rightarrow X\alpha\beta$. Replace the set of rules $A_1 \rightarrow X\alpha\beta_1, \ldots, A_n \rightarrow X\alpha\beta_n$ with $A' \rightarrow X\alpha, A_1 \rightarrow A'\beta_1, \ldots, A_n \rightarrow A'\beta_n$, where A' is a new nonterminal symbol.

Like left factoring, this transformation is repeated until it is no longer applicable. Griffiths and Petrick do not give this transformation a name, so we will call it "bottom-up prefix merging".

It should be noted that all of these grammar transformations simply add additional levels of nonterminals to the grammar, without otherwise disturbing the structure of the analyses produced by the grammar. Thus, when parsing with a grammar produced by one of these transformations, the original analyses can be recovered simply by ignoring the newly introduced nonterminals, and treating their subconstituents as subconstituents of the next higher original nonterminal of the grammar.

Before we apply our LC parsers to our test grammars transformed in these three ways, we make a few small adjustments to the implementations. First, as noted above, full left factoring requires the ability to handle empty categories, at least as the right-most daughter of a rule. We have created modified versions of LC_1–LC_4 specifically to use with the fully left-factored grammar. Second we note that with a left-factored grammar,[10] the non-unary rules have the property that, given the mother and the left-most daughter, there is only one possibility for the rest of the rule. With a bottom-up prefix-merged grammar, the non-unary rules have the property that, given the two left-most daughters, there is only one possibility for the rest of the rule. We take advantage of these facts to store the indexed forms of the rules more compactly and simplify the logic of the implementations of variants of our parsers specialized to these grammar forms.

		CT Grammar	ATIS Grammar	PT Grammar
LC_1	UTF	4.3	15.6	45.0
LC_1	FLF	7.4	63.5	timed out
LC_1	PLF	6.2	66.2	timed out
LC_1	BUPM	3.6	11.7	34.1
LC_2	UTF	3.4	11.9	43.0
LC_2	FLF	5.1	38.2	timed out
LC_2	PLF	4.2	37.7	timed out
LC_2	BUPM	3.1	7.0	27.0
LC_3	UTF	3.1	11.6	41.8
LC_3	FLF	4.2	12.3	45.4
LC_3	PLF	3.8	12.1	43.6
LC_3	BUPM	5.0	17.1	64.6
LC_4	UTF	2.7	11.8	42.3
LC_4	FLF	3.6	11.9	46.6
LC_4	PLF	3.2	11.7	44.4
LC_4	BUPM	3.2	14.7	63.6

Table 3: LC parsing grammar transformation performance comparisons

The results of applying our four LC parsing algorithms with these three grammar transformations are displayed in Table 3, along with results for the untransformed grammars presented previously. The grammar transformations are desginated by the symbols UTF (untransformed), FLF (fully left-factored), PLF (partially left-factored), and BUPM (bottom-up prefix-merged). We set a time-out of 10 minutes on some experiments, since that was already an order of magnitude longer than any of the other parse times. Several observations stand out from these results. First, in every case but one, partial left factoring out-performed full left factoring. Much more surprising is that, in every case but one, either form of left factoring degraded parsing performance relative to the untransformed grammar. For LC_1 and LC_2, the algorithms that use left-corner filtering, the degradation is dramatic, while for LC_3 and LC_4, which use Cocke-Schwartz filtering, the degradation is very slight in the case of the ATIS and PT grammars, but more pronounced in the case of the CT grammar. On the other hand, bottom-up prefix-merging significantly—in some cases dramatically—speeds up parsing for LC_1 and LC_2, while significantly degrading the performance of LC_3 and LC_4.

Looking at the overall results of these experiments, we see that bottom-up prefix merging reverses the previous advantage of Cocke-Schwartz filtering over left-corner filtering. With bottom-up prefix merging, LC_2 is at least 66% faster on the ATIS grammar and 55% faster on the PT grammar than either LC_3 or LC_4; and it is only 15% slower than LC_4 on the CT grammar, and the same speed as LC_3. Averaging over the three test grammars, LC_2 is 40% faster than LC_3 and 38% faster than LC_4.

[10]either fully left-factored, or partially left-factored using our restricted transformation.

7 Extracting Parses from the Chart

The Leermakers optimization of omitting recognized daughters from items raises the question of how parses are to be extracted from the chart. The daughters to the left of the dot in an item are often used for this purpose in item-based methods, including Earley's original algorithm. Graham, Harrison, and Ruzzo (1980), however, suggest storing with each noninitial edge in the chart a list that includes, for each derivation of the edge, a pair of pointers to the preceding edges that caused it to be derived. This provides sufficient information to extract the parses without additional searching, even without the daughters to the left of the dot.

In fact, we can do even better than this. For each derivation of a noninitial edge, even in the Leermakers representation, it is sufficient to attach to the edge only the mother category and starting position of the complete edge that was used in the last step of the derivation. Every noninitial edge is derived by combining a complete edge with an incomplete edge. Suppose $\langle A \rightarrow .\beta, k, j\rangle$ is a derived edge, and we know that the complete edge used in the derivation had category X and start position i. We then know that the complete edge must have been $\langle X, i, j\rangle$, since the complete edge and the derived edge must have the same end position. We further know that the incomplete edge used in the derivation must have been $\langle A \rightarrow .X\beta, k, i\rangle$, since that is the only incomplete edge that could have combined with the complete edge to produce the derived edge. In this way, for any complete edge, we can trace back through the chart until we have found all the complete edges for the daughters that derived it. The back-trace terminates when we reach an incomplete edge that has the same start point as the complete edge it was derived from.

8 Comparison to Other Algorithms

We have compared our LC parsers to efficient implementations of three other important approaches to context-free parsing: Cocke-Kasami-Younger (CKY), Earley/Graham-Harrison-Ruzzo (E/GHR), and generalized LR (GLR) parsing. We include CKY, not because we think it may be the fastest parsing algorithm, but because it provides a baseline of how well one can do with no top-down filtering. Our implementation of E/GHR includes many optimizations not found in the original descriptions of this approach, including the techniques used to optimize our LC parsers, where applicable. In our GLR parser we used the same reduction method as Tomita's (1985) original parser, which results in greater-than-cubic worst-case time complexity, after verifying that a cubic-time version was, in fact, slower in practice, as Tomita has asserted.

	CT Grammar	ATIS Grammar	PT Grammar
LC_2+BUPM	3.1	7.0	27.0
CKY	25.0	7.7	50.9
E/GHR	7.3	8.6	27.7
GLR(0)	3.2	14.0	timed out
LC+follow	2.4	6.6	29.6
GLR(0)+follow	2.3	14.1	timed out
GLALR(1)	3.8	14.7	—

Table 4: Alternative parsing algorithm performance comparisons

Table 4 shows the comparison between these three algorithms, and our best overall LC algorithm.

As the table shows, LC_2+BUPM outperforms all of the other algorithms with all three grammars. While each of the other algorithms approaches our LC parser in at least one of the tests, the LC parser outperforms each of the others by at least a factor of 2 with at least one of the grammars.

The comparison between LC_2+BUPM and GLR is instructive in view of the claims that have been made for GLR. While GLR(0) was essentially equal in performance to LC_2+BUPM on the least ambiguous grammar, it appears to scale very badly with increasing ambiguity. Moreover, the parsing tables required by the GLR parser are far larger than for LC_2+BUPM. For the CT grammar, LC_2+BUPM requires 27,783 rules in the transformed grammar, plus 210,701 entries in the left-corner table. For the (original) CT grammar, GLR requires 1,455,918 entries in the LR(0) parsing tables.

The second part of Table 4 shows comparisons of LC_2+BUPM and two versions of GLR with look ahead. The "LC+follow" line is for LC_2+BUPM plus an additional filter on complete edges using a "follow check" equivalent to the look ahead used by SLR(1) parsing. The "GLR(0)+follow" line adds the same follow check to the GLR(0) parser. This builds exactly the same edges as a GSLR(1) parser would, but allows smaller parsing tables at the expense of more table look ups.[11] With the follow check, the parse times for the CT grammar are substantially reduced, but LC_2+BUPM and GLR remain essentially equivalent, while only small changes are produced for the ATIS and PT grammars.

The final line gives results for GLALR(1) parsing with the CT and ATIS grammars.[12] These results are not directly comparable to the others because the LALR(1) reduce tables for the CT and ATIS grammars contained more than 6.1 million and 1.8 million entries, respectively, and they would not fit in the memory of the test machine along with the other LR tables. Various methods were investigated to obtain timing results by loading only a subset of the reduce tables sufficient to handle the test set. These gave inconsistent results, but in all cases times were longer than for either GLR(0) or GLR(0)+follow, presumably due to additional overhead caused by the large tables, with relatively little additional filtering (5–6% fewer edges). The numbers in the table represent the best results obtained for each grammar.

9 Conclusions

Probably the two most significant results of this investigation are the discoveries that:

- LC chart parsing incorporating both a top-down left-corner check on the mother of a proposed incomplete edge and a bottom-up left-corner check on the symbol immediately to the right of the dot in the proposed incomplete edge is substantially faster if the bottom-up check is performed before the top-down check.

- Bottom-up prefix merging is a particularly good match to LC chart parsing based on left-corner filtering, and in fact substantially out performs left factoring combined with LC chart parsing in most circumstances.

Moreover we have shown that with these enhancements, LC parsing outperforms several other major approaches to context-free parsing, including some previously claimed to be the best general context-free parsing method. We conclude that our improved form of LC parsing may now be the leading contender for that title.

[11]A GSLR(1) reduce table is just the composition of a GLR(0) reduce table and a follow-check table.

[12]No experiments were done for the PT grammar due to the excessively long time required to compute LALR(1) parsing tables for that grammar, given the expectation that the parser would still time out.

References

Graham, S.L., M.A. Harrison, and W.L. Ruzzo (1980) "An Improved Context-Free Recognizer," *ACM Transactions on Programming Languages and Systems*, Vol. 2, No. 3, pp. 415–462.

Griffiths, T.V., and S.R. Petrick (1965) "On the Relative Efficiencies of Context-Free Grammar Recognizers," *Communications of the ACM*, Vol. 8, No. 5, pp. 289–300.

Kay, M. (1980) "Algorithm Schemata and Data Structures in Syntactic Processing," Report CSL–80–12, Xerox PARC, Palo Alto, California.

Leermakers, R. (1992) "A Recursive Ascent Earley Parser," *Information Processing Letters*, Vol. 41, No. 2, pp. 87–91.

Matsumoto, Y., et al. (1983) "BUP: A Bottom-Up Parser Embedded in Prolog," *New Generation Computing*, Vol. 1, pp. 145–158.

Moore, R., et al. (1997) "CommandTalk: A Spoken-Language Interface for Battlefield Simulations," in *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Association for Computational Linguistics, Washington, DC, pp. 1–7.

Moore, R., and H. Alshawi (1991) "Syntactic and Semantic Processing," in *The Core Language Engine*, H. Alshawi (ed.), The MIT Press, Cambridge, Massachusetts, pp. 129–148.

Nederhof, M.J. (1993) "Generalized Left-Corner Parsing," in *Proceedings of the Sixth Conference of the European Chapter of the Association for Computational Linguistics*, Utrecht, The Netherlands, pp. 305–314.

Nederhof, M.J. (1994) "An Optimal Tabular Parsing Algorithm," in *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, Las Cruces, New Mexico, pp. 117–124.

Pratt, V.R. (1975) "LINGOL — A Progress Report," in *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, Tbilisi, Georgia, USSR, pp. 422–428.

Rosenkrantz, D.J., and P.M. Lewis (1970) "Deterministic Left Corner Parsing," in *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pp. 139–152.

Shann, P. (1991) "Experiments with GLR and Chart Parsing," in *Generalized LR Parsing*, M. Tomita (ed.), Kluwer Academic Publishers, Boston, Massachusetts, pp. 17–34.

Schabes, Y. (1991) "Polynomial Time and Space Shift-Reduce Parsing of Arbitrary Context-free Grammars," in *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, California, pp. 106–113.

Tomita, M. (1985) *Efficient Parsing for Natural Language*, Kluwer Academic Publishers, Boston, Massachusetts.

Wirén, M. (1987) "A Comparison of Rule-Invocation Strategies in Context-Free Chart Parsing," in *Proceedings of the Third Conference of the European Chapter of the Association for Computational Linguistics*, Copenhagen, Denmark, pp. 226–233.

Measure For Measure: Parser Cross-Fertilization*

— Towards Increased Component Comparability and Exchange —

Stephan Oepen, Ulrich Callmeier

Computational Linguistics, Saarland University,

Postfach 15 11 50, 66041 Saarbrücken, Germany

e-mail: {oe | uc}@coli.uni-sb.de

Abstract

Over the past few years significant progress was accomplished in efficient processing with wide-coverage HPSG grammars. HPSG-based parsing systems are now available that can process medium-complexity sentences (of ten to twenty words, say) in average parse times equivalent to real (i.e. human reading) time. A large number of engineering improvements in current HPSG systems were achieved through collaboration of multiple research centers and mutual exchange of experience, encoding techniques, algorithms, and even pieces of software. This article presents an approach to grammar and system engineering, termed *competence & performance profiling*, that makes systematic experimentation and the precise empirical study of system properties a focal point in development. Adapting the profiling metaphor familiar from software engineering to constraint-based grammars and parsers, enables developers to maintain an accurate record of system evolution, identify grammar and system deficiencies quickly, and compare to earlier versions or between different systems. We discuss a number of exemplary problems that motivate the experimental approach, and apply the empirical methodology in a fairly detailed discussion of what was achieved during a development period of three years. Given the collaborative nature in setup, the empirical results we present involve research and achievements of a large group of people.

Dramatis Personæ

[...] *we view the discovery of parsing strategies as a largely experimental process of incremental optimization.* (Erbach, 1991a)

[...] *the study and optimisation of unification-based parsing must rely on empirical data until complexity theory can more accurately predict the practical behaviour of such parsers.* (Carroll, 1994)

Early 1994, research groups at Saarbrücken[1] (Uszkoreit et al., 1994) and CSLI Stanford[2] (Copestake, 1992; Flickinger & Sag, 1998) started to collaborate on the development of large-scale HPSG grammars, suitable grammar engineering platforms, and efficient processors. While both sites had worked on HPSG implementation before, the joint effort has greatly increased productivity, resulted in a mutual exchange of knowledge and technology, and helped building a collection of grammar development environments, several highly engineered parsers (Kiefer, Krieger, Carroll, & Malouf, 1999) and an efficient generator (Carroll, Copestake, Flickinger, & Poznanski, 1999). Recently, the HPSG group at Tokyo University[3] (Torisawa & Tsujii, 1996) has entered the stage and now supplies additional expertise on (abstract-machine-based) compilation of typed feature structures, Japanese HPSG, and grammar approximation techniques.

*This play builds heavily on results obtained through collaboration between a largish group of people at several sites; Act 5 presents a list of individuals who have contributed to the development providing the background for our current discussion. Technical details and empirical assessments of individual techniques will be documented in a forthcoming Special Issue of the *Journal of Natural Language Engineering* (Flickinger, Oepen, Uszkoreit, & Tsujii, 2000).

[1]See 'http://www.dfki.de/lt/' and 'http://www.coli.uni-sb.de/' for information on the DFKI Language Technology Laboratory and the Computational Linguistics Department at Saarland University, respectively.

[2]The 'http://lingo.stanford.edu/' web pages list HPSG-related projects and people involved at CSLI, and also provide an on-line demonstration of the LKB system and LinGO grammar.

[3]Information on the Tokyo laboratory, founded and managed by Professor Jun'ichi Tsujii, can be found at 'http://www.is.s.u-tokyo.ac.uk/'.

Although the individual systems often supply extra functionality, the groups have converged on a common descriptive formalism — a conservative blend of Carpenter (1992), Copestake (1992), and Krieger and Schäfer (1994) — that allows grammars[4] a to be processed by five different platforms. The LinGO grammar, a multi-purpose, broad-coverage grammar of English developed at CSLI and among the largest HPSG implementations currently available, serves as a common reference for all three groups (while of course the sites continue development of additional grammars for English, German, and Japanese). With one hundred thousand lines of source, roughly eight thousand types, an average feature structure size of some three hundred nodes, twenty seven lexical and thirty seven phrase structure rules, and some six thousand lexical (stem) entries, the LinGO grammar presents a fine challenge for processing systems. While scaling the systems to this rich set of constraints and improving processing and constraint resolution algorithms, the groups have regularly exchanged benchmarking results, in particular at the level of individual components, and discussed benefits and disadvantages of particular encodings and algorithms. Precise comparison was found essential in this process and has facilitated a degree of cross-fertilization that proved beneficial for all participants.

Act 1 below introduces the profiling methodology, supporting tools, and the sets of common reference data and benchmarking metrics that were used among the groups. By way of example, the profiling metaphor is then applied in Act 2 to a choice of engineering issues that (currently) can only be approached empirically. Act 3 introduces the PET platform (Callmeier, 2000) as another actor in the experimental setup; PET synthesizes a variety of techniques from the individual systems in a fresh, modular, and highly parameterizable reimplementation. Finally, on the basis of empirical data obtained with PET, Act 4 provides a detailed comparison of competence and performance profiles obtained in October 1996 with the current development status (as of October 1999).

1 Competence & Performance Profiling

In system development and optimization, subtle algorithmic and implementational decisions often have a significant impact on system performance, so monitoring system evolution very closely is crucial. Developers should be enabled to obtain a precise record of the status of the system at any given point; also, comparison with earlier results, between various parameter settings, and across platforms should be automated and integrated with the regular development cycle. System performance, however, cannot be adequately characterized merely by measurements of overall processing time (and perhaps memory usage). Properties of (i) individual modules (in a classical setup, especially the unifier, type system, and parser), (ii) the grammar being used, and (iii) the input presented to the system all interact in complex ways. In order to obtain an analytical understanding of strengths and weaknesses of a particular configuration, finer-grained records are required. By the same token, developer intuition and isolated case studies are often insufficient, since in practice, people who have worked on a particular system or grammar for years still find that an intuitive prediction of system behaviour can be incomplete or plainly wrong.

Although most grammar development environments supply facilities to batch-process a test corpus and record the results produced by the system, these are typically restricted to processing a flat, unstructured input file (listing test sentences, one per line) and outputting a small number of processing results into a log file.[5] In total, we note a striking methodological and technological deficit in the area

[4]In the HPSG universe (and accordingly our present play) the term 'grammar' is typically used holistically, referring to the linguistic system comprised of (at least) the type hierarchy, lexicon, and rule apparatus.

[5](Meta-)Systems like PLEUK (Calder, 1993) and HDRUG (van Noord & Bouma, 1997) that facilitate the exploration of multiple descriptive formalisms and processing strategies come with slightly more sophisticated benchmarking facilities and visualization tools. However, they still largely operate on monolithic, unannotated input data sets, restrict

readings	number of complete analyses obtained (when applicable, after unpacking)
filter	percentage of parser actions predicted to fail (rule filter plus 'quick check')
etasks	number of attempts to instantiate an argument position in a rule
stasks	number of successful instantiations of argument positions in rules
aedges	number of active edges built by the parser (where appropriate)
pedges	number of passive edges built by the parser (typically in all-paths search)
unifications	number of top-level calls into the feature structure unification routine
copies	number of top-level feature structure copies made
tcpu	amount of cpu time (in milliseconds) spent in processing
space	amount of dynamic memory allocated during processing (in bytes)

Table 1: Some of the parameters making up a competence & performance profile.

of precise and systematic, let alone comparable, assessment of grammar and system behaviour.

Oepen and Flickinger (1998) propose a methodology, termed *grammar profiling*, that builds on structured and annotated collections of test and reference data (traditionally known as *test suites*). The *competence & performance profiling* approach we advocate in this play can be viewed as a generalization of this methodology — in line with the experimental paradigm suggested by, among others, Erbach (1991a) and Carroll (1994). A *competence & performance profile* is defined as a rich, precise, and structured snapshot of system behaviour at a given development point. The production, maintenance, and inspection of profiles is supported by a specialized software package (called [incr tsdb()][6]) that supplies a uniform data model, an application program interface to the grammar-based processing components, and graphical facilities for profile analysis and comparison. Profiles are stored in a relational database which accumulates a precise record of system evolution, and which serves as the basis for flexible report generation, visualization, and data analysis via basic descriptive statistics. All tables and figures used in this play were generated using [incr tsdb()].

The profiling environment defines a common set of descriptive metrics which aim both for in-depth precision and also for sufficient generality across a variety of processing systems. Most parameters are optional, though analysis potential may be restricted for partial profiles. Roughly, profile contents can be classified into information on (i) the processing *environment* (grammar, platform, versions, parameter settings and others), (ii) grammatical *coverage* (number of analyses, derivation and parse trees per reading, corresponding semantic formulae), (iii) *ambiguity* measures (lexical items retrieved, number of active and passive edges, where applicable, both globally and per result), (iv) *resource consumption* (various timings, memory allocation), and indicators of (v) *parser* and *unifier* throughput. Excluding relations and attributes that encode annotations on the input data, the current *competence & performance* database schema includes some one hundred attributes in five relations. Table 1 summarizes some of the profiling parameters as they are relevant to the drama to come.

While the current [incr tsdb()] data model has already been successfully adapted to six different parsing systems (with another two pending; see Act 5), it remains to be seen how well it scales to the description of a larger variety of processing regimes. And although absolute numbers must be viewed *cum grano salis*, the common metric has greatly increased comparability and data exchange among the groups mentioned above, and has in some cases also helped to identify unexpected sources of performance variation. For example, we have found that two Sun UltraSparc servers (at different sites) with identical hardware configuration (down to the level of cpu revision) and OS release reproducibly

accounting of system results to a small number of parameters (e.g. number of analyses, overall processing time, memory consumption, possibly the total number of chart edges), and only offer a limited, predefined choice of analysis views.

[6]See '`http://www.coli.uni-sb.de/itsdb/`' for the (draft) [incr tsdb()] user manual, pronunciation rules, and instructions on obtaining and installing the package.

Set	Aggregate	total items ♯	word string ϕ	lexical entries ϕ	total results ♯	parser analyses ϕ	passive edges ϕ	fs size ϕ
'tsnlp'	wellformed	1574	4·96	13·8	945	2·00	86	273
	illformed	2775	4·50	11·5	409	1·83	39	257
'csli'	wellformed	918	6·45	15·3	732	2·16	115	302
	illformed	375	6·11	14·9	85	2·31	84	298
'aged'	wellformed	96	8·41	23·1	72	7·00	292	315
'blend'	wellformed	1910	11·13	32·1	1008	51·39	1181	336
	illformed	142	11·05	34·2	24	20·33	611	317

Table 2: Reference data sets used in comparison and benchmarking with the LinGO grammar.

exhibit a performance difference of around ten per cent. This appears to be caused by different installed sets of vendor-supplied operating system patches. Also, average cpu load and availability of main memory have been observed to have a noticeable effect on cpu time measurements; therefore, all data reported in this play, was collected in an (artificial, in some sense) environment in which sufficient cpu and memory resources were guaranteed throughout each complete test run.

The [incr tsdb()] package includes a number of test suites and development corpora for English, German, and French (and has facilities for user-level import of additional test data). For benchmarking purposes with the LinGO grammar four test sets were chosen: (i) the English TSNLP test suite (Oepen, Netter, & Klein, 1997), (ii) the CSLI test suite derived from the original Hewlett-Packard data (Flickinger, Nerbonne, Sag, & Wasow, 1987), (iii) a small collection of transcribed scheduling dialogue utterances collected in the VerbMobil context, and (iv) a larger extract from recent VerbMobil corpora that was selected pseudo-randomly to achieve a balanced distribution of one hundred samples for each input length below twenty words. Some salient properties of these test sets are summarized in Table 2.[7] Looking at the degrees of lexical (i.e. the ratio between columns five and four), global (column seven), and local (approximated in column eight by the number of passive edges created in pure bottom-up parsing) ambiguity, the three test sets range from very short and unambiguous to mildly long and highly ambiguous. The *'blend'* test set is a good indicator of maximal input complexity that the available parsers can currently process (in plausible amounts of time and memory). Contrasting columns six and three (i.e. items accepted by the grammar vs. total numbers of well- or ill-formed items) provides a measure of grammatical coverage and overgeneration, respectively.

2 Strong Empiricism: A Few Examples

A fundamental measure in comparing two different versions or configurations of one system as well as for contrasting two distinct systems is correctness and equivalence of results. No matter what unification algorithm or parsing strategy is chosen, parameters like the numbers of lexical items retrieved per input word, total analyses found, passive edges derived (in non-predictive bottom-up parsing, at least) and others should only vary when the grammar itself is changed. Therefore, regular regression testing is required. In debugging and experimentation practice, we have found that minor divergences in results are often hard to identify; using an experimental parsing strategy, for example, over- and undergeneration can even out for the number of readings and even the accounting of passive edges. Hence, assuring an exact match in results (for a given test set) is a non-trivial task.

[7]While wellformedness and item length are properties of the test data proper, the indicators for average ambiguity and feature structure (fs) size were obtained using the current release version of the LinGO grammar, frozen in August 1999. Here and in the tables to come the symbol '♯' indicates absolute numbers, while 'ϕ' denotes average values.

The [incr tsdb()] package eases comparison of results on a per-item basis, using an approach similar to Un*x `diff(1)`, but generalized for structured data sets. By selection of a set of parameters for intersection (and optionally a comparison predicate), the user interface allows to browse the subset of items that fail to match in the selected properties. One dimension that we found especially useful in intersecting profiles is on the derivation trees (a bracketed structure labeled with rule names and identifiers of lexical items) associated with each parser analysis. Once a set of missing or extra derivations (representing under- or overgeneration, respectively) between two profiles is identified, they can be fed back into the defective parser as a request to try and reconstruct each derivation. Reconstruction of derivation trees, in a sense, amounts to fully deterministic parsing, and enables the processor to record where the failure occurs that caused undergeneration in the first place; conversely, when dealing with overgeneration, reconstruction in the correct parser can be requested to identify the missing constraint(s). While these techniques illustrate basic debugging facilities that the profiling and experimentation environment provides, the following two scenes discuss algorithmic issues in parser design and tuning that can only be addressed empirically.

2.1 Hyper-Active Parsing

The two established development platforms, the LKB (CSLI Stanford) and PAGE (DFKI Saarbrücken) systems, have undergone homogenization of approaches and even individual modules (the conjunctive PAGE unifier, for instance, was developed by Rob Malouf at CSLI Stanford) for quite a while.[8] Until recently, however, the parsing regimes deployed in the two systems were significantly different. Both parsers use quasi-destructive unification, are purely bottom-up, chart-based, perform no ambiguity packing, and can be operated in exhaustive (all paths) or agenda-driven best-first search modes; before any unification is attempted, both parsers apply the same set of pre-unification filters, viz. a test against a rule compatibility table (Kiefer et al., 1999), and the 'quick check' partial unification test (Malouf, Carroll, & Copestake, 2000). The LKB passive chart parser (in exhaustive mode) uses a breadth-first CKY-like algorithm; it processes the input string strictly from left to right, constructing all admissible complete constituents whose right vertex is at the current input position before moving on to the next lexical item. Attempts at rule application are made from right to left. All and only complete constituents found (passive edges) are entered in the chart. The active PAGE parser, on the other hand, uses a variant of the algorithm described by Erbach (1991b). It operates bidirectionally, both in processing the input string and instantiating rules; crucially, the *key* daughter (see Scene 2.2 below) of each rule is analyzed first, before the other daughter(s) are instantiated.

But while the LKB and the PAGE developers both assumed the strategy chosen in their own system was the best-suited for parsing with large feature structures (as exemplified by the LinGO grammar), the choices are motivated by conflicting desiderata. Not storing active edges (as in the passive LKB parser) reduces the amount of feature structure copying but requires frequent recomputation of partially instantiated rules, in that the unification of a daughter constituent with the rightmost argument position of a rule is performed as many times as the rule is applied to left-adjacent sequences of candidate chart edges. Creating active edges that add partial results to the chart, on the other hand, requires that more feature structure copies are made, which in turn avoids the necessity of redoing unifications. Given the effectiveness of the pre-unification filters it is likely that for some active edges no attempts to extend them with adjacent inactive edges will ever be executed, so that the copy

[8]Still, the two systems are by no means merely two instantiations of the same concept, and continue to focus on different application contexts. While the LKB is primarily used for grammar development and generation (in an AAC basic research project), recent PAGE develoment has emphasized efficient and robust parsing methods with speech recognizer output (in application to VerbMobil).

Set	Parser	filter %	etasks ϕ	stasks ϕ	unifs ϕ	copies ϕ	tcpu ϕ (s)	space ϕ (kb)
	passive	94·2	658	555	663	114	0·38	2329
'csli'	*active*	95·8	283	180	288	180	0·31	2432
	hyper-active	95·8	283	180	354	114	0·28	1686
	passive	94·2	1843	1604	1845	293	1·14	5692
'aged'	*active*	96·1	716	452	718	452	0·93	5449
	hyper-active	96·1	716	452	928	293	0·71	3830
	passive	93·6	9209	7968	9214	1074	5·87	16757
'blend'	*active*	96·0	2849	1580	2853	1580	3·42	13767
	hyper-active	96·0	2849	1580	4156	1074	3·31	10393

(generated by [incr tsdb()] at 3-nov-1999 (19:08 h)

Table 3: Contrasting parser performance: passive, active, and hyper-active in the LKB.

associated with the active edge was wasted effort. Profiling the two parsers individually showed that overall performance is roughly equivalent (with a minimal lead for the passive LKB parser in both time and space). While the passive parser executes far more parser tasks (i.e. unifications), it creates significantly fewer copies — as should be expected from what is known about the differences in parsing strategy. Hence, from a superficial comparison of parser throughput one could conclude that the passive parser successfully trades unifications for copies, and that both basic parsing regimes perform equally well with respect to the LinGO grammar.

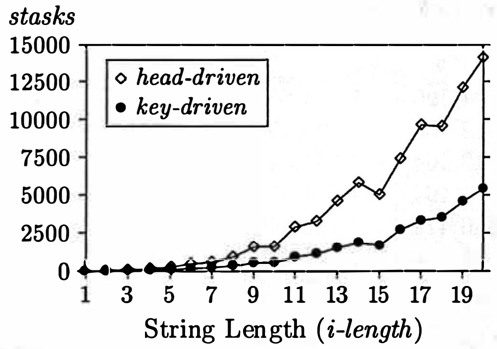
To obtain fully comparable results, the algorithm used in PAGE was imported into the LKB, which serves as the (single) experimentation environment for the remainder of this scene. The direct comparison is shown in Table 3 for three of the standard test sets. The re-implementation of the active parser in the LKB, in fact, performs slightly better than the passive version and does not allocate very much more space. On the *'aged'* test set, the active parser even achieves a modest reduction in memory consumption which most likely reflects the larger proportion of extra unifications compared to the savings in copies (columns five and six) for this test set. Having profiled the two traditional parsing strategies and dissected each empirically, it now seems natural to synthesize a new algorithm that combines the advantages of both strategies (i.e. reduced unification *and* reduced copying). The following algorithm, termed 'hyper-active' by Oepen and Carroll (2000), achieves this goal:

- use the bottom-up, bidirectional, key-driven control strategy of the active parser;
- when an 'active' edge is derived, store this partial analysis in the chart but do *not* copy the associated feature structure;[9]
- when an 'active' edge is extended (combined with a passive edge), recompute the intermediate feature structure from the original rule and already-instantiated daughter(s);
- only copy feature structures for complete passive edges; partial analyses are represented in the chart but the unification(s) that derived each partial analysis are redone on-demand.

Essentially, storing 'active' (or, in a sense, hyper-active) edges without creating expensive feature structure copies enables the parser to perform a key-driven search effectively, and at the same time avoids over-copying for partial analyses; additional unifications are traded for the copies that were avoided only where hyper-active edges are actually extended in later processing.[10]

[9]Although the intermediate feature structure is not copied, it is used to compute the 'quick-check' vector for the next argument position to be filled; as was seen already, this information is sufficient to filter the majority (i.e. up to ninety five per cent) of subsequent operations on the 'active' edge.

[10]There is an additional element — termed 'excursion' — to the algorithm proposed in Oepen and Carroll (2000)



PET (cheap)	**head-driven**	**key-driven**	Δ
filter (%)	93·2	95·5	—
etasks	8441	2995	64·5 %
stasks	4170	1465	64·9 %
tcpu (s)	1·47	0·64	56·4 %
space (kb)	7861	6472	17·7 %

Figure 1: Effects of head- vs. key-driven rule instantiation on parser work load (PET on '*blend*').

Table 3 confirms that hyper-active parsing combines the desirable properties of both basic algorithms: the number of copies made is exactly the same as for the passive parser, while the number of unifications is only moderately higher than for the active parser (due to on-demand recomputation of intermediate structures). Accordingly, average parse times are reduced by twenty six ('*csli*') and thirty seven ('*aged*') per cent, while memory consumption drops by twenty seven and thirty two per cent, respectively. Applying the three parsers to the much more challenging '*blend*' test set, reveals that the greater search space poses a severe problem for the passive parser, and limits the relative advantages of the hyper-active over the plain active strategy somewhat: while in the latter comparison the amount of copying is reduced by one third in hyper-active parsing, the number of unifications increases by thirty per cent at the same time (but see the discussion of rule instantiation below). Still, the hyper-active algorithm greatly reduces memory consumption, which by virtue of lower garbage collection times (not included in *tcpu* values) results in a significant overall speed-up. Compared to the original LKB passive parser, hyper-active parsing achieves a time and space reduction of forty three and thirty eight per cent, respectively. Thorough profiling and eclectic engineering have resulted in an improved parsing algorithm that is now used standardly in both the LKB and PAGE; for the German and Japanese VerbMobil grammars in PAGE, the observed benefits of hyper-active parsing were broadly confirmed.

2.2 Rule Instantiation Strategies

Head-driven approaches to parsing have been explored successfully with lexicalized grammars like HPSG (see van Noord, 1997, for a recent overview) because, basically, they can avoid proliferation of partial rule instantiations (i.e. active edges in a chart parser) with rules that contain very unspecific argument positions. Many authors either implicitly (Kay, 1989) or explicitly (Bouma & van Noord, 1993) assume the *linguistic head* to be the argument position that the parser should instantiate first. However, the right choice of argument position in each rule, such that it best constrains rule applicability (with respect to all categories derived by the grammar) cannot be determined analytically. Though the selection is likely to be related to the amount and specificity of information encoded for each argument, for some rules a single feature value (e.g. the [WH +] constraint on the non-head daughter in one of the instantiations of the filler–head schema used in LinGO) can be most important. For terminological clarity, PAGE introduces the term *key* daughter to refer to the argument position in each rule that is the best discriminator with respect to other categories that the grammar derives; at the same time, the notion of *key-driven* parsing emphasizes the observation that for individual rules in a particular grammar a non-(linguistic)head daughter may be a better candidate.

that aims to take advantage of the feature structure associated with an active edge while it is still valid (i.e. within the same unification generation). Put simply, the hyper-active parser is allowed to deviate slightly from the control strategy governed by the agenda, to try and combine the active edge with one suitable passive edge immediately.

Rule Name	head	key	aedges		pedges	ratio
			left→right	*right→left*		
HEAD – COMPLEMENT	left	left	84,396	1,404,652	264,137	3·13
SPECIFIER – HEAD	right	right	582,736	108,450	14,849	0·14
SUBJECT – HEAD	right	left	48,464	364,846	300,561	6·20
HEAD – MARKER	left	left	1,494	1,404,652	106,349	71·18
HEAD – ADJUNCT (*scopal*)	left	right	856,419	12,946	73,975	5·71
ADJUNCT – HEAD (*isective*)	right	left	34,482	1,260,660	37,343	1·08
ADJUNCT – HEAD (*scopal*)	right	left	11,177	1,260,660	119,513	10·69
FILLER – HEAD (*wh, subj*)	right	left	162	147,636	546	3·37

Table 4: Head and key positions and distribution of active vs. passive edges for selected rules.

Figure 1 compares parser performance (using the PET parser; see below) for a rule instantiation strategy that always fills the (linguistic) head daughter first (labelled '*head-driven*') with a variant that uses an idiosyncratically chosen key daughter for each rule (termed '*key-driven*'; see below for key selection). The data shows that the number of executed (*etasks*) as well as the number of successful (*stasks*) parser actions increase far more drastically with respect to input length in the head-driven setup (on the '*blend*' test suite, truncated above 20 words due to sparse data). Since parser tasks are directly correlated to overall parser performance, the key-driven strategy on average reduces parsing time by more than a factor of two. Clearly, for the LinGO grammar at least, linguistic headedness is not a good indicator for rule instantiation. Thus, the choice of good parsing keys for a particular grammar is an entirely empirical issue. Key daughters, in the current setup, are stipulated by the grammar engineer(s) as annotations to grammar rules; in choosing the key positions, the grammarian builds on knowledge about the grammar and observations from parsing test data. The [incr tsdb()] performance profiling tools can help in this choice since they allow the accounting of active and passive edges to be broken down by individual grammar rules (as they were instantiated in building edges). Inspecting the ratio of edges built per rule, for any given choice of parsing keys, can then help to identify rules that generate an unnecessary number of active edges. Thus, in the experimental approach to grammar and system optimization the effects of different key selections can be analyzed precisely and compared to earlier results.[11] Table 4 shows the head and key positions together with the differences in the number of active edges derived (in strict left to right vs. right to left rule instantiation modes) for a subset of binary grammar rules in LinGO. For the majority of head – argument structures (with the notable exception of the subject – head rule) the linguistic head corresponds to the key daughter, in adjunction and (most) filler – head constructions we see the reverse image; for some rules, choosing the head daughter as the key can result in an increase of active edges close to two orders of magnitude.

Inspecting edge proliferation by individual rules reveals another property of the particular grammar: the ratio of passive to active edges (column seven in Table 4, using the key-driven values for *aedges*) varies drastically. The specifier – head rule, for example, licenses a large number of active edges but, on average, only one out of seven active edges can be completed to yield a passive edge. The head – marker rule, on the other hand, on average generates seventy one passive edges from just one active edge. While the former should certainly benefit from hyper-active parsing, this seems very unlikely for the latter; Scene 2.1 above suggests that no more than three unifications should be traded for one copy in the LKB. Therefore, it seems plausible to apply the hyper-active parsing regime selectively to rules with a *pedges* to *aedges* ratio below a certain threshold *t*. We plan to explore this technique in

[11]For a given test corpus, the optimal set of key daughters can be determined (semi- or fully automatically) by comparing results for unidirectional left to right to pure right to left rule instantiation; the optimal key position for each rule is the one that generates the smallest number of active items.

a series of experiments, evaluating parser performance in relation to varied values for t.

3 PET — Synthesizing Current Best Practice

PET is a platform to build processing systems based on the descriptive formalism represented by the LinGO grammar. It aims to make experimentation with constraint-based parsers easy, including comparison of existing techniques and evaluating new approaches. Thus, flexibility and extendibility are primary design objectives. Both desiderata are achieved by a tool box approach — PET provides an extendible set of configurable building blocks, that can be combined and configured in different ways to instantiate a concrete processing system. The set of building blocks includes objects like *chart*, *agenda*, *grammar*, *type hierarchy*, and *typed feature structure*. Using the available objects, a simple bottom-up chart parser, for instance, can be realized in a few lines of code.

Alternative implementations of a certain object may be available to allow comparison of different approaches to one aspect of processing in a common context. For example, the current PET environment provides a choice of destructive, semi-destructive, and quasi-destructive implementations of the *typed feature structure* object (viz. the algorithms proposed by Wroblewski (1987), Ciortuz (2000), Tomabechi (1991), and Malouf et al. (2000); experimentation with additional algorithms and fixed-arity encodings is under development). In this setup properties of various graph unification algorithms and feature structure representations can be compared among each other and in interaction with different processing regimes.

In a parser called cheap, PET implements all relevant techniques from Kiefer et al. (1999) (i.e. conjunctive-only unification, rule filters, quick-check, restrictors), as well as techniques originally developed in other systems (e.g. key-driven parsing from PAGE, caching type unification and hyper-active parsing from the LKB, and partial expansion from CHIC). Re-implementation and strict modularization often resulted in improved representations and algorithmic refinement; since individual modules can be specialized for a particular task, the overhead often found in monolithic implementations (like slots in internal data structures, say, that are only required in a certain configuration) could be reduced.

Efficient memory management and minimizing memory consumption was another important consideration in the development of PET. Experience with Lisp-based systems suggests that memory throughput is one of the main bottlenecks when processing large grammars. In fact, one observes a close correlation between the amount of dynamically allocated memory and processing time, indicating much time is spent moving data, rather than in actual computation. Using builtin C++ memory management, allocation and release of feature structure nodes can account for up to forty per cent of total run time. Like in the WAM (Aït-Kaci, 1991), a general memory allocation scheme allowing arbitrary order of allocation and release of structures is not necessary in this context. Within a larger unit of computation, the application of a rule, say, the parser typically builds up structure monotonically; memory is only released in the case of a top-level unification failure when all partial structure built during this unification is freed. Therefore, PET employs a simple stack-based memory management strategy, acquiring memory from the operating system in large chunks which are then sub-allocated. A *mark–release* mechanism allows saving the current allocation state (the current stack position) and returning to that saved state at a later point. Thus, releasing a chunk of objects amounts to a single pointer assignment.

Also, feature structure representations are maximally compact.[12] In combination with other memory-reducing techniques (e.g. partial expansion and shrinking, substructure sharing, hyper-active

[12]The size of one dag node in the PET implementation of Tomabechi (1991) is only twenty four bytes, compared to, for example, fifty six in the Lisp-based LKB system.

Test Set	test items ♯	October 1996 lexical ϕ	parser ϕ	in %	out %	August 1999 lexical ϕ	parser ϕ	in %	out %
***'tsnlp'* test set**	**4463**	**2·32**	**1·75**	**65·3**	**26·7**	**2·67**	**2·21**	**76·7**	**26·5**
S_Types	174	2·70	2·16	78·7	40·0	3·37	1·24	96·0	51·6
C_Agreement	123	2·59	1·33	58·8	10·0	2·27	1·28	77·9	10·0
C_Complementation	1010	2·45	2·19	62·2	12·1	2·99	1·67	83·1	10·5
C_Diathesis-Passive	220	3·58	2·87	25·3	8·1	3·52	3·52	50·5	6·3
NP_Agreement	1196	1·56	1·06	47·8	14·8	1·70	1·21	62·2	15·9
Other	1740	2·28	1·72	73·2	54·9	2·70	2·66	79·9	53·3
***'aged'* test set**	**95**	**2·11**	**2·55**	**65·8**	—	**2·74**	**7·00**	**75·0**	—

(generated by [incr tsdb()] at 5-nov-1999 (17:11 h))

Table 5: Development of grammatical coverage and overgeneration over three years.

parsing) this results in very attractive memory consumption characteristics for the cheap parser, allowing to process the *'blend'* test set with a process size of around one hundred megabytes (where Lisp- or Prolog-based implementations easily grow beyond half a gigabyte). To maximize compactness and efficiency, PET is implemented in ANSI C++, but uses traditional C representations (rather than C++ objects) for some central objects where minimal overhead is required (e.g. the basic features structure elements).

4 Quantifying Progress

The preceding acts have exemplified the benefits of competence and performance profiling in the application to isolated properties of various parsing algorithms. In this final act we take a wider perspective and use the profiling approach to give an impression of overall progress made in processing the LinGO grammar over a development period of three years. The oldest available profiles (for the *'tsnlp'* and *'aged'* test sets) were obtained with PAGE (version 2·0 released in May 1997) and the October 1996 version of the grammar; the current best parsing performance, to our best knowledge, is achieved in the cheap parser of PET. All data was sampled on the same Sun UltraSparc server (dual 300 Mhz; 1.2 gbytes memory; mildly patched Solaris 2.6) at Saarbrücken.

The evolution of grammatical coverage is depicted in Table 5, contrasting salient properties from the individual competence profiles (see Table 2) side by side; to illustrate the use of annotations on the test data, the table is further broken down by selected syntactic phenomena for the TSNLP data (Oepen et al., 1997, give details of the phenomenon classification). Comparison of the *lexical* and *parser* averages shows a modest increase in lexical but a dramatic increase in global ambiguity (by close to a factor of three for *'aged'*). Columns labeled *in* and *out* indicate coverage of items marked wellformed and overgeneration for ill-formed items, respectively. While the *'aged'* test set does not include negative test items, it confirms that coverage within the VerbMobil domain has improved. However, the TSNLP test suite is far better suited to gauge development of grammatical coverage, since it was designed to systematically exercise different modules of the grammar. In fact, a net increase in coverage (from sixty five to seventy seven per cent) in conjunction with slightly reduced overgeneration confirms that the LinGO grammar engineers have steadily improved the overall quality of the linguistic resource.

The assessment of parser performance shows a more dramatic development. Average parsing times per test item have dropped by more than two orders of magnitude (a factor of one hundred and fifty on the *'aged'* data), while memory consumption was reduced to about two per cent of the original values.

Version	Platform	Test Set	filter %	etasks ϕ	pedges ϕ	tcpu ϕ (s)	space ϕ (kb)
October 1996	PAGE	*'tsnlp'*	49·9	656	44	3·69	19016
		'aged'	51·3	1763	97	21·16	79093
August 1999	PET (cheap)	*'tsnlp'*	93·9	170	55	0·03	333
		'aged'	95·1	753	292	0·14	1435
		'blend'	95·5	3084	1140	0·65	10589

(generated by [incr tsdb()] at 5-nov-1999 (21:23 h))

Table 6: Development of salient performance parameters (PAGE VS. PET) over three years.

Because in the early PAGE data the 'quick check' pre-unification filter was not available, current filter rates for PET (and the other systems alike) are much better and result in a reduction of parser tasks that are actually executed. At the same time, comparing the number of passive edges licensed by the two versions of the grammar, provides a good estimate on the size of the search space processed by the two parsers. Although for the (nearly) ambiguity-free TSNLP test suite the *pedges* averages are almost stable, the *'aged'* data shows an increase by a factor of three. Asserting that the average number of passive edges is a direct measure for input complexity (with respect to a particular grammar), we extrapolate the overall speed-up in processing the LinGO grammar as a factor of roughly five hundred (again, *tcpu* values in Table 6 do *not* include garbage collection for PAGE which in turn is avoided in PET). Finally, Table 6 includes PET results on the currently most challenging *'blend'* test set (see above). Despite of greatly increased search space and ambiguity, the cheap parser achieves an average parse time of 650 milliseconds and processes almost ninety per cent of the test items in less than one second.[13]

5 Conclusion, Outlook, and Acknowledgments

Precise, in-depth comparison has enabled a large, multi-national group of developers to quantify and exchange algorithmic knowledge and benefit from each others experience. The [incr tsdb()] profiling package has been integrated with six processing environments for (HPSG-type) unification grammars so far; developers of other systems are encouraged to seek assistance in interfacing to the common metric (connections to the Alvey Tools GDE and Xerox XLE are currently under consideration) — scaling up and generalizing the set of competence and performance parameters is, once more, an empirical challenge. In parallel, the range of experimental choices in PET will be increased, aiming for import (and adaptation) of recent results, especially in the areas of fixed-arity feature structure encodings (inspired by the Tokyo LiLFeS implementation) and ambiguity packing (from the LKB).

The cathartic effect achieved in this period of close collaboration between sites over several years would not have been possible without the main characters, researchers, engineers, grammarians, and managers at Saarbrücken, Stanford, Tokyo, and Sussex University. To name a few, John Carroll, Liviu Ciortuz, Ann Copestake, Dan Flickinger, Bernd Kiefer, Hans-Ulrich Krieger, Takaki Makino, Rob Malouf, Yusuke Miyao, Stefan Müller, Mark-Jan Nederhof, Günter Neumann, Takashi Ninomiya, Kenji Nishida, Ivan Sag, Kentaro Torisawa, Jun'ichi Tsujii, and Hans Uszkoreit have all greatly contributed to the development of efficient HPSG processors as described above. Many of the individual achievements and results are reflected in the bibliographic references given throughout the play.

[13]To obtain the results on the *'blend'* test set shown in Table 6, an upper limit on the number of passive edges was imposed in the cheap parser; with a permissible maximum of twenty thousand edges, around fifty (in a sense pathological) items from the *'blend'* set cannot be processed within the limit and, accordingly, are excluded in the overall assessment. Maximal parsing times for the remaining test items range to around fourteen seconds for input strings that approximate twenty thousand edges and derive a very large number of readings.

References

Aït-Kaci, H. (1991). *Warren's Abstract Machine: A tutorial reconstruction.* Cambridge, MA: MIT Press.

Bouma, G., & van Noord, G. (1993). Head-driven parsing for lexicalist grammars. Experimental results. In *Proceedings of the 6th Conference of the EACL.* Utrecht, The Netherlands.

Calder, J. (1993). Graphical interaction with constraint-based grammars. In *Proceedings of the 3rd Pacific Rim Computational Linguistics Conference* (pp. 160 – 169). Vancouver, BC.

Callmeier, U. (2000). PET — A platform for experimentation with efficient HPSG processing techniques. In *(Flickinger et al., 2000).*

Carpenter, B. (1992). *The logic of typed feature structures.* Cambridge, UK: Cambridge University Press.

Carroll, J. (1994). Relating complexity to practical performance in parsing with wide-coverage unification grammars. In *Proceedings of the 31st meeting of the ACL* (pp. 287 – 294). Las Cruces, NM.

Carroll, J., Copestake, A., Flickinger, D., & Poznanski, V. (1999). An efficient chart generator for (semi-)lexicalist grammars. In *Proceedings of the7th European Workshop on Natural Language Generation.* Toulouse, France.

Ciortuz, L. (2000). *Compiling HPSG into C.* In preparation, DFKI, Saarbrücken, Germany.

Copestake, A. (1992). The ACQUILEX LKB. Representation issues in semi-automatic acquisition of large lexicons. In *Proceedings of ANLP 1992* (pp. 88 – 96). Trento, Italy.

Erbach, G. (1991a). An environment for experimenting with parsing strategies. In J. Mylopoulos & R. Reiter (Eds.), *Proceedings of IJCAI 1991* (pp. 931 – 937). San Mateo, CA: Morgan Kaufmann Publishers.

Erbach, G. (1991b). A flexible parser for a linguistic development environment. In O. Herzog & C.-R. Rollinger (Eds.), *Text understanding in LILOG* (pp. 74 – 87). Berlin: Springer.

Flickinger, D., Nerbonne, J., Sag, I. A., & Wasow, T. (1987). *Toward evaluation of NLP systems* (Tech. Rep.). Hewlett-Packard Laboratories.

Flickinger, D., Oepen, S., Uszkoreit, H., & Tsujii, J. (Eds.). (2000). *Journal of Natural Language Engineering. Special Issue on Efficient processing with HPSG: Methods, systems, evaluation.* Cambridge, UK: Cambridge University Press. (in preparation)

Flickinger, D. P., & Sag, I. A. (1998). Linguistic Grammars Online. A multi-purpose broad-coverage computational grammar of English. In *CSLI Bulletin 1999* (pp. 64 – 68). Stanford, CA: CSLI Publications.

Kay, M. (1989). Head-driven parsing. In *Proceedings of International Parsing Technology Workshop* (pp. 52 – 62). Pittsburgh, PA.

Kiefer, B., Krieger, H.-U., Carroll, J. A., & Malouf, R. (1999). A bag of useful techniques for efficient and robust parsing. In *Proceedings of the 37th meeting of the ACL* (pp. 473 – 480). Baltimore, MD.

Krieger, H.-U., & Schäfer, U. (1994). *TDL* — A type description language for constraint-based grammars. In *Proceedings of the 15th International Conference on Computational Linguistics COLING* (pp. 893 – 899). Kyoto, Japan.

Malouf, R., Carroll, J., & Copestake, A. (2000). Efficient feature structure operations without compilation. In *(Flickinger et al., 2000).*

van Noord, G. (1997). An efficient implementation of the head-corner parser. *Computational Linguistics, 23,* 425 – 456.

van Noord, G., & Bouma, G. (1997). HDRUG. a flexible and extendible development environment for natural language processing. In *Proceedings of the EACL/ACL workshop ENVGRAM.* Madrid, Spain.

Oepen, S., & Carroll, J. (2000). Performance profiling for parser engineering. In *(Flickinger et al., 2000).*

Oepen, S., & Flickinger, D. P. (1998). Towards systematic grammar profiling. Test suite technology ten years after. *Journal of Computer Speech and Language, 12 # 4 (Special Issue on Evaluation),* 411 – 436.

Oepen, S., Netter, K., & Klein, J. (1997). TSNLP — Test Suites for Natural Language Processing. In J. Nerbonne (Ed.), *Linguistic databases.* Stanford, CA: CSLI Publications.

Tomabechi, H. (1991). Quasi-destructive graph unification. In *Proceedings of the 29th meeting of the ACL* (pp. 315 – 322). Berkeley, CA.

Torisawa, K., & Tsujii, J. (1996). Computing phrasal signs in HPSG prior to parsing. In *Proceedings of COLING 1996* (pp. 949 – 955). Kopenhagen, Denmark.

Uszkoreit, H., Backofen, R., Busemann, S., Diagne, A. K., Hinkelman, E. A., Kasper, W., Kiefer, B., Krieger, H.-U., Netter, K., Neumann, G., Oepen, S., & Spackman, S. P. (1994). DISCO — an HPSG-based NLP system and its application for appointment scheduling. In *Proceedings COLING 1994.* Kyoto, Japan.

Wroblewski, D. A. (1987). Nondestructive graph unification. In *Proceedings of the 6th National Conference on Artificial Intelligence* (pp. 582 – 587). Seattle, WA.

COMPUTING THE MOST PROBABLE PARSE FOR A DISCONTINUOUS PHRASE STRUCTURE GRAMMAR

Oliver Plaehn*
Department of Computational Linguistics
University of the Saarland
66041 Saarbrücken, Germany
plaehn@coli.uni-sb.de

Abstract

This paper presents a probabilistic extension of Discontinuous Phrase Structure Grammar (DPSG), a formalism designed to describe discontinuous constituency phenomena adequately and perspicuously by means of trees with crossing branches. We outline an implementation of an agenda-based chart parsing algorithm that is capable of computing the Most Probable Parse for a given input sentence for probabilistic versions of both DPSG and Context-Free Grammar. Experiments were conducted with both types of grammars extracted from the NEGRA corpus. In spite of the much greater complexity of DPSG parsing in terms of the number of (partial) analyses that can be constructed for an input sentence, accuracy results from both experiments are comparable. We also briefly hint at future lines of research aimed at more efficient ways of probabilistic parsing with discontinuous constituents.

1 Introduction

Natural languages exhibit a good deal of discontinuous constituency phenomena, especially with regard to languages with a relatively free word order like German or Dutch, that cannot be described adequately by context-free phrase structure trees alone. Arguments from linguistics motivate the representation of structures containing discontinuous constituents by means of trees with crossing branches (McCawley, 1982; Blevins, 1990; Bunt, 1996). Bunt (1991, 1996) proposed a formalism called *Discontinuous Phrase Structure Grammar* (DPSG) for use in generation of and parsing with discontinuous trees and outlined an active chart parsing algorithm for DPSGs.

DPSG is quite a natural and straightforward extension of common Context-Free Grammar (CFG). Hence, it is reasonable to assume that *probabilistic* approaches to parsing with CFGs might also be extended with similar ease to deal with crossing branches. In order to test this hypothesis, we adapt the algorithm for computing the Most Probable Parse (MPP) for a sentence given a probabilistic CFG (PCFG) to perform the same task for a DPSG enhanced with rule probabilities (PDPSG) and report on experiments conducted with this algorithm based on grammars extracted from the NEGRA corpus (Skut, Krenn, Brants, & Uszkoreit, 1997). It turns out that accuracy results from PDPSG experiments are comparable to those obtained in PCFG experiments on the same corpus.

In Sections 2 and 3, we define precisely what we mean by a discontinuous tree and a DPSG and present our chart parser for DPSGs. The theoretical foundations of our MPP algorithm for DPSG are discussed in Section 4, in which we also explain how the MPP is computed by a slightly modified version of our parser. Section 5 reports on experiments conducted with grammars extracted from a

*The author is grateful to Thorsten Brants and Alexander Koller for valuable comments and discussion, and also to Nicole Siekmann for proof-reading an earlier version of this paper.

corpus of discontinuous trees and discusses the results we obtained. The paper[1] concludes with a brief summary and an outline of possible future lines of research.

2 Discontinuous Phrase Structure Grammar

The formalism of Discontinuous Phrase Structure Grammar (DPSG) is originally due to Bunt (1991, 1996). Below, we slightly deviate from Bunt's account in order to have suitable formal definitions on which we can base our algorithm for computing the MPP. However, the motivation and ideas underlying DPSG are preserved.

We start with a recursive definition of discontinuous trees (or discotrees for short). Discotrees consist of substructures which themselves are not necessarily valid discotrees. Consider, for instance, the discotree (a) in Figure 1 that contains the two substructures in (b) and (c).

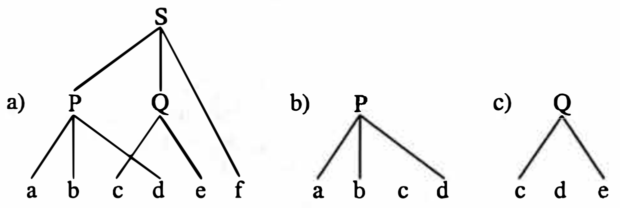


Figure 1: The discotree in (a) contains substructures (b) and (c), which are not valid discotrees

We thus need a definition of these substructures first, which we provide below.

Definition 1 (Subdiscotree)
Let N be a non-empty set of nodes. A subdiscotree *is recursively defined as follows.*

1. *If $x \in N$, then the pair $\langle x, [\,]\rangle$ is a subdiscotree; such a subdiscotree is called* atomic.

2. *If $x \in N$ and $X_1, \ldots, X_n$ $(n \geq 1)$ are subdiscotrees that do not share any subdiscotrees[2], then the pair $\langle x, [X_1, X_2^{c_2}, \ldots, X_{n-1}^{c_{n-1}}, X_n]\rangle$ is a subdiscotree, where $c_i \in \{0,1\}$ for $1 < i < n$.*

3. *No other structures than those defined by (1) and (2) are subdiscotrees.* ■

In a subdiscotree $\langle x, [X_1, X_2^{c_2}, \ldots, X_{n-1}^{c_{n-1}}, X_n]\rangle$, the node x *immediately dominates* the top nodes of X_1, X_n, and of every $X_i^{c_i}$ such that $c_i = 0$. In other words, subdiscotrees consist of a top node x and a sequence of daughter constituents whose top nodes are immediately dominated by x, possibly interrupted by constituents whose top nodes are not dominated by x. These latter nodes are called *internal context* of x or *context daughters* of x. As shorthand notation for $\langle x, [X_1, \ldots, X_n]\rangle$, we use $x(X_1, \ldots, X_n)$ and enclose context daughters in brackets instead of using superscripts for this purpose. For example, the structure in Figure 1a is a graphical representation of the subdiscotree S(P(a, b, [c], d), Q(c, [d], e), f).

Let I denote the immediate dominance relation. *Dominance* (D) is defined as the reflexive and transitive closure of I, as usual. A *discotree* is now simply a subdiscotree T in which every node except the root is immediately dominated by some other node in T.

[1]Due to limitations of space, parts of our exposition are rather brief; a more detailed account can be found in (Plaehn, 1999).
[2]Two subdiscotrees X and Y share a subdiscotree Z, if both X and Y contain a node dominating the top node of Z. The inclusion of this condition ensures that subdiscotrees do not contain multidominated nodes.

Next, we provide a suitable definition of linear precedence in discotrees. To this end, we first need two auxiliary definitions. The *leftmost daughter* of a subdiscotree $T = x(X_1, \ldots, X_n)$, $Lm(T)$, is the top node of X_1; if $T = \langle x, [\,]\rangle$ is atomic, then $Lm(T) = x$. The *leaf sequence* of a discotree T, $LeafSeq(T)$, is the sequence of terminal nodes in T's formal representation in left to right order; if a leaf appears more than once, only its leftmost appearance is included in the sequence. The leaf sequence of $\mathrm{S(P(a,b,[c],d),Q(c,[d],e),f)}$ is thus $\langle \mathrm{a,b,c,d,e,f}\rangle$.

Definition 2 (Linear precedence (<))
Linear precedence *($<$) in a discotree A with LeafSeq(A) = $\langle x_1, \ldots, x_n\rangle$ is defined as follows.*

1. *For two leaves x_i and x_j in LeafSeq(A), $x_i < x_j$ if and only if $i < j$.*
2. *For two arbitrary nodes x and y in A, $x < y$ if and only if*
   (a) $Lm(x) < Lm(y)$ and
   (b) $\exists$ node z: $Lm(x) < z \leq Lm(y) \ \wedge\ \neg(xDz)$. ■

In words, the leaves of a discotree are totally ordered according to $<$. For two arbitrary nodes x and y, x precedes y if and only if the leftmost daughter of x precedes the leftmost daughter of y and there exists a node z in between $Lm(x)$ and $Lm(y)$ or identical to the latter, such that x does not dominate z. In the discotree in Figure 1a, we thus have a<b, a<c, a<d, a<e, a<f, b<c, b<d, b<e, b<f, c<d, c<e, c<f, d<e, d<f, e<f, P<c, P<d, P<e, P<f, a<Q, b<Q, Q<d, Q<e, Q<f, P<Q. Note that both P<d and PId holds; that is, there is no "Exclusivity Condition". This is necessary because we want a node n to precede its first context daughter (and $<$ to be a strict partial order) since the latter might be the leftmost daughter of a different constituent that is preceded by n. We say that two nodes x and y are *adjacent* (denoted by $x + y$) if x precedes y and there is no node z such that $x < z < y$. In our example discotree, the following adjacency relations hold: a+b, b+c, c+d, d+e, e+f, P+c, b+Q, P+Q, Q+d.

A CFG rule is used to rewrite its left-hand side category as a sequence of pairwise adjacent constituents. If we directly apply this concept to the generation of discotrees, we encounter a problem. To see this, consider again the discotree in Figure 1a which we would like to be generated by the rules in (1).

(1) S → P Q f	(2) S → P Q [d] [e] f
P → a b [c] d	P → a b [c] d
Q → c [d] e	Q → c [d] e

But the first rule is not applicable, since Q and f are not adjacent. We would thus be forced to use the rules in (2) which are quite awkward and counterintuitive. To eliminate this problem, we relax the condition that the symbols on the right-hand side of a rule have to be pairwise adjacent and instead require them to form an adjacency sequence, as defined below.

Definition 3 (Adjacency sequence)
A sequence of nodes $\langle x_1, x_2, \ldots, x_n\rangle$ in a subdiscotree A is an adjacency sequence *if and only if*

1. $\forall 1 \leq i < n$: $\big(x_i + x_{i+1} \ \vee (\exists$ nodes $y_1, \ldots, y_m$ in A: $x_i + y_1 \ \wedge\ y_1 + y_2 \ \wedge\ \ldots\ \wedge\ y_{m-1} + y_m \ \wedge y_m + x_{i+1} \ \wedge\ \forall 1 \leq j \leq m$: $\exists 1 \leq k \leq i$: $x_k D y_j)\big)$ *and*
2. $\forall 1 \leq i < j \leq n$: $\neg(\exists$ node z in A: $x_i D z \ \wedge\ x_j D z)$. ■

In words, we require in clause (1) that every pair $\langle x_i, x_{i+1}\rangle$ in the sequence is either an adjacency pair or is connected by a sequence of adjacency pairs of which all members are dominated by some element in the subsequence $\langle x_1, \ldots, x_i\rangle$ and in clause (2) that the elements of the sequence do not share any constituents. Thus, $\langle \mathrm{P}, \mathrm{Q}, \mathrm{f}\rangle$ constitutes an adjacency sequence since P and Q are adjacent, Q and f are connected by the sequence of adjacency pairs Q+d, d+e, e+f, and d and e are dominated by P and Q, respectively. Furthermore, P, Q and f do not share any constituents. We can therefore use the rules in (1) to generate the discotree in Figure 1a.

A Discontinuous Phrase Structure Grammar (DPSG) then essentially consists of a set of rules which can be used in generating a discotree such that the nodes corresponding to the symbols on the rules' right-hand sides form adjacency sequences.

Definition 4 (Discontinuous Phrase Structure Grammar)
A Discontinuous Phrase Structure Grammar (DPSG) *is a quadruple* $\langle V_N, V_T, S, R\rangle$, *where*

- V_N *is a finite set of* nonterminal symbols;
- V_T *is a finite set of* terminal symbols; *let* V *denote* $V_N \cup V_T$;
- $S \in V_N$ *is the distinguished* start symbol *of the grammar;*
- R *is a finite set of* rules $X \rightarrow Y^{1,c_1}Y^{2,c_2}\ldots Y^{n,c_n}$, *where* $X \in V_N$, $Y^1, \ldots, Y^n \in V$, *and* $c_i \in \{0,1\}$ $(1 \leq i \leq n)$ *indicates whether* Y^i *is a context daughter in the rule. Since neither the first nor the last daughter may be marked as internal context,* $c_1 = c_n = 0$. ■

This definition makes apparent the close similarity between CFG and DPSG. In fact, a DPSG not containing rules with context daughters degenerates to a CFG.

3 The Parsing Algorithm

Bunt (1991) outlined an active chart parsing algorithm for DPSG that constructs discotrees bottom-up (see also (van der Sloot, 1990)). Here, we present an agenda-based chart parser that provides us with greater flexibility with respect to the order in which edges are to be processed.

The algorithm makes use of three data structures. *Edges* correspond to (partial) parse trees (sub-discotrees, that is) and an *agenda* stores edges considered for combination with other edges already residing on the *chart*. An edge consists of the starting and ending position of its corresponding sub-parse, a pointer to the DPSG rule which has been used to construct the edge, and the number of its right-hand side symbols for which constituents have already been found (fields `start`, `end`, `rule` and `dot_pos`, resp.). This information alone, though, is not sufficient to uniquely describe a possibly discontinuous constituent. We also need to know which terminal symbols are dominated by direct daughters of the edge and which by context daughters. To this end, each edge is additionally associated with two bit-strings (fields `covers` and `ctxt_covers`)[3].

For instance, during the construction of the discotree in Figure 1a given the DPSG in (1), our parser creates the *inactive* edge [0,4,P $\rightarrow$ a b [c] d •, 110100, 001000] for the subparse in Figure 1b. Later on, it also builds the edge [0,4,S $\rightarrow$ P • Q f, 110100, 000000], which is called *active* since

[3]The use of bit-strings for representing locations of discontinuous constituents is motivated by (Johnson, 1985).

it still needs to be combined with constituents for Q and f. Notice that the input symbol c is not yet dominated by a direct or a context daughter of the edge. Therefore, the third bit in both covers (110100) and ctxt_covers (000000) is unset (equals 0).

The core of the parsing algorithm works as follows.

```
for $i = 0$ to $n - 1$ do
    $edge$ ← new edge [$i, i+1, t_i \rightarrow \bullet, 0^i\ 1\ 0^{n-i-1}, 0^n$];
    agenda.add( $edge$ );
while not agenda.is_empty() do
    $edge$ ← agenda.get_next();
    chart.add($edge$);
if chart.success($goal$) then output parse(s);
```

During initialization, edges corresponding to the terminal symbols $t_0, \ldots, t_{n-1}$ of the input sentence are added to the agenda. Edges are then popped off the agenda and added to the chart, one after the other. In the process of adding an active edge to the chart, it is combined with all matching inactive edges already on the chart, thus giving rise to new edges, which are added to the agenda. Likewise, an inactive edge popped from the agenda is combined with suitable active ones on the chart and, in addition, gives rise to new edges based on matching DPSG rules. This is repeated until the agenda is empty. If the chart then contains an inactive edge that is headed by the goal category and spans the entire input (that is, if *edge*.covers $= 1^n$), it corresponds to one or more complete parse trees for the input sentence.

Let us make the pecularities due to dealing with discotrees instead of ordinary context-free trees more precise. Firstly, what conditions must hold so that an active edge *ae* and an inactive edge *ie* can be combined? The left-hand side category of *ie* (*ie*.lhs) must match the symbol to the right of the dot on *ae*'s right-hand side (*ae*.next_cat). Furthermore, in order to ensure that the constituents corresponding to the right-hand side symbols of *ae* form an adjacency sequence, the next edge to be combined with *ae* has to start at the position of the first terminal symbol within *ae*'s span that is neither dominated by a direct nor by a context daughter of *ae*. We therefore compute the bit-wise 'or' ($\vee$) of *ae*.covers and *ae*.ctxt_covers restricted to the interval [*ae*.start, *ae*.end], set *ae*.next_pos to the position of the first unset bit in this bit-string, and require that *ie*.start = *ae*.next_pos holds. Additionally, we need to check that the two edges do not share any constituents. This is the case, if the bit-wise 'and' of *ae*.covers and *ie*.covers and of *ae*.ctxt_covers and *ie*.covers is zero. To summarise,

(3) $ae.\texttt{next\_pos} = ie.\texttt{start}$, $\quad ae.\texttt{next\_cat} = ie.\texttt{lhs}$,
$ae.\texttt{covers} \wedge ie.\texttt{covers} = 0$, and $ae.\texttt{ctxt\_covers} \wedge ie.\texttt{covers} = 0$

must hold so that *ae* and *ie* can be combined, resulting in a new edge n with

(4)
$$\begin{aligned}
n.\texttt{start} &\leftarrow ae.\texttt{start}\\
n.\texttt{end} &\leftarrow \max\{ae.\texttt{end}, ie.\texttt{end}\}\\
n.\texttt{rule} &\leftarrow ae.\texttt{rule}\\
n.\texttt{dot\_pos} &\leftarrow ae.\texttt{dot\_pos} + 1\\
n.\texttt{covers} &\leftarrow \begin{cases} ae.\texttt{covers} & \text{if } ie \text{ is context daughter}\\ ae.\texttt{covers} \vee ie.\texttt{covers} & \text{otherwise}\end{cases}
\end{aligned}$$

$$n.\texttt{ctxt\_covers} \leftarrow \begin{cases} ae.\texttt{ctxt\_covers} \vee ie.\texttt{covers} & \text{if } ie \text{ is context daughter} \\ ae.\texttt{ctxt\_covers} & \text{otherwise.} \end{cases}$$

Suppose, for example, that the inactive edge $ie =$ [2, 5, Q $\rightarrow$ c [d] e $\bullet$, 001010, 000100] has just been popped from the agenda and added to the chart, and that the active edge $ae =$ [0, 4, S $\rightarrow$ P $\bullet$ Q f, 110100, 000000] is already contained in the chart. We thus check whether these two edges can be combined. $ae.$next_pos is 2 (we start counting at 0) and equals $ie.$start. The left-hand side category of ie and the symbol to the right of the dot in ae are both Q. Furthermore, $ae.\texttt{covers} \wedge ie.\texttt{covers} = 110100 \wedge 001010 = 0$ and $ae.\texttt{ctxt\_covers} \wedge ie.\texttt{covers} = 000000 \wedge 001010 = 0$. All necessary conditions are fulfilled, so ae and ie are combined, yielding the new edge [0, 5, S $\rightarrow$ P Q $\bullet$ f, 111110, 000000], which is added to the agenda. Additionally, ip is combined with each rule in the given DPSG whose left-corner category equals the left-hand side category of ip. Suppose that the input DPSG contains a rule X $\rightarrow$ Q R. This would result in a new edge [2, 2, X $\rightarrow$ $\bullet$ Q R, 000000, 000000] to be added to the agenda[4].

The mechanisms described above are sufficient to ensure that constituents corresponding to right-hand side symbols of edges constructed by our parser form adjacency sequences, as is required by the DPSG formalism. One problem remains, though. Consider the DPSG in Figure 2a.

a) S $\rightarrow$ a P X
P $\rightarrow$ b [Y] e
X $\rightarrow$ c d
Y $\rightarrow$ c d

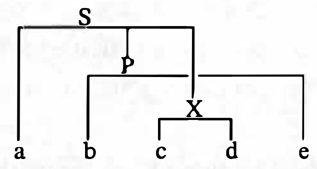
b)


Figure 2: The discotree in (b) should not be constructed from the DPSG in (a)

The parser would construct the discotree shown in Figure 2b, even though the P constituent assumes a context daughter Y dominating the input symbols c and d, whereas the direct daughter corresponding to Y in span is headed by an X. To put it differently, the X constituent is not licensed as a context daughter of P according to rule P $\rightarrow$ b [Y] e. In order to prevent the construction of such ill-formed structures, our algorithm additionally checks that context daughters and corresponding direct daughters match in category[5].

In order to be able to reconstruct all (partial) parse trees corresponding to an edge e, we associate with e a list of pairs of edge pointers (field children), each pair representing one possible way in which e has been constructed from an active and an inactive edge. When a new edge n is to be added to the chart, we check whether an edge e corresponding to an equivalent subparse already exists. If yes, the children list of n is appended to e's list, and n is discarded. Two edges correspond to equivalent subparses if they would behave in the same way would parsing proceed without them being merged. It turns out that e and n can only be merged safely, if they both correspond to a continuous constituent[6]. Additionally, if e and n are inactive, they have to agree in the values of their fields start, end and lhs. If the two edges are active, their fields start, end, rule and dot_pos have to contain the same values.

[4]To improve efficiency, we perform a lookahead test for each new active edge n to check whether the remaining symbols on n's right-hand side can be "satisfied" at all with the input symbols not already dominated by n. If not, we discard n. The details of this test are beyond the scope of this paper.

[5]As far as we can tell from the information available in (Bunt, 1991) and (van der Sloot, 1990), Bunt's parser would build the discotree in Figure 2b when presented with the DPSG in Figure 2a.

[6]We say that an edge e is *continuous*, if and only if $\prod_{i=e.\texttt{start}}^{e.\texttt{end}-1} b_i = 1$ where b_i is the i-th bit in $e.$covers.

Note, finally, that the parsing algorithm described above constructs context-free trees when presented with a CFG as input. The additional mechanisms for dealing with discotrees come into play only if at least some of the input rules contain context daughters.

4 Computing the Most Probable Parse

As in the CFG/PCFG case, we can extend a DPSG with a probability distribution on its rule set, yielding a *Probabilistic Discontinuous Phrase Structure Grammar* (PDPSG).

Definition 5 (Probabilistic Discontinuous Phrase Structure Grammar)
A Probabilistic Discontinuous Phrase Structure Grammar (PDPSG) *is a quintuple* $\langle V_N, V_T, S, R, P\rangle$, *where* $\langle V_N, V_T, S, R\rangle$ *is a DPSG and* P *is a function* $R \mapsto [0,1]$ *that assigns a probability to each rule such that* $\forall X \in V_N \colon \sum_{\Delta} P(X \rightarrow \Delta) = 1$. ■

In words, we assign a probability to each rule such that the probabilities of all rules with the same left-hand side category sum to 1. Note that Δ denotes right-hand sides of DPSG rules and as such contains information about which symbols are marked as context daughters. We define the probability of a string of terminal symbols as the sum of the probabilities of all parse trees that yield this string. The probability of a parse tree is the product of the probabilities of all rule instances used in constructing this tree.

We adapted the algorithm for computing the MPP given a PCFG (see e.g. (Brants, 1999) for a nice presentation) to perform the same task for PDPSGs. The algorithm maintains a set of accumulators $\delta_n(Y)$ for each symbol $Y \in V$ and each node n in the parse tree. Contrary to the context-free case, a node in a discontinuous tree cannot uniquely be described by the starting and ending position of its corresponding subparse. We instead use two bit-strings for this purpose, as explained in the previous section, which we shall below denote by b (= `covers`) and $\hat{b}$ (= `ctxt_covers`) for the sake of brevity. A node n dominates all terminal symbols for which the corresponding bit in b is set (equals 1); $\hat{b}$ indicates which input symbols are dominated by context daughters of n. That is to say, given a DPSG $G = \langle V_N = \{X^1, \ldots, X^N\}, V_T, X^1, R, P\rangle$ and an input string $w_{0,T} = w_0, \ldots, w_{T-1}$, we define the accumulators as variables $\delta_{b,\hat{b}}(Y)$, where $Y \in V$ and $b, \hat{b} \in \{0,1\}^T$, and compute their values bottom-up as follows.

Initialization:

$$(5) \quad \delta_{b,\hat{b}}(Z) = \begin{cases} 1 & \text{if } Z = w_{t-1} \\ 0 & \text{if } Z \neq w_{t-1} \end{cases} \qquad 1 \leq t \leq T,\ Z \in V_T,\ b = 0^{t-1}\,1\,0^{T-t},\ \hat{b} = 0^T$$

Recursion:

$$(6) \quad \delta_{b,\hat{b}}(X^i) = \max_{\substack{(X^i \rightarrow \Delta) \in R, \\ \Delta = Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k}, \\ \mathrm{AdjSeq}\left(Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k}\right), \\ b = \dot{\bigvee}_{1 \leq j \leq k,\, c_j = 0} b_j,\ \hat{b} = \dot{\bigvee}_{1 \leq j \leq k,\, c_j = 1} b_j}} P(X^i \rightarrow \Delta) \prod_{\substack{1 \leq j \leq k \\ c_j = 0}} \delta_{b_j,\hat{b}_j}(Y^j) \qquad 1 \leq i \leq N,\ b, \hat{b} \in \{0,1\}^T$$

Termination:

$$(7) \quad P_{\mathrm{MPP}}(w_{0,T} \mid G) = \delta_{b,\hat{b}}(X^1) \qquad b = 1^T,\ \hat{b} = 0^T$$

We initialize the algorithm by assigning a value of 1 to $\delta_{b,\hat{b}}(Z)$ for each terminal symbol $w_{t-1} = Z$ $(1 \leq t \leq T)$ in the input, such that b is a bit-string in which only the t-th bit is set and $\hat{b}$ is an all-zero bit-string. Next, we recursively compute the values of accumulators for larger and larger subparses. For each $\delta_{b,\hat{b}}(X^i)$, the algorithm explores all ways in which the subparse $X^i_{b,\hat{b}}$ can be constructed[7] from smaller parts and maximises over the respective probabilities. The basic idea behind the algorithm is thus the same as in the PCFG case. The PDPSG version of the algorithm differs from the PCFG one only with respect to how partial parses are combined to yield larger constituents. Each alternative for $X^i_{b,\hat{b}}$ is based on a DPSG rule $X^i \rightarrow Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k}$, where $Y^j_{b_j,\hat{b}_j}$ denotes the subparse corresponding to the j-th right-hand side symbol of the rule and $c_j \in \{0,1\}$ indicates whether this symbol is marked as a context daughter. A (partial) parse $X^i_{b,\hat{b}}$ can be constructed from the smaller parts corresponding to the $Y^j_{b_j,\hat{b}_j}$'s if the latter form an adjacency sequence, as indicated by the term $\text{AdjSeq}\left(Y^{1,c_1}_{b_1,\hat{b}_1} \ldots Y^{k,c_k}_{b_k,\hat{b}_k}\right)$ in the recursion formula (6). Given the two bit-strings of each subconstituent, we can perform the checks described in the previous section to ensure that the subconstituents form an adjacency sequence.

The probability of each alternative is computed as the product of the probability of the rule used to construct it and the accumulators for all right-hand side symbols corresponding to its direct daughters, and $\delta_{b,\hat{b}}(X^i)$ is set to the maximum of all alternatives' probabilities. After the values for all $\delta_{b,\hat{b}}(X^i)$ have been computed, the probability of the MPP is that of the accumulator for parse trees headed by the start symbol of the input grammar (X^1) that dominate all terminal symbols in the input ($b = 1^T$, $\hat{b} = 0^T$).

The computation of these accumulators can easily be incorporated within a slightly modified version of our DPSG parser. To this end, we associate each edge with an additional field prob that stores the accumulator value for the subparse the edge corresponds to. Edges for the terminal symbols in the input added to the agenda during initialization are assigned a prob value of 1, mirroring Equation (5). A new active edge that results from a DPSG rule matching an inactive edge that has been added to the chart inherits its probability from the underlying rule. When an active edge *ae* and an inactive edge *ie* are combined, the probability of the new edge is computed as the product of *ae*'s and *ie*'s probabilities, if *ie* is a direct daughter in the new edge, and is set to *ae*'s probability otherwise (cf. Equation (6)). Furthermore, we do not store more than one subparse with each edge, but only the most probable one found so far. In other words, if we encounter a new edge that would be merged with an already existing one, we instead discard the edge that has a lower probability value. This implies that the discarded edge must not already be contained in a higher subparse because, if this were the case, the probability of the higher subparse would have been computed incorrectly. To prevent this, we organise the agenda in such a way that parsing proceeds strictly bottom-up. Finally, when the agenda is empty, the edge that is headed by the start symbol of the grammar and that spans the entire input represents the Most Probable Parse for the given input sentence. The probability of the MPP is contained in the prob field of this edge (cf. Equation (7)).

Again, this algorithm can be used to find the MPP for either a PCFG or a PDPSG, depending on whether the input grammar contains rules with context daughters.

[7]We use $X^i_{b,\hat{b}}$ to denote a subparse headed by the nonterminal symbol X^i whose direct daughters dominate the input symbols for which the corresponding bits in b are set and whose context daughters dominate the symbols for which the bits in $\hat{b}$ are set.

5 Experiments

All experiments we conducted were based on grammars extracted from the NEGRA corpus (Skut et al., 1997), which consists of German newspaper text. The version we used contains 20 571 sentences. All sentences are part-of-speech tagged, and their syntactic structures are represented as discontinuous trees. In a preprocessing step, we removed sentences without syntactic structure, attached punctuation marks to suitable nodes in the discotree, and removed unbound tokens. The corpus of discotrees thus obtained consists of 19 445 sentences with an average length of 17 tokens. In order to have some sort of baseline against which we could compare our results from PDPSG parsing, we additionally transformed the discotrees in this corpus to context-free trees by re-attaching all continuous parts of discontinuous constituents to higher nodes. We kept 1 005 sentences from each corpus to be used in case of unforeseen events and split the remaining corpus into a training set of 16 596 sentences (90%) and a test set of 1 844 sentences (10%), such that both test sets (and both training sets) contained the same sentences, albeit with different structures. We extracted rule instances from both training sets[8] and computed each rule's probability as the ratio of its frequency to the frequency of all rules with the same left-hand side category, thus obtaining a PCFG and a PDPSG[9].

Due to limited computational resources, we restricted the two test sets to sentences with a maximum length of 15 tokens. We ran our parser on the part-of-speech tag sequences of these 959 sentences, once with the PCFG as input, and once using the PDPSG. The parse trees from the PCFG (PDPSG) experiment were then compared against the correct context-free trees (discotrees) in the test set. We determined average CPU time[10] per sentence and various accuracy measures for both experiments, which are summarised in Figure 3.

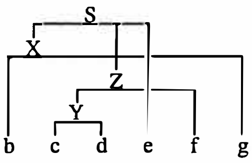
	PCFG	PDPSG
Precision	79.24%	77.75%
Recall	78.09%	76.81%
F-score	78.66%	77.28%
Labeled precision	75.12%	73.61%
Labeled recall	74.03%	72.72%

	PCFG	PDPSG
Labeled F-score	74.57%	73.16%
Coverage	96.35%	96.04%
Exact matches	39.83%	39.00%
Structural matches	43.07%	42.23%
CPU time per sent.	0.53 secs	24.78 secs

Figure 3: Accuracy results and average CPU time per sentence for both experiments

[8]For context-free trees, we extract rules in the usual way. For each nonterminal n in the tree, a rule is generated in which n's label constitutes the left-hand side and the labels of n's direct daughters form the right-hand side. Given a discotree, we also need to determine context daughters of discontinuous constituents, such that right-hand sides form adjacency sequences. This is done as follows. For each terminal t between two direct daughters d_1 and d_2 of a nonterminal n such that n does not dominate t, we determine the highest node dominating t that does not dominate n and whose yield is contained within the bounds of n. These nodes are the context daughters of n between d_1 and d_2 (with duplicates removed). For instance, we extract the rules in (i) from the discotree given in (ii).

(i) S → X Z e
X → b [Z] [e] g
Z → Y [e] f
Y → c d

(ii)


[9]The PCFG training corpus gave rise to 120 831 rule instances and 18 709 rules, of which 13 419 appear only once in the corpus. The PDPSG consists of 21 965 rules (generated from 123 528 rule instances); 16 317 of these rules appear only once in the training corpus.

[10]On a Sun Ultra Sparc 300 MHz with 1 GB main memory running Solaris 2.6.

Precision is defined as the percentage of constituents proposed by our parser which are actually correct according to the tree in the corpus. "Correct" means that the two constituents dominate the same terminals; for the different "labeled" measures, the node labels must match in addition. Recall is the percentage of constituents in the test set trees which are found by our parser. We define F-score as the harmonic mean of recall R and precision P, that is, as $F = \frac{2PR}{P+R}$. Coverage denotes the percentage of sentences for which a complete parse has been found by our parser. A proposed parse tree with an F-score of 100% is a structural match; if the *labeled* F-score is 100%, the tree is called an exact match.

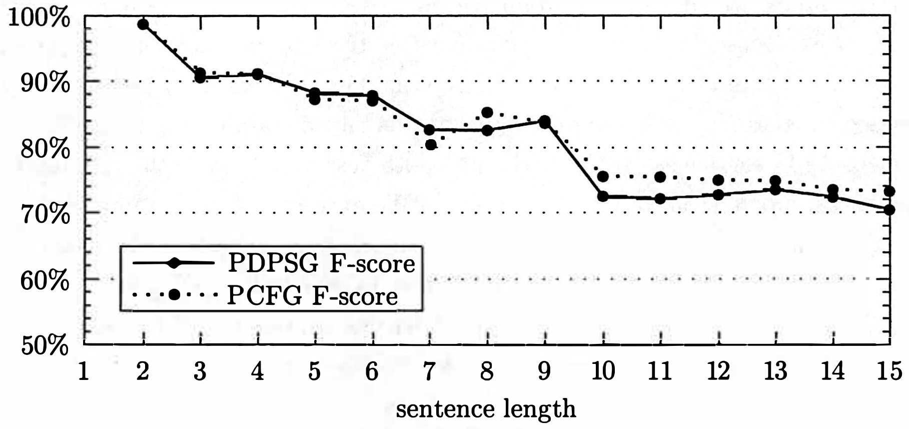


Figure 4: F-scores from both experiments plotted against sentence length

In comparing the results from the PDPSG experiment against the PCFG results, it is important to keep in mind that parsing with discotrees is a much harder task than parsing with context-free trees. The complexity of the latter is cubic in the sentence length, whereas our parsing algorithm for PDPSG takes, in the worst case, exponential time (see also (Reape, 1991)). Therefore, unsurprisingly, the average CPU time per sentence in the PDPSG experiment is almost 50 times larger than in the PCFG case. To make things worse, the difference in running time would be even larger for longer sentences.

The good news, on the other hand, is that accuracy drops only slightly (see Figure 4) when we accommodate within our parser the possibility of constituents to be discontinuous.

6 Conclusion and Future Work

We have presented a probabilistic extension of Discontinuous Phrase Structure Grammar, a formalism suitable for describing restricted discontinuities in a perspicuous single-level representation. Furthermore, we have developed a parser that can be used with probabilistic and non-probabilistic versions of both Context-Free Grammar and Discontinuous Phrase Structure Grammar, constructing context-free or discontinuous trees, respectively.

Although the probabilistic method applied is rather simplistic, the accuracy results obtained in the PDPSG experiment are comparable to the PCFG results. A severe drawback of our approach is its worst-case exponential running time. Future work will thus be primarily aimed at reducing this complexity (and additionally at increasing accuracy). There are at least two avenues towards this goal.

Firstly, we could try to restrict the formalism of DPSG further with respect to the kinds of discontinuities it is capable of representing. Vogel and Erjavec (1994) presented a restricted version of DPSG, called DPSG^R, and claimed that this formalism is properly contained in the class of mildly context-sensitive languages and that consequently a polynomial time recognition procedure exists for it. Note, however, that DPSG^R does not allow for the description of cross-serial dependencies; it is therefore too restricted to represent the discontinuity phenomena occuring in the NEGRA corpus[11]. A similar account, although within a different grammatical framework, was provided by Müller (1999) who presented an HPSG system in which linguistically motivated constraints on (dis)continuity were imposed, which led to a notable decrease in running time.

An alternative, orthogonal approach towards faster mechanisms for probabilistic parsing with discontinuous constituents is to stick to DPSG and try to find suitable approximation algorithms that reduce the *observed* running time. Several such approaches exist in the literature on probabilistic parsing, mostly based on (P)CFGs. Since DPSG is a straightforward extension of CFG, we expect that at least some of these can be extended with reasonable effort to also deal with discontinuous trees. In fact, the work presented in this paper serves as a first indication confirming this intuition. A more sophisticated approach could, for instance, be based on the statistical parser devised by Ratnaparkhi (1997). He utilized a set of procedures implementing certain actions to incrementally construct (context-free) parse trees. The probabilities of these actions were computed by Maximum Entropy models based on certain syntactic characteristics (features) of the current context, and effectively rank different parse trees. A beam search heuristic was used that attempts to find the highest scoring parse tree. In order to extend this approach to DPSG, we would need a different set of procedures capable of constructing discontinuous trees and a suitable set of features.

Edge-based best-first chart parsing (Charniak, Goldwater, & Johnson, 1998; Charniak & Caraballo, 1998) is another very promising approach. Charniak et al. (1998) proposed to judge edges according to some probabilistic figure of merit that is meant to approximate the likelihood that an edge will ultimately appear in a correct parse. Edges are processed in decreasing order of this value until a complete parse has been found (or perhaps several ones), leaving edges on the agenda. They reported results equivalent to the best prior ones using only one twentieth the number of edges.

The edge-based best-first parsing approach could easily be applied to DPSG and our chart parsing algorithm as well. To this end, we would need to organise the agenda as a priority queue, so that edges are processed in decreasing order of their respective figure of merit values. We are confident that suitable figures of merits can be found for DPSG chart parsing that will lead to a significant decrease in running time.

References

Blevins, J. P. (1990). *Syntactic complexity: Evidence for discontinuity and multidomination.* PhD thesis, University of Massachusetts, Amherst, MA.

Brants, T. (1999). *Tagging and parsing with cascaded Markov models — automation of corpus annotation.* PhD thesis, University of the Saarland, Saarbrücken, Germany.

[11]Bresnan, Kaplan, Peters, and Zaenen (1982) provided additional evidence that formalisms capable of representing cross-serial dependencies are desirable.

Bresnan, J., Kaplan, R. M., Peters, S., & Zaenen, A. (1982). Cross-serial dependencies in dutch. *Linguistic Inquiry, 13*(4), 613–635.

Bunt, H. (1991). Parsing with discontinuous phrase structure grammar. In M. Tomita (Ed.), *Current issues in parsing technology* (pp. 49–63). Dordrecht, Boston, London: Kluwer Academic Publishers.

Bunt, H. (1996). Formal tools for describing and processing discontinuous constituency structure. In H. Bunt & A. van Horck (Eds.), *Discontinuous constituency* (pp. 63–83). Berlin, New York: Mouton de Gruyter.

Charniak, E., & Caraballo, S. (1998). New figures of merit for best-first probabilistic chart parsing. *Computational Linguistics, 24*(2), 275–298.

Charniak, E., Goldwater, S., & Johnson, M. (1998). Edge-based best-first chart parsing. In *Proceedings of the Sixth Workshop on Very Large Corpora.* Montreal, Canada.

Johnson, M. (1985). Parsing with discontinuous constituents. In *Proceedings of the 23rd ACL meeting* (pp. 127–132). Chicago: Association for Computational Linguistics.

McCawley, J. D. (1982). Parentheticals and discontinuous constituent structure. *Linguistic Inquiry, 13*(1), 91–106.

Müller, S. (1999). Restricting discontinuity. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium 1999 (NLPRS '99).* Peking.

Plaehn, O. (1999). *Probabilistic parsing with discontinuous phrase structure grammar.* Diplom thesis, University of the Saarland, Saarbrücken, Germany. (`http://www.coli.uni-sb.de/~plaehn/papers/dt.html`)

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP-97.* Providence, RI.

Reape, M. (1991). Parsing bounded discontinuous constituents: Generalisations of some common algorithms. In T. van der Wouden & W. Sijtsma (Eds.), *Computational Linguistics in the Netherlands. Papers from the First CLIN-meeting.* Utrecht: Utrecht University-OTS.

Skut, W., Krenn, B., Brants, T., & Uszkoreit, H. (1997). An annotation scheme for free word order languages. In *Proceedings of the Fifth Conference on Applied Natural Language Processing ANLP-97.* Washington, DC.

van der Sloot, K. (1990). *The TENDUM 2.7 parsing algorithm for DPSG* (ITK Research Memo). Tilburg: ITK.

Vogel, C., & Erjavec, T. (1994). Restricted discontinuous phrase structure grammar and its ramifications. In C. Martin-Vide (Ed.), *Current issues in mathematical linguistics* (pp. 131–140). Amsterdam: Elsevier.

AN EFFICIENT LR PARSER GENERATOR FOR TREE ADJOINING GRAMMARS

Carlos A. Prolo*
Dept. of Computer and Information Science
University of Pennsylvania
Philadelphia, PA, 19104, U.S.A.
prolo@linc.cis.upenn.edu

Abstract

The first published LR algorithm for Tree Adjoining Grammars (TAGs [Joshi and Schabes, 1996]) was due to Schabes and Vijay-Shanker [1990]. Nederhof [1998] showed that it was incorrect (after [Kinyon, 1997]), and proposed a new one. Experimenting with his new algorithm over the XTAG English Grammar[XTAG Research Group, 1998] he concluded that LR parsing was inadequate for use with reasonably sized grammars because the size of the generated table was unmanageable. Also the degree of conflicts is too high.

In this paper we discuss issues involved with LR parsing for TAGs and propose a new version of the algorithm that, by maintaining the degree of prediction while deferring the "subtree reduction", dramatically reduces both the average number of conflicts per state and the size of the parser.

1 Introduction

For Context Free Grammars, LR parsing [Knuth, 1965, Aho et al., 1986] can be viewed as follows. If at a certain state q_0 of the LR automaton, during the parsing of a sentence, we expect to see the expansion of a certain non-terminal A, and there is a production $A \to X_1\ X_2\ X_3\ ...\ X_n$ in the grammar, then the automaton must have a path labeled $X_1\ X_2\ X_3\ ...\ X_n$ starting at q_0. This is usually represented by saying that each state in the path contains a "dotted item" for the production, starting with $A \to \bullet\ X_1\ X_2\ X_3\ ...\ X_n$ at q_0, with the dot moving one symbol ahead at each state in the path. We will refer to the last state of such paths as *final*. The dot being in front of a symbol X_i represents the fact that we expect to see the expansion of X_i in the string. If X_i is a non-terminal, then, before crossing to the next state, we first have to check that some expansion of X_i is actually next in the input.

The situation is depicted in Figure 1, where paths are represented as winding lines and single arcs as straight lines. At a certain state q_1, where some possible yield of the prefix α_1 of a production $A \to \alpha_1 B \alpha_2$ has just been scanned, the corresponding dotted item is at a non-terminal B. This state turns out to be itself the beginning of other paths, like β in the picture, that lead to the recognition of some match for B through some of its rules. The machine, guided by the input string, could traverse this sub-path until getting to q_4. At this final state, some sort of memory is needed to get back to the previous path for A, to then cross from q_1 to q_2 (i.e. B has just been seen). In LR parsing this

*I am grateful to Aravind Joshi, Joseph Rosenzweig, Anoop Sarkar, Mahesh Viswanathan, Fei Xia, and the very kind IWPT 2000 reviewers, for their contribution to the development of this paper.

is realized by a stack that stores the sequence of states traversed during recognition. At each final state, the recognized production is unwound from the stack, in an operation called a reduction, leaving exposed the state q_1 and making a transition over the non-terminal B to q_2 (a *goto* transition).

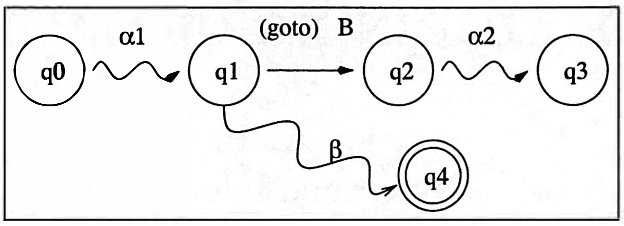
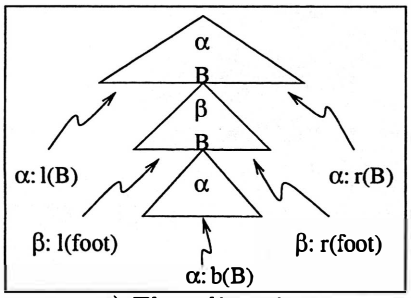


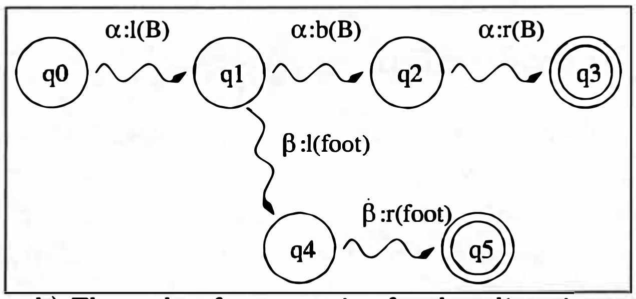
Figure 1: LR parsing for Context Free Grammars

A reduction means a commitment to a certain substructure in the attempt to find a parse for the sentence. Whenever the parser reaches a state that has a reduction (a final state), and there are also additional *reduce* operations or a *shift*, it has decide whether to do that reduction or try another alternative. That is certainly a limitation of the method, but at least, the decision has to be taken only after witnessing that the input has a complete yield for the production. That is exactly what will be hard to guarantee in the TAG case.

The problem with LR parsing for TAGs is adjunction, as illustrated in Figure 2. Figure 2.a sketches the adjunction of a tree β at a node labeled B of a tree α. $\alpha : l(B)$ is the part of α that appears before (left of) node B. Similarly, $\alpha : b(B)$ and $\alpha : r(B)$ are the parts below and to the right of B respectively. $\beta : l(foot)$ and $\beta : r(foot)$ are the two halves of β split by its spine.



a) The adjunction



b) The paths of computation for the adjunction

Figure 2: The problem of adjunction

There are two natural paths the LR parsing nature would induce the machine to have (Figure 2.b). One traverses α: state q_1 is the natural candidate to predict what comes below B (even after having recognized the left side of the adjoined tree). And State q_2 in the same way is the natural state to start the prediction of what comes after B. As for the other path, state q_1 predicts the adjunction of β, more specifically its left side. And the right side (i.e., past the foot node) is naturally predicted at q_4. Now let's look at the dynamic path of computation, corresponding to the projection of each segment in the sentence. This is:

$$q_0 \Rightarrow \alpha : l(B) \Rightarrow q_1 \Rightarrow \beta : l(foot) \Rightarrow q_4 \Rightarrow (q_1) \Rightarrow \alpha : b(B) \Rightarrow q_2 \Rightarrow (q_4) \Rightarrow \beta : r(foot) \Rightarrow$$
$$\Rightarrow q_5 \Rightarrow (q_2) \Rightarrow \alpha : r(B) \Rightarrow q_3$$

Although we marked only q_3 and q_5 as final, as we would like it to be, since they mark the end of the recognition of each tree, there are two other points of disruption at states q_4 and q_2. In a normal LR model of computation these points would correspond to reduction-like operations with *goto*'s to jump to the resuming state. Two problems then immediately arise: one that we have called "early reduction"; the other is the unsuitability of the normal stack model.

Figure 3 shows an intuitive but rather naive first attempt to solve the problem. At the State q_4 a reduction would be triggered sending the machine via a "*goto-below*" to state q_6. But notice that this operation would pop out all the states after q_1, including q_4. Later, at q_2, when a new reduction would be required, we do not know anymore how to get back to the state that contains the "*goto-foot*", the popped out q_4, unless we change the underlying storing model to something different from a stack. We are not finished. Even if we could get to q_8, we have to keep q_2 alive somehow, because when reaching q_5 we need it to access the "*goto-right*" to state q_7. It is important to note that we do not know in advance (i.e., at compilation time) which states those are. They are dependent on the actual computation. Although we could look for another storing model, the graver problem is the early reduction: the need to commit to tree β, before seeing its right side in the input.

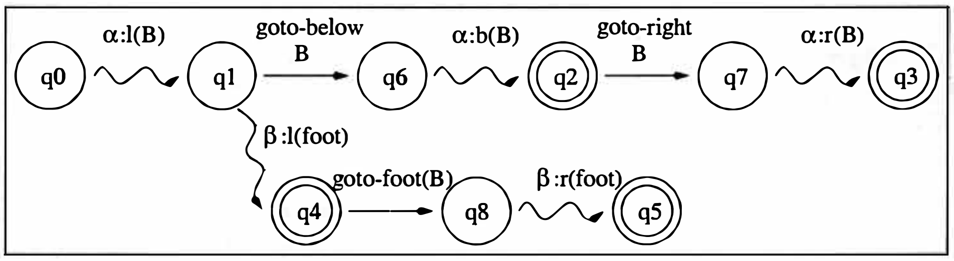


Figure 3: A naive solution for adjunction

Our second model, used by Nederhof[1998], shown in Figure 4, is clearly an enhancement of the naive one, as it increases the prediction capacity. The prediction of $\alpha : b(B)$ is made at q_4, instead of q_1. Hence we lose the need for the first reduction mentioned above. The reduction in q_2 does not cause any trouble since the state from where we want to take the *goto*, q_4, will be in the stack at that moment and is easy to recover. At q_5 the reduction takes its *goto-right* to q_7 from q_1, which is recoverable from the stack, instead of q_2 as in the previous approach, which has been popped out. Alas, things are not that simple.

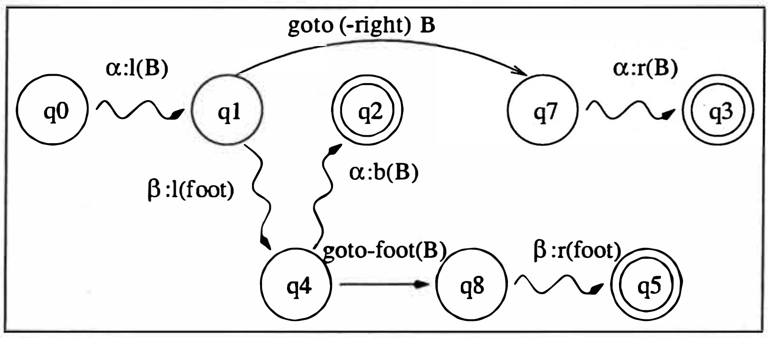


Figure 4: Nederhof's solution for adjunction

The problem of "early-reduction" is still significant, although not as much as in the first model. When the parser is in a state where (as one of the alternatives) it has just finished recognizing the subtree below a node where something has adjoined, as in state q_2 in Figure 4, somehow it has to get back to state q_4, and with a *goto* (*goto-foot*) to resume at state q_8 the recognition of the adjoined tree. In Nederhof's approach this is done by performing a reduction, that is, choosing one of the items in q_2 that signifies that a subtree below an adjunction node has been recognized (e.g., the subtree corresponding to $\alpha : b(B)$), and promoting a reduction of that subtree. This actually means committing to tree α too early, before seeing whether the rest of the input matches $\alpha : r(B)$. In the example in Figure 5, immediately after seeing a prefix "N", as in (1), that could be anchoring any of the innumerably many α_i trees generating continuations such as in (2), the parser would have to commit in advance to one of them, due to the need to turn back to the right side of the relative clause tree β.[1] The most visible consequence of this problem in the table is the huge rate of conflicts involving this kind of reduction that we drastically reduce with our approach as shown in Section 4.

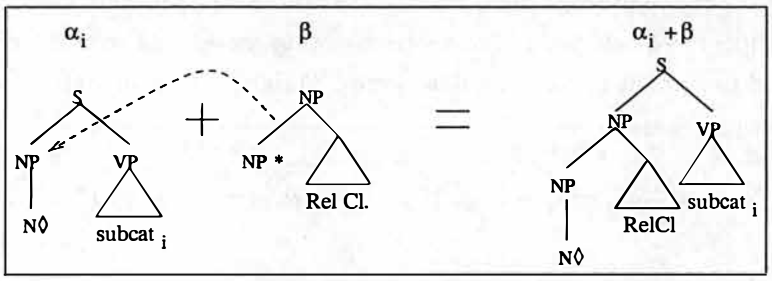


Figure 5: Example of early reduction

(1) (the) cat/N ...
(2) ... [that John saw] died.
... [that John saw] ate an apple.
... [that John saw] asked for MEOW MIX.

The second problem is an immediate consequence of the first. *goto*'s are traditionally made on a symbol basis. In Nederhof's approach, however, because we have already committed to a (certain node of a) tree at state q_2, a strategy of having one *goto* per node is adopted. Besides increasing the number of states, it simultaneously causes an explosion in the number of *goto* transitions per state, making the size of the table unmanageable. What we had in mind when looking for a new algorithm was precisely to solve those two problems.

There is still a third problem, related to the lack of the valid prefix property. Predicting the path $\alpha : b(B)$ below a node where adjunction is supposed to take place after the left part of the adjoined tree is hard. Actually, we can show it to be impossible with Nederhof's approach as well as our own (although it would not be a problem for the "naive" solution).[2] An attempt to do that in the algorithm for table generation would lead to non-termination for some grammars. Hence, at state q_4 all items corresponding to adjunction nodes labeled B in some tree are inserted, with the dot past the node, even if many of those nodes were not possible at that state, for a particular input prefix. Although in

[1]The example given is for explanatory purposes, and one could claim that the subjects of the α_i trees should be substitution nodes, for instance, which is true. The problem of early reduction described, however, largely appears in more subtle contexts.

[2]Recall the natural candidate for such prediction would be q_1.

our approach the effects of misprediction and overgeneration are less harmful, it decisively affects the design of the algorithm as we see below.

Figure 6 sketches our proposed solution. The *goto* now depends on both q_1 and q_2. At state q_2, the items that correspond to the end of the recognition of bottom subtrees labeled B, like α:*b*(B), are grouped into subsets of the same size, size being the number of leaves of the subtree. Hence *goto-adj*(q_1, q_2, B, l) has the set of possibilities of continuation like α:*r*(B) that were predicted at q_1 and confirmed at q_2 that have l leaves under B. The dependency on q_1 has the sole purpose of fixing the overprediction we mentioned above, originating at q_4, and propagated to q_2. When q_2 is reached, an action *bpack(B,l)* extracts from the stack the l nodes under B (α:*b*(B), uncovering q_4 and the *goto-foot*), and puts them back in the stack as a single embedded stack element. The material is unpacked during β reduction before moving to q_7.[3] The details of the algorithm are in the next section.

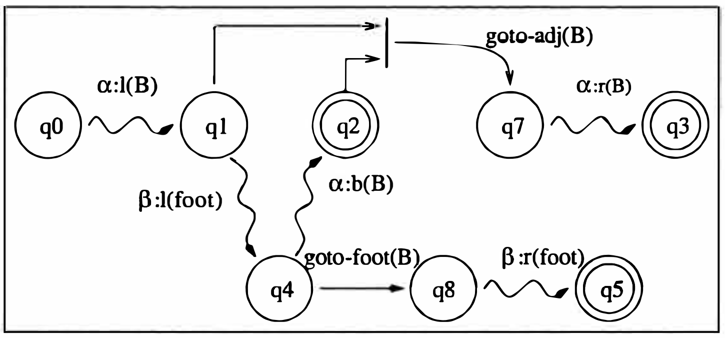


Figure 6: Proposed solution for adjunction

2 The Algorithm

2.1 The Table Generation for the NA/OA Case

We describe, in this subsection, an algorithm for a TAG grammar in which all nodes are marked either NA(for Null Adjunction) or OA(Obligatory Adjunction). Later we deal with the general case. We do not consider selective adjunction (SA). To refer to the symbol (label) of a node n, we use *symb*(n). ϵ is used to label anchors to represent the absence of a terminal symbol (the "empty" label).

A *dotted node* is a pair (n, *pos*) where n is a tree node and *pos*, as in [Schabes, 1990], can be *la* (at the left and above the node), *lb* (left and below), *rb* (right and below), or *ra* (right and above). We also represent the dotted nodes pictorially as ${}^{\bullet}n$, ${}_{\bullet}n$, $n_{\bullet}$, and $n^{\bullet}$. A *dotted item* (*item*, for short) is a pair (t, *dn*), where t is an elementary tree, and *dn* a dotted node whose node component belongs to t.

We define a congruence relation $\cong$ as the least symmetric and transitive relation such that, for any pair of items $i_1 = (t, dn_1)$ and $i_2 = (t, dn_2)$ of the same tree t, $i_1 \cong i_2$ if any of the following conditions apply:

[1.] $dn_1 = {}^{\bullet}n$, $dn_2 = {}_{\bullet}n$, and n is marked for null adjunction (NA).

[2.] $dn_1 = {}_{\bullet}n$, $dn_2 = n_{\bullet}$, and n is an anchor labeled ϵ.

[3.] $dn_1 = {}_{\bullet}n$, $dn_2 = {}^{\bullet}m$, and m is the leftmost child of n.

[4.] $dn_1 = n_{\bullet}$, $dn_2 = n^{\bullet}$, and n is marked for null adjunction (NA).

[3]This is quite similar to how stacks of stacks are used in the theories of automata for TAGs. Also the model resembles the one used in [Schabes and Vijay-Shanker, 1990] in their attempt.

[5.] $dn_1 = n_\bullet$, $dn_2 = m^\bullet$, and m is the rightmost child of n,

[6.] $dn_1 = n^\bullet$, $dn_2 = {}^\bullet m$, and m is the right sibling of n.

Congruent items are indistinguishable for the purpose of our algorithm and its underlying theory. Hence instead of dealing with separate items, we use equivalence classes under $\cong$. If i is a term, $[i]$ is the congruence class to which i belongs. Each congruence class has one and only one *active* item, where an active item has one of the following forms:

- $(t, {}^\bullet n)$, where n is marked OA: this item triggers adjunction on n.
- $(t, {}_\bullet n)$, where n can be: a substitution node (triggers substitution), a non-ϵ anchor (triggers a *shift* action), or a foot node (triggers the return to the tree where adjunction occurred).
- $(t, n_\bullet)$, where n is marked OA: this item triggers a *bpack* action, that we will define later.
- $(t, n^\bullet)$, where n is the root of t: triggers a *reduce* (alpha or beta) operation.

Given a set S of dotted items (under $\cong$) of a TAG grammar, we define $closure(S)$ as the minimal set that satisfies the following conditions:

1. if $[i] \in S$, then $[i] \in closure(S)$.

2. if $[(t, {}_\bullet n)] \in closure(S)$, and n is a substitution node, then, for every initial tree t_1 with root node r labeled $symb(n)$, $[(t_1, {}^\bullet r)] \in closure(S)$.

3. if $[(t, {}^\bullet n)] \in closure(S)$ and n is a node marked OA, then, for every auxiliary tree t_1 with root node r labeled $symb(n)$, $[(t_1, {}^\bullet r)] \in closure(S)$.

4. if $[(t, {}_\bullet n)] \in closure(S)$, where n is a foot node, then, for every OA node m of the grammar such that $symb(m) = symb(n)$, $[(t_1, {}_\bullet m)] \in closure(S)$, where t_1 is the elementary tree that contains m. We point out that it is at this closure operation that the loss of the valid prefix property occurs.

To define a parsing table for a grammar G with goal symbol S, we first extend G by adding one new tree called *start*, with two nodes: the root, labeled with a fresh symbol ST, marked NA; and ST's single child, a substitution node labeled S. Then, let I be the set of all equivalence classes of items of G under $\cong$. Let N be the set of symbols, $T \subseteq N$ be the set of symbols that appear in some anchor, and $\mathbb{N}$ the set of non-negative integers. We define the "parsing table" as a set $Q \subseteq 2^I$ of states with initial state $q_0 = closure(\{[(start, {}^\bullet ST)]\}) \in Q$, together with the functions $GOTO_{subst} : Q \times N \to Q$, $GOTO_{foot} : Q \times N \to Q$, $GOTO_{adj} : Q \times Q \times N \times \mathbb{N} \to Q$, and $ACTIONS : Q \times T \to 2^A$, where $A = \{\text{shift } q \mid q \in Q\} \cup \{\alpha\text{-reduce } t \mid t \text{ is an alpha tree }\} \cup \{\beta\text{-reduce } t \mid t \text{ is a beta tree }\} \cup \{\text{bpack } (A, l) \mid A \in N, l \in \mathbb{N}\} \cup \{\text{accept}\}$. Q, $GOTO_{subst}$, $GOTO_{foot}$, $GOTO_{adj}$, and $ACTIONS$ are the minimal set and functions that satisfies the following inductive definition.

1. $q_0 \in Q$.

2. If $q \in Q$ and $p = \{[(t, n_\bullet)] \mid [(t, {}_\bullet n)] \in q$ and n is an anchor $\}$, then $p' = closure(p) \in Q$, and (shift p') $\in ACTIONS(q, symb(n))$.

3. If $q \in Q$ and $[(t, n^\bullet)] \in Q$, where n is the root of t, then, for every $a \in T$: if t is an initial tree, then (α-reduce t) $\in ACTIONS(q, a)$, else (β-reduce t) $\in ACTIONS(q, a)$.

4. If $q \in Q$ and $[(t, n_\bullet)] \in Q$, where n is marked OA, then, for every $a \in T$, (bpack($symb(n), l$)) $\in ACTIONS(q, a)$, where l is the number of non-ϵ leaves under n.

5. If $q \in Q$ and $[(start, ST^{\bullet})] \in Q$, then accept $\in ACTIONS(q, \$)$.

6. If $q \in Q$ and $p = \{[(t, n_{\bullet})] \mid [(t, {}_{\bullet}n)] \in q$ and n is a substitution node, then $p' = closure(p) \in Q$, and $GOTO_{subst}(q, symb(n)) = p'$.

7. If $q \in Q$ and $p = \{[(t, n_{\bullet})] \mid [(t, {}_{\bullet}n)] \in q$ and n is a foot node, then $p' = closure(p) \in Q$, and $GOTO_{foot}(q, symb(n)) = p'$.

8. If $q_1, q_2 \in Q$, and for some $k \in \mathbb{N}$, $p = \{[(t, n^{\bullet})]) \mid [(t, {}^{\bullet}n)] \in q_1, [(t, n_{\bullet})] \in q_2$, n is marked OA, and the number of leaves under n in t equals $k\}$, then $p' = closure(p) \in Q$, and $GOTO_{adj}(q_1, q_2, symb(n), k) = p'$.

The state corresponding to the empty set is called *error*. The domain of k, in practice, is bounded by the grammar: the maximum number of leaves of any node. As usual, if an entry contains more than one action, we say that there is a *conflict* in the entry. As expected, two *shift* actions are never in conflict. Although *reduce* and *bpack* actions do not actually depend on a terminal symbol for the current algorithm, in practice the table could be so constrained, either by having the algorithm extended to an LR(1) version or by using empirical evidence to reduce/resolve conflicts. In our inductive definition, each state was associated with the closure of a (usually much smaller) set of items. This set prior to the closure is generally known as its *kernel*. It is a property of our framework that an alternative definition whose states are associated with kernels would produce an automaton isomorphic to the one we defined above.

2.2 The Driver

The algorithm for the driver presented below uses a stack. Let *st*, st_1 be stacks and *el* be a stack element. **PUSH** (*el*,*st*), **TOP** (*st*) and **POP** (*st*) have the usual meanings. **POP** (*st*, *k*) pops out the top *k* elements from *st*. **EXTRACT** (*st*, *k*) pops out the top *k* elements returning another stack with the *k* elements in the same order they were in *st*. Its counterpart, **INSERT** (*st1*, *st*), pushes onto *st* all elements in *st1*, again preserving the order. null is the empty stack. size(*st*) is the number of elements in the stack *st*. The stack is a sort of higher order structure, in the sense that an element of the stack may contain an embedded stack. More precisely, an element of the stack is a pair (X, q), where q is a state of the parsing table, and X is either a grammar symbol or another stack.

The algorithm for the driver also uses *input*, the sequence of symbols to be recognized. Two operations are defined over it: **look**, which returns the leftmost symbol of *input* or \$ if *input* is the null sequence; and **advance**, which removes the leftmost symbol from *input*.

Let $ACTIONS$, $GOTO_{subst}$, $GOTO_{foot}$, and $GOTO_{adj}$ be the four tables/functions for a grammar G. Let q_0 be the initial state of the corresponding machine. Let *input* and the stacks *stack*, *emb-stack* be as defined above. The algorithm for the driver is then as follows.

```
stack = null
PUSH (<null,$q_0$ >, stack)
forever
    let <−, state> = TOP (stack)
    let lookahead = look (input)
    if ACTIONS (state, lookahead) = error then return failure
    else let action be in ACTIONS (state, lookahead)      (a non-deterministic choice)
        case action of:
            shift p:
```

```
    PUSH (<lookahead,p >, stack)
    advance (input)
  $\alpha$-reduce t:
    let $k$ be the number of non-empty leaves of $t$
    let $A$ be the symbol at the root of $t$
    POP (stack, k)
    let <–, state1> = TOP (stack)
    PUSH (<A,$GOTO_{subst}$(state1,A)>, stack)
  $\beta$-reduce t:
    let $kl$ and $kr$ be the number of non-empty leaves of $t$
          respectively to the left and to the right of the foot of $t$
    POP (stack, kr)
    let <aux-stack,-> = TOP (stack)
    POP (stack)
    let $k$ = size (aux-stack)
    let <-,state2> = TOP (if $k > 0$ then aux-stack else stack)
    POP (stack, kl)
    let <-, state1> = TOP (stack)
    INSERT (aux-stack, stack)
    let $A$ be the symbol at the root of $t$
    let state = $GOTO_{adj}$ (state1,state2,A,k)
    if state = error then return failure
          (a consequence of the lack of the prefix property)
    let <el,- > = TOP (stack)
    POP (stack)
    PUSH (<el, state>, stack)
  bpack (A,k):
    emb-stack = EXTRACT (stack, k)
    let <-, state1>= TOP (stack)
    PUSH (<emb-stack, $GOTO_{foot}$ (state1,A)>, stack)
  accept:
    return success
```

2.3 The General Case

The obvious way to handle the general case is to transform the arbitrary input grammar into an equivalent one with all nodes marked NA/OA prior to the application of the algorithm of Subsection 2.1. For every TAG grammar G, there is a grammar G' equivalent to G with respect to the possible derivations, whose nodes are all OA/NA marked. For instance, given the grammar G, in Figure 7,[4] we can construct G' in Figure 8 by replacing each tree t of G by 2^n new trees, where n is the number of unmarked nodes of t. Each new tree corresponds to one of the possible assignments of marks NA/OA to each of the unmarked nodes. In fact, in a TAG derivation, a tree instance together with the Gorn addresses at which other trees have adjoined defines exactly one of these G' component trees.

The approach we implemented, however, takes advantage of the original compact representation, avoiding exploding the number of trees, and hence items and item sets. An OA/NA tree can be represented in items as the original unmarked one plus a list of the OA nodes, much like the Gorn addresses in derivations. It turns out that certain distinctions are irrelevant for the algorithm. For

[4]Substitution nodes are always assumed to be NA,even if not explicitly marked as such in the trees. Nodes are represented by their label, e.g. NP, plus an optional subscript for identification purposes, as in NP, NP_s, and NP_o. Subscripts are ignored by the algorithm.

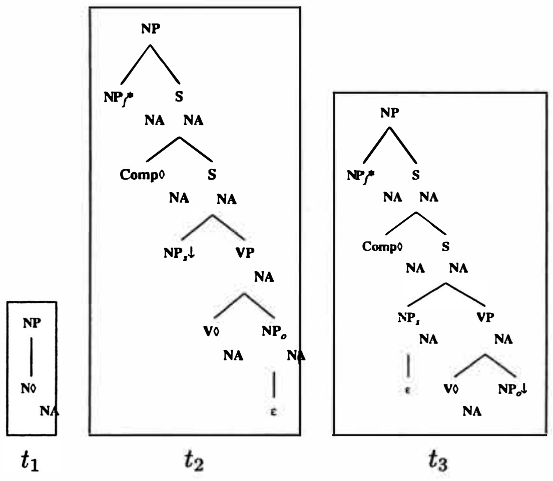


Figure 7: TAG grammar G for relative clauses

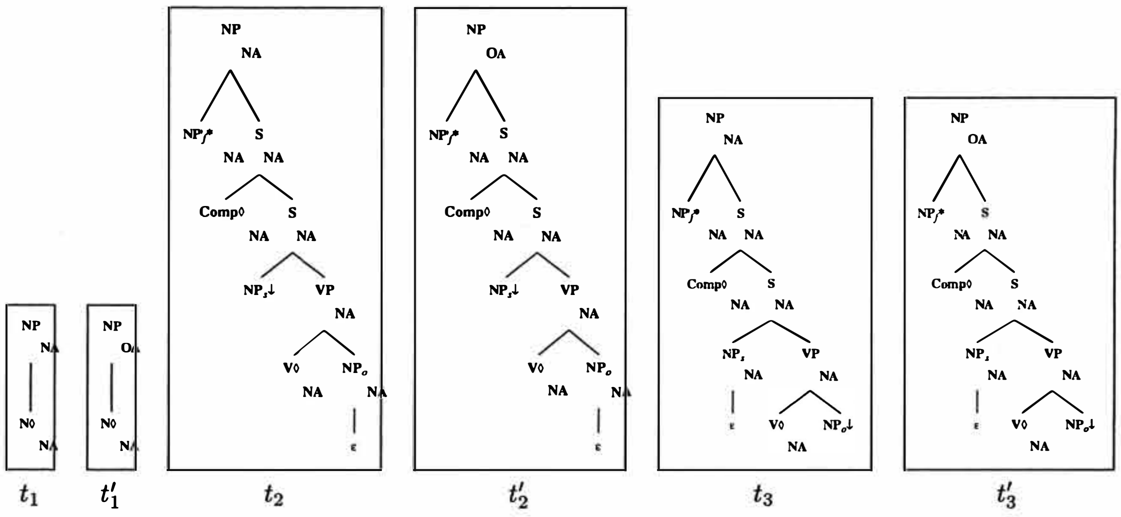


Figure 8: TAG grammar G', equivalent to G, with only OA/NA nodes

instance, it is not relevant to identify in the item whether tree nodes to the right of the dotted item allow for adjunction or not. In fact markings at nodes that come "after" the dotted node according to the usual dot traversal order defined in [Schabes, 1990] are not distinguishable. Our algorithm only keeps track of the innermost OA node that dominates the dotted node in an item, the only strong requirement, so that the *bpack* operation and the $GOTO_{adj}$ function can be obtained. We omit details of changes to be made in the algorithm of Subsection 2.1. Mostly they concern closure rule 4, and rules 4 and 8 in the definition of the automaton.

One last word: the approach we took is not totally equivalent to the bare OA/NA decomposition version. Keeping track of the history of adjunction for nodes that appear before the dotted node would generate tables with slightly more fine grain distinctions. Whether this could have some practical importance is still to be investigated.

3 An Example

The set of states and the $GOTO_{adj}$ function produced by the OA/NA algorithm for grammar G' of Figure 8 are shown below. Non-*shift* actions were defined together with the states. Functions $GOTO_{foot}$, $GOTO_{subst}$, and *shift* actions are better viewed as a graph, in Figure 9. Figure 10 shows the parsing sequence for the string "girls/N that/Comp John/N likes/V".

S_0 : { [(*start*, ${}^\bullet$NP)], [(t_1, ${}_\bullet$N)], [(t'_1, ${}^\bullet$NP)], [(t_2, ${}_\bullet$NP$_f$)], [(t'_2, ${}^\bullet$NP)], [(t_3, ${}_\bullet$NP$_f$)], [(t'_3, ${}^\bullet$NP)], [(t'_1, ${}_\bullet$N)], [(t'_2, ${}_\bullet$NP$_f$)], [(t'_3, ${}_\bullet$NP$_f$)] }

S_1 : { [(*start*, ST${}^\bullet$)] } *ACTIONS*(S_1,\$) = { accept }

S_2 : { [(t_1, NP${}^\bullet$)], [(t'_1, NP${}_\bullet$)] } *ACTIONS*(S_2,-) = { α-reduce t_1, bpack(NP,1) }

S_3 : { [(t_2, ${}_\bullet$Comp)], [(t_3, ${}_\bullet$Comp)], [(t'_2, ${}_\bullet$Comp)], [(t'_3, ${}_\bullet$Comp)] }

S_4 : { [(t_2, ${}_\bullet$NP$_s$)], [(t_3, ${}_\bullet$V)], [(t'_2, ${}_\bullet$NP$_s$)],[(t'_3, ${}_\bullet$V)], [(t_1, ${}_\bullet$N)], [(t'_1, ${}^\bullet$NP)], [(t_2, ${}_\bullet$NP$_f$)], [(t'_2, ${}^\bullet$NP)], [(t_3, ${}_\bullet$NP$_f$)], [(t'_3, ${}^\bullet$NP)], [(t'_1, ${}_\bullet$N)], [(t'_2, ${}_\bullet$NP$_f$)], [(t'_3, ${}_\bullet$NP$_f$)] }

S_5 : { [(t_2, ${}_\bullet$V)], [(t'_2, ${}_\bullet$V)] }

S_6 : { [(t_3, ${}_\bullet$NP$_o$)], [(t'_3, ${}_\bullet$NP$_o$)], [(t_1, ${}_\bullet$N)], [(t'_1, ${}^\bullet$NP)], [(t_2, ${}_\bullet$NP$_f$)], [(t'_2, ${}^\bullet$NP)], [(t_3, ${}_\bullet$NP$_f$)], [(t'_3, ${}^\bullet$NP)], [(t'_1, ${}_\bullet$N)], [(t'_2, ${}_\bullet$NP$_f$)], [(t'_3, ${}_\bullet$NP$_f$)] }

S_7 : { [(t_2, NP${}^\bullet$)], [(t'_2, NP${}_\bullet$)] } *ACTIONS*(S_7,-) = { β-reduce t_2, bpack(NP,4) }

S_8 : { [(t_3, NP${}^\bullet$)], [(t'_3, NP${}_\bullet$)] } *ACTIONS*(S_8,-) = { β-reduce t_3, bpack(NP,4) }

S_9 = $GOTO_{adj}$(S_0,S_2,NP,1) = $GOTO_{adj}$(S_4,S_2,NP,1): { [(t'_1, NP${}^\bullet$)] }
ACTIONS(S_9,-) = { β-reduce t'_1 }

S_{10}= $GOTO_{adj}$(S_0,S_7,NP,4) = $GOTO_{adj}$(S_4,S_7,NP,4): { [(t'_2, NP${}^\bullet$)] }
ACTIONS(S_{10},-) = { β-reduce t'_2 }

S_{11}= $GOTO_{adj}$(S_0,S_8,NP,4) = $GOTO_{adj}$(S_4,S_8,NP,4): { [(t'_3, NP${}^\bullet$)] }
ACTIONS(S_{11},-) = { β-reduce t'_3 }

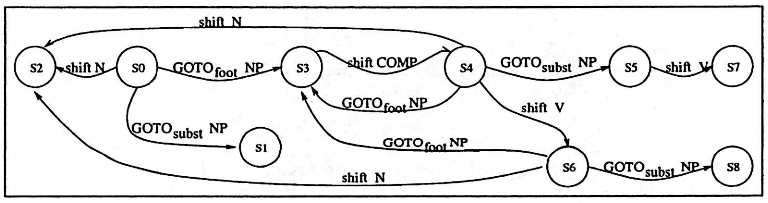


Figure 9: *shift* actions and functions $GOTO_{subst}$, $GOTO_{foot}$ for G'

4 Evaluation of the Algorithm

We show in Table 11 the results of the application of our algorithm as well as Nederhof's[5] to a recent version of the XTAG grammar with 1009 trees and 11490 nodes. *conflicts* is the average number of actions per pair (state,terminal). *reductions* and *bpacks* are respectively the average number of *reduce* and *bottom pack* actions in a state.[6] *transitions* is the total number of transition entries, including *shift*'s and *goto*'s. The number of conflicts is drastically reduced with our algorithm. In particular the nasty early reduction(*bpack*) conflicts are decreased, improving the general quality of the conflicts

[5]We reimplemented Nederhof's algorithm

[6]For Nederhof's approach *reduce* stands for "reduce tree" and *bpack* stands for "reduce subtree".

stack	input	sel. action
[(-,S_0)]	girls/N ...	shift S_2
[(-,S_0) (girls/N, S_2)]	that/Comp ...	bpack (NP,1)
[(-,S_0) ([(girls/N, S_2)], S_3)]	that/Comp ...	shift S_4
[(-,S_0) ([(girls/N, S_2)], S_3) (that/Comp, S_4)]	John/N ...	shift S_2
[(-,S_0) ([(girls/N, S_2)], S_3) (that/Comp, S_4) (John/N, S_2)]	likes/V $	α-reduce t_1
[(-,S_0) ([(girls/N, S_2)], S_3) (that/Comp, S_4) (NP, S_5)]	likes/V $	shift S_7
[(-,S_0) ([(girls/N, S_2)], S_3) (that/Comp, S_4) (NP, S_5) (likes/V, S_7)]	$	β-reduce t_2
[(-,S_0) (girls/N, S_9)]	$	α-reduce t'_1
[(-,S_0) (NP, S_1)]	$	accept

Figure 10: Parsing of "*girls/N that/Comp John/N likes/V*"

(shifting part of it to full tree reductions). The size of the table, roughly evaluated by the sum of the number of transitions and the number of action entries, is reduced by a factor of about 75 from unmanageable 75M to 1M. We also reported in the second half of the table the application of both algorithms to a grammar with 3161 trees and 22641 nodes, extracted from the Penn Treebank [Marcus et al., 1994] with the help of Fei Xia's accurate extractor [Xia, 1999].[7]

ALGORITHM	grammar	states	transitions	conflicts	reductions	bpacks
Nederhof's	XTAG	17490	40 M	114	1.93	120
Ours	XTAG	5861	202 K	7.6	3.2	5.1
Nederhof's	Treebank	22101	60 M	639	3.5	742
Ours	Treebank	13058	844 K	15.6	5.15	12.8

Figure 11: Summary of the parsing tables generated by the algorithms

5 Conclusion and Future Research

We aim at building a parser that, parsing sentences from left to right using LR techniques, is able to take sound decisions towards reaching their correct analysis. The algorithm we presented here is suited for a non-lexicalized TAG, and in fact the results we presented here are all for the non-lexicalized component (set of template trees) of LTAG grammars where the terminals are parts of speech. We were able to produce a parse table of small size, with reasonably low degree of conflicts, considering the degree of ambiguity inherent to the sets of trees.

The misprediction of bottom trees, due to lack of the valid prefix property, remains a concern. It is not clear whether a rescue of that property would have some significant impact in reducing conflicts in our current algorithm. Of course, as we said before, the property does not hold for the general case. But we are studying the subclass of TAG grammars for which our algorithm would never mispredict to see whether they are adequate for modeling Natural Language. That would mean, among other things,

[7]We set the parameters of the extractor to build a grammar with the same set of symbols as in the Penn Treebank, which is much higher than in the XTAG grammar, especially the terminals. In the version that converts the alphabet to the one of the XTAG project the numbers are much smaller for both algorithms, that is, the automaton makes much less distinction among states.

that function $GOTO_{adj}$ would depend only on one state (q_2 in Figure 6), a significant improvement in size.

We are currently working on an extension that takes into account the lexical items of the lexicalized LTAG grammars while keeping the size of the parsing table manageable. The effort we made in reducing the size of the generated table is essential due to the expected scaling up with the inclusion of lexicalization (which has already been confirmed in preliminary experiments).

The algorithm could be extended to have reductions dependent on lookaheads (e.g., an LALR(1) version), what is reasonably easy to do, that would not increase the number of states or transitions and would likely reduce substantially the degree of conflicts. The reason we did not do it is because we intend to use empirical evidence from annotated corpora to rank the conflict alternatives, for any given pair (state, lookahead), in the lexicalized version. Of course this subsumes the effect of the LR(1) version.

References

[Aho et al., 1986] Alfred Aho, Ravi Sethi, and Jeffrey Ullman. 1986. *Compilers: principles, techniques, and tools.* Addison-Wesley, Reading, MA, USA.

[Joshi and Schabes, 1996] Aravind Joshi and Yves Schabes. 1996. Tree-adjoining grammars. In *Handbook of Formal Languages and Automata.* Springer-Verlag, Berlin.

[Kinyon, 1997] A. Kinyon. 1997. Un algorithme d'analyse LR(0) pour les grammaires d'arbres adjoints lexicaliseées. In D. Genthial, editor, *Quatrième conférence annuelle sur Le Traitement Automatique du Langage Naturel, Actes*, pages 93–102, Grenoble, France.

[Knuth, 1965] Donald E. Knuth. 1965. On the translation of languages from left to right. *Information and Control*, 8(6):607–639.

[Marcus et al., 1994] Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the 1994 Human Language Technology Workshop.*

[Nederhof, 1998] Mark-Jan Nederhof. 1998. An alternative LR algorithm for TAGs. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 16th International Conference on Computational Linguistics*, Montreal, Canada.

[Schabes and Vijay-Shanker, 1990] Y. Schabes and K. Vijay-Shanker. 1990. Deterministic left to right parsing of tree adjoining languages. In *Proceedings of 28th Annual Meeting of the Association for Computational Linguistics*, pages 276–283, Pittsburgh, Pennsylvania, USA.

[Schabes, 1990] Yves Schabes. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars.* Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

[Xia, 1999] Fei Xia. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium(NLPRS-99)*, Beijing, China.

[XTAG Research Group, 1998] The XTAG Research Group. 1998. A Lexicalized Tree Adjoining Grammar for English. Technical Report IRCS 98-18, University of Pennsylvania.

Parsing Scrambling with Path Set: A Graded Grammaticality Approach

Siamak Rezaei (Durroei)
Université du Québec à Montréal
Montréal, Canada
rezaei_s@gip.uqam.ca

Abstract

In this work we introduce the notion of path set for parsing free word order languages. The parsing system uses this notion to parse examples of sentences with scrambling. We show that by using path set, the performance constraints on scrambling such as Resource Limitation Principle (RLP) can be represented easily. Our work contrasts with models based on the notion of immediate dominance rule and binary precedence relations. In our work the precedence relations and word order constraints are defined locally for each clause. Our binary precedence relations are examples of fuzzy relations with weights attached to them. As a result, the word order principles in our approach can be violated and each violation contributes to a lowering of the overall acceptability and grammaticality. The work suggests a robust principle-based approach to parsing ambiguous sentences in verb final languages.

1 Introduction

Numerous formalisms and systems have been designed for representing the grammar of free word order languages [10], [18], [19], [9], [15], [7], [14]. Each formalism has tried to capture some examples of local scrambling or long distance scrambling. Some of the formalisms considered the role of discourse in scrambling and the fact that under a specific intonation, one word order may be more acceptable. Another issue which has not been thoroughly investigated is the notion of *acceptability* itself and implementing this imperfect notion for scrambling cases. Recently, notions such as probability, optimality, possibility, plausibility, acceptability and graded grammaticality have been incorporated into the linguistic theories. Despite the fact that scrambling and word order introduce a degree of acceptability and graded grammaticality, nevertheless the necessary acceptability or plausibility notions have not been added to the scrambling rules.

Modelling graded grammaticality has been neglected in many of the past works, and [2] is one of the few that tried to incorporate them into the linguistic competence. Is graded grammaticality part of competence or performance or part of both?

Graded grammaticality and its interaction with word order constraints have also been studied from another perspective in *performance* models for languages [6]. The main problem is that not many significant theoretical works have been done to incorporate graded grammaticality in a unified model of competence and syntax. The lack of methods for gathering data and formal models of graded grammaticality are also complicating the problem.

For a flexible word order language such as Persian, an Indo-European language spoken in the Middle East, accounting for graded grammaticality is essential because there are different levels of ambiguity in the grammar that interact together. In Persian the subject and object of a sentence can be missing (i.e. Pro-drop property) and subject and object marking is ambiguous in some cases. The notion of *specificity* which is a graded notion in Persian plays an important role in the disambiguation between subject and object. So modelling graded grammaticality becomes essential and it interacts with the word order rules.

In a computational framework, one can model graded grammaticality as a form of competition among a set of alternatives with different degrees of grammaticality. In a competition framework, the result depends on the entities that are taking part. The violation of the principles of the grammar reduces the graded amount of grammaticality (i.e. acceptability) for each alternative. Competition in a grammar can arise for acquiring the highest degree of grammaticality among a set of plausible interpretations, but competition can also arise for limited linguistic *resources*. What are these resources and are there specific principles in languages that put further restrictions for acquiring these resources? We will answer these questions in the specific domain of modelling Persian and the scrambling in its word order.

In this paper we will look at some of these issues and by introducing competition and parallelism at the same time. We avoid some of the problems of backtracking and the inefficiency that it causes. We will further investigate linguistic limitations which one can impose on the processing architecture to restrict some of the possible alternatives. For this purpose we turn to recent proposals for adding resource limitation strategies to the processing [8].

The structure of the paper is the following: in Section 2 we discuss the details of the parser. Section 3 illustrates the parser. In Section 4 we discuss some major aspects of the parser.

2 A Pipeline Parser

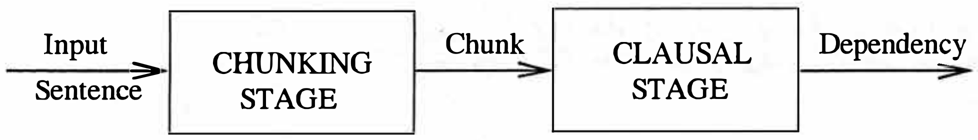


Figure 1: Parser Modules

Based on the grammar of Persian and previous experience in parsing Persian by PATR-II [16] and [17] we have implemented a two level parsing system.

(1) **Main Body:**
PAR (run in parallel)
a. parse-chunk(Pipe) to read a word and output a chunk on the pipe.
b. parse-clause(Pipe) to read a chunk from the pipe and output dependencies.

The first level of the parser, which is a variant to the PATR-II system, groups the words of the sentence into chunks: NP, PP, V, and Comp using context-free phrase structure rules. As soon as a chunk is found it is passed to the second level of the parser. The two stages are run in parallel in contrast to [12] (which the parser is based on). Abney [1] uses a similar notion of pipeline parsing. He refers to the first stage as *chunk level* and to the second stage as the level of *simplex clauses*. Abney uses a finite-state cascade and his system uses finite-state models for grammatical representation at both stages. Instead of finite-state models we have used an extension to CFG rules in the first stage and regular grammars for the second stage. CFGs are more flexible and powerful in representing constituents with levels of recursion. We have also introduced a look ahead for these rules at the first stage.

For representing scrambling we have extended the regular grammar rules for clauses with a special path set that keeps record of possible interpretations for the arguments of the clause. This path set is used to represent competition for grammatical functions and backtracking is avoided. It is updated incrementally. For example if the first constituent can be attached to the clause as SUBJect and OBJect, and if the next constituent can be attached as both SUBJect and OBJect, then the path set will include all possible combinations of: [SUBJ.SUBJ, OBJ.SUBJ, SUBJ.OBJ, OBJ.OBJ]. Some of these possibilities are restricted by the use of *word order constraints*. In this example SUBJ.OBJ is referred to as a *path*. Each path in the path set has an activation or possibility value attached to it which shows the plausibility of that particular path relative to the others. The value corresponding to each path in the path set is calculated based on word order constraints and the numeric values considered for each word order constraint.

In other words, in our framework the word order constraints are defined locally to a clause (and not for rules), and they specify the precedence relations between two grammatical functions. The precedence relations are probabilistic and each possible word order has a probability measure attached to it.

The word order constraints are of two types: hard and soft. The hard constraints cannot be violated, while the soft ones can be violated. The violation of a hard constraint makes the corresponding path inactive, while the violation of a soft constraint reduces the level of activity of that specific path. For simplicity we assume that the activity level is the same as a probability number.

In the following we will explain the details of the system and will elaborate on hard and soft constraints that put restrictions on these alternatives (paths).

2.1 First Stage

For parsing the phrase structure rules of the grammar we have used a Prolog implementation of the standard version of PATR-II.

We will first review a simple example of parsing.

The input sentence is *ali seab xord.*

(2) ali seab xord.
Ali apple ate
'Ali ate an apple.'

1. Dictionary look up:
   Input: `[ali, seab, xord].`

   Output:
   `Noun(ali,3,80), Noun(seab,3,20), Verb(xord,3,`$< Obj, Subj >$`).`

2. phrase chunking: (bottom-up)
   Input: `Noun(ali,3,80), Noun(seab,3,20), Verb(xord,3,`$< Obj, Subj >$`).`
   Output:
```
NP(dp(ali),3, ▷ obj:20 ◁, ▷ subj:80 ◁)
NP(dp(seab),3, ▷ obj:80 ◁, ▷ subj:20 ◁)
verb(verb(xord),3, < Obj,Subj >, 100)
```

Noun(ali,3,80) gives this information about *Ali* that is 3rd person singular (3) and has a specificity of 80. ▷ gram-func:activation ◁ such as ▷ obj:20 ◁ shows a pair of grammatical functions and activation values. Each constituent (chunk) may have one or more of these pairs. The number indicates the plausibility of that grammatical function for the constituent. The verb entry also shows that the verb has an object and subject and is third person singular (3). We have used an activation value of 100 to raise the activation of clauses that have verb, compared to those which lack one and are not completed. Note that we have assumed no ambiguity for the verb and hence the activation value here reflects the notion of possibilty of this interpretation.

At this stage we specify for each marked NP the possible grammatical functions that it can accept. The numbers after the grammatical functions correspond to the possibility of that alternative. These numbers are derived from the specificity value of a noun and the presence or absence of *ra* after the constituent. For example in the above *Ali* is a proper noun and it is specific. Since it is not marked by *ra* (specificity object marker), its object value is low (20%) and its subject value is high (80%). For NPs which are not marked with *ra* we have considered subjecthood equal to the specificity value and objecthood = 100 – specificity-value. We have used a numeric value for specificity because specificity of a phrase varies over a non-discrete range. In the absence of a corpus for deriving the probabilities of words and their co-occurrence we have used this notion to initialize the activation value, because we mainly use it for subject-object disambiguation which relies on specificity.

In contrast *seab* 'apple' is not a proper noun; as it is not marked by *ra*, it can be either subject or object. For objects like *seab* the subjecthood value of 20% and objecthood of 80% have been considered. This is because the corresponding specificity value for *seab* is 20. Note that one can consider different numbers, but the choice of numbers and their relation with specificity and object marking by *ra* should be taken into account.

Our goal in designing the phrase structure (PS) component of the parser was to parse the input string into chunks and pass these chunks to the next level of parsing. By using the parallelism concept of Linda, the interface between the two stages is implemented. In our model the chunks are transmitted as Linda tuples between the two stages.

2.2 Parsing Stage II

At this stage, the constituents of a clause are assembled. This stage is run in parallel with the first stage and as a chunk is produced in the first stage, the attachment of it to the clause will be started. In other words, the second stage processes the chunks incrementally. The grammatical knowledge at this stage is represented procedurally and the parser gets the incoming chunks (from previous stage) and adds them to the clause that it is currently processing.

Depending on the incoming chunk, there are different cases. The finite-state model of this module is shown in Figure 2.

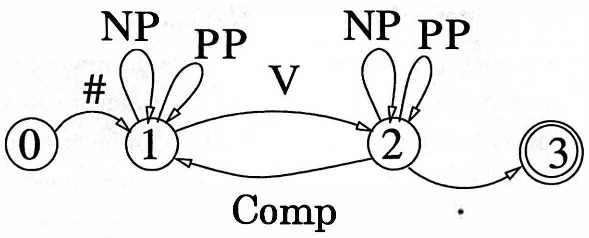


Figure 2: Second Stage

```
(3)   Main Body:
      parse-clause(Pipe)
        a. Initialize a new Clause
        b. Input a Chunk from the pipe.
        c. Do while Chunk not end of sentence.
         i. If Chunk is a complementizer: attach-new-clause(Clause).
         ii. else If Chunk is a phrase: attach(Chunk,Clause).
        Input a new Chunk from the pipe.
```

In the first case (c.i) the chunk will be added to the present clause and the parser continues with reading the next chunk and adding it to the present clause. In the other case (c.ii), the parser spawns a new clause and initialises the variables of the clause with chunks that can be exported into it. In this way the parser represents long distance scrambling and control. When the parser reaches the end of the sentence, the work of the parser is completed.

```
(4)   Main Body:
      attach(Chunk,Clause)
        a. If Chunk is NP:
         i. Add grammatical functions of the Chunk to the Path-set of Clause.
         ii. Use Apply-filter to impose word order constraints on the new Path-set.
         iii. Update Export-set (and Import-set) for long distance scrambling.
        b. If Chunk is PP:
         i. Add the grammatical function of the Chunk to the Clause.
         ii. Block ungrammatical parses which violate the word order Principles.
         iii. Update Export-set (and Import-set) for long distance scrambling.
        c. If Chunk is Verb:
         i. Add the subcategorization frame of the Chunk as expected-resources of the Clause.
```

For representing local scrambling, we have used the notion of the path set. This notion allows one to have competing alternatives of plausible word orders and rank them according to some constraints.

The word order constraints that we have considered are listed in Table 1. The word order constraints are designed to reduce the activity of those alternatives which are not marked and which deviate from the canonical word order. A zero in the precedence constraint imposes a hard constraint to filter out illegal word orders[1]. A non-zero value imposes a soft constraint to reduce the activation value for non-canonical word orders.

[1]We introduced the notion of filtering in the algorithm (4).

Implemented	Constraint	Explanation
Yes	precede(obj,subj, 0.90)	Subjects normally precede objects
Yes	precede(v,obj, 0.20)	Objects in most cases precede verbs
Yes	precede(v,subj, 0.20)	Subjects in most cases precede verbs
No	precede(obj,topic, 0)	Topics always precede objects
No	precede(obj2,obj, 0)	Object always precede object2
No	precede(obj2,subj, 0)	Subjects always precede object2

Table 1: Precedence Constraints in the Second Stage

3 Parsing Local Scrambling

The constituent rules in this stage are simple CFG rules. A clause can be generated as a result of the combination of a clause and a constituent, or it can introduce a new embedded clause, or by reaching the end of sentence, a clause can be terminated.

These automata do not specify the precedence relations between the constituents, and a separate Linear Precedence component imposes the precedence constraints. This is done incrementally and as a constituent is added to a clause, all the possible word order constraints are applied between it and the constituents which are already part of the clause. Note that we haven't considered any immediate dominance (ID) component and the binary precedence relations are not imposed on sisters of an ID rule. The system parses a sentence by initialising a clause and attaches the incoming chunks to this clause.

For (2), repeated in (5), the first chunk is *Ali*. As a result of incremental attachment at this stage we will have:

(5) ali seab xord.
Ali apple ate
'Ali ate an apple.'

np([0,1],▷ obj:20 ◁,▷ subj:80 ◁)
Rule: Clause → NP Clause

Candidates: ▷ [0,1]:obj ◁ 44.72
▷ [0,1]:subj ◁ 89.44

We have kept the indexes for each constituent. For example [0,1] shows that this constituent starts at point 0 and ends at point 1 in the input string. We use these indexes in generating the output dependecies for the parser. The parser generates these after it reaches the end of the clause (not sentence) which it is parsing.

The candidates also show the competing paths in the path set for each clause. At the beginning when the clause is initiated this path set is empty and after parsing the first constituent, the candidates (or path set) will be initiated. It is at this stage that the activation values for each path will be calculated. We have assumed 100 as the initial number for an empty clause and when it is combined separately with 20 and 80, the results will be 44.72 and 89.44. 100 is the maximum value of activation and the activation value can range from 0 to 100. We have used the square root function because it implements blocking constraints as reduction by multiplication to zero.

$\sqrt{20 \times 100} = 44.27$

The second chunk is *seab* and as a result of multiplication, we will have 4 grammatical-function pairs as potential candidates in the path set: subj.obj, obj.subj, obj.obj, subj.subj.

np([1,2],$\triangleright$ obj:80 $\triangleleft$,$\triangleright$ subj:20 $\triangleleft$)

Rule: Clause $\rightarrow$ NP Clause

Candidates: $\left| \begin{array}{l} \triangleright \text{[0,1]:obj} \triangleleft\triangleright \text{[1,2]:subj} \triangleleft 28.37 \\ \triangleright \text{[0,1]:subj} \triangleleft\triangleright \text{[1,2]:obj} \triangleleft 84.58 \end{array} \right|$

At this stage only two of the four possible alternatives can pass the filters. Since no sentence can have two objects or two subjects, the activation values of those sequences which have two subjects or two objects are reduced to zero and only two will survive.

verb-comp([2,3],$< Obj, Subj >$,100)

Rule: Clause $\rightarrow$ V-comp Clause

Candidates: $\left| \begin{array}{l} \triangleright \text{[0,1]:obj} \triangleleft\triangleright \text{[1,2]:subj} \triangleleft 28.37 \\ \triangleright \text{[0,1]:subj} \triangleleft\triangleright \text{[1,2]:obj} \triangleleft 84.58 \end{array} \right|$

When the verb is added with the activation of 100, those subjects which don't agree with verb will be deleted. Since both of the subjects agree with the verb, both alternatives will survive[2]. Finally, for the attachment of the arguments to the verb the path with the highest activity (acceptability) will be chosen and the arguments are bound to the verb. Since no word order constraint has been violated the activation value will be $91.97 = \sqrt{84.58 \times 100}$.

Note that with the same constituents and a different order, the constraints will interact to yield a different measure of acceptability. For (6) the acceptability measure is 89.58. This is because the example with canonical word order is considered more correct.

(6) seab ali xord.
apple Ali ate
'Ali ate an apple.'

In this example, the object precedes the subject and hence violates the canonical word order. As a result the activation will be multiplied by 0.90 (see Table 1 for precedence rules). A simple matrix for deriving the acceptability measure can be calculated by multiplying the values for the constraints which were violated. One can calculate all violations and yield a final violation measure and multiply the end result with this number. Instead we have multiplied each violation as soon as it is found. This incremental approach ensures that alternatives which are reduced to zero are not further extended.

Finally in Persian, PPs can scramble freely and in parsing them we do not add them to the competition (unmarked) set, because they contribute the same to all the competing paths. Instead of adding them to all paths we factor them out and store them in another marked structure because their contribution to all parallel paths are similar.

The subcat(egorisation) expectations of the verb are also added to a separate structure in our model and when the end of the clause is reached the resources and the expectations are matched with each

[2]For avoiding confusion, we have not shown the agreement features in the examples.

other and the dependency links are generated. In our model we choose the path with the highest activation and discard the other ones.

Note the difference in order of items for subcat resources and the normal resources. In subcat we have subj:[4,5] where [4,5] corresponds to the location of the verb in the sentence, while in the unmarked resources we have [0,1]:subj (note the difference in the order of grammatical relation and the brackets in the two). When the parser reaches the end of a clause, the highest active path in unmarked will be selected and the matching /bf subj resource and subcat /bf subj resource[3] are joined.

As a result of joining these two a dependency link with value [0,1]:subj:[4,5] will be generated. This depicts a transaction or communication across a subject communication link; in this transaction [0,1] is the producer and [4,5] the receiver. Similarly two other transactions for obj and pp(ba) links will be generated by this distributed approach to communication resources.

If some of the resources in marked or unmarked parts could not be unified by a corresponding element in the subcategorization frame of the verb, then these resources are moved into the embedded clauses. This is because of long distance scrambling in Persian in which some resources might belong to other embedded clauses. It is also possible that some of the expectations of the verb might not be satisfied due to the nature of Pro-drop in Persian for the arguments of a verb[4]. The feature EXT(ernal)-COM(munication) is introduced for these cases and is a set based extension to traditional gapping for long distance scrambling.

To sum up, in parsing local scrambling, as the unmarked arguments (subject, object) are added incrementally to the clause, the parser creates a parallel set of the plausible paths. The paths are restricted by some constraints. The constraints on local scrambling can be divided into hard and soft constraints. The hard constraints are strict precedence relations and verb-subject agreement which could block a path by reducing its activation value to zero. The other constraints which only reduce or increase the activation values to a non-zero value are soft constraints. The accumulative result of these values contributes to the possibility (activation) of a solution. The most active solution or path (i.e. among unmarked paths) will be chosen. Depending on the function which we use we will have different results. The constraints can be summerised as:

- Word order restrictions. These were illustrated in 1 and are used to penalize possible alternatives which deviate from the canonical word order. They also block alternatives which violate obligatory word order rules.

- Verb subject agreement. In Persian a subject must agree with the verb of the clause.

- One example of each resource in the sentence or Resource Limitation Principle (RLP).

(7) **Resource Limitation Principle**
No two NPs can exist in a clause with the same grammatical function.

In the rest of this section we will elaborate on RLP. Consider example (8).

(8)	amir	seab	be man	qol=dad	[ke	be	ali	bedahad].
	Amir	apple	to me	promise=gave-3S	that	to	Ali	gave-3S

‘Amir promised me to give the apple to Ali.’

[3]In our model we consider grammatical relations as pairs of links that attach NPs and Verbs. These links can be considered as communication resources between two linguistic processes [17].

[4]The number of fulfilled expectations can contribute to the activation positively, but we have not considered it in our implementation.

In Persian it is not possible for two grammatical resources with the same grammatical function to appear in the same clause. Hence (9) is ungrammatical.

(9) * amir seab be man be ali qol=dad [ke – bedahad].
Amir apple to me to Ali promise=gave-3S that – gave-3S
'Amir promised me to give the apple to Ali.'

In other free word order languages such as German, such an example with two dative NPs does not create a problem.

This performance constraint that we call Resource Limitation Principle (RLP) is not restricted in Persian to datives, and no clause can exist in which two phrases (resources) have the same grammatical function. RLP has been implemented in our system as a general constraint that a resource cannot precede another with the same case or grammatical function (as is implemented in our system). We have used an extension to blocking word order restrictions. For example the constraint that 'no subject can precede another subject' implements the existence of at most one subject-marked resource in a clause.

4 Discussion

The path set implements a mechanism for modelling competition among a set of paths. Corresponding to each path we had an activation value. By assigning a value to each path, we implemented a numerical notion for competition. Such competition notion is robust enough to capture soft and hard constraints for word order rules. Each word order rule had numeric value attached to it. In modelling competition numerically, one can easily represent blocking as reduction of the activation value to zero. Another advantage is that such a mechanism can be extended to allow robustness and degrees of ungrammaticality. In the following we will discuss major aspects of our work in comparison to other approaches.

Comparison With Classical Word Order Rules

ID/LP [3] can be considered as the classical approach for representing flexible word order. ID/LP uses a set of immediate dominance (ID) rules and a distinct component for linear precedence (LP) which specifies the precedence relations between the right-hand side sisters in ID rules.

Unlike a phrase structure (PS) rule which specifies two distinct relations of ID and LP at the same time, the order of the constituents in an ID rule is specified separately by LP component and in this way ID/LP format captures word order generalisations. The advantages arising from factoring out the ordering component from constituency rules are particularly evident in the case of languages with a flexible word order.

The linear precedence relations in LP component are binary relations and they can only be specified for two sister categories in the right hand side of an ID rule. As a result, no precedence relation can be specified for two categories which do not occur as sisters of a single ID rule.

Another restriction of classical ID/LP rule is the prohibition against referring to the categories inside the internal structure of phrases, in the LP relations. In other words, the LP relations can only specify relations between two sisters in an ID rule and not relations between one sister and another category dominated by the other sister.

In our approach we haven't employed ID rules and instead have used regular rules which allow different word orders. The possible word orders are restricted by a separate notion of word order binary constraints which restrict the possible order of grammatical relations in the paths.

Reape introduces word order domains to deal with word order in Germanic [15]. In his approach, the word order domains of the constituents that join with each other are merged. Unlike the word order domain in Reape's notation, in our approach we only allow one instance of a grammatical resource to be present in a word order path in each domain and each word order domain can consist of a set of parallel competing word order paths. Corresponding to each possible word order path in the word order domain, we have an activation measure.

The activation measure for a specific word order path is reduced if a word order constraint is violated. The reduction corresponds to the strength of that constraint and the stronger the constraint, the bigger the reduction. In the case of hard constraints, violation of a constraint makes the word order illegal and blocks that word order path.

In this way relaxation of word order constraints can be achieved. A general restriction of classical LP constraints is that they cannot be relaxed and they must always be satisfied. [18] proposes to extend these by use of complex LP constraints. In complex LP constraints, as long as at least one of the LP rules is satisfied the other LP rules can be violated. Our approach is a numerical extension to LP rules which allows the possibility of relaxing the LP rules. Introducing a complex LP extension is also feasible in the model to have non-binary word order constraints.

Similar to fuzzy logic sets, one can consider linguistic measures for referring to different relaxation possibilities for each word order rule and a linguist can use these for encoding the strength of the word order rules. Ideally, the relaxation of word order relations and their strengths should be derived from a corpus of texts, so that the most dominant word order gets the highest activation. In other words, these word order rules are statistically prevalent and are designed in such a way that the less plausible word orders get penalized and their activation gets reduced.

In our framework we have used the unmarked order as the most optimal path and deviations from this unmarked order are penalized. This approach can be considered as an extension to a notion of optimal parsing introduced in [6] based on typological research.

Comparison With Other Approaches

Another alternative to modelling the competition and word order constraints is to use Optmality Theory (OT). OT [13] uses a notion of constraint ranking. For parsing free word order languages the cumulative sum of constraints from Syntax, Semantics, Discourse and world knowledge determines the grammaticality of an utterance and the preference of one alternative over another. It is not clear whether one can come up with a constraint ranking of these separate modules.

[11] has also shown that for implementation of OT in a finite-state framework, one should restrict the OT model and a subset of it be considered. Another alternative is to use a cumulative and weighted approach to ranking the constraints[5] that we have adopted.

A general criticism to Optimality Theory (OT) is that the ranking of constraints does not allow any cumulative effect in which a number of lower ranked constraints can compete against a higher ranking constraint. This so-called 'ganging up' effect can be represented by using a numerical representation. OT is a limited case of such a numerical representation with no 'ganging up' effect.

[5]See [4] for another discussion on optimality ranking vs. weighted constraints.

A further criticism to OT parsing model has been raised in the literature [5]. In OT model of parsing, only the most harmonic alternative will be selected and the algorithm does not allow for a number of alternatives with a lower degree of harmony to compete in parallel with the most harmonic alternative, so that if the most harmonic alternative fails, one of the alternatives with lower degree of harmony be selected for the parse to continue. In our work, we have adopted a parallel competitive model that allows a number of alternatives to be run in parallel. This also allows a degree of robustness to be incorporated into the parser in the future.

	PERSIS85	R. Ghasem91	Rezaei92	Rezaei93	Riazati97	SHIRAZ	system
Approach	Production Rule	Conceptual Dependency	ATN	ID/LP	KIMMO PATR	feature str/type	PATR path/LP
Parser	Bottom-Up (BUP)	Procedural	BUP Top Down	BUP	BUP	BUP	BUP
Tokenization	NO	NO	NO	NO	NO	YES	NO
Morphology	NO	NO	NO	NO	YES	YES	NO
Explicit Ezafe	YES	YES	YES	YES	YES	NO	YES
Coordination	YES	NO	YES	NO	NO	YES	NO
Local Scram.	V-final	unrestricted	YES	YES	Limited	V-final	YES
Complement Cl.	YES	NO	NO	YES	NO	NO?	YES
Relative Cl.	YES	NO	NO	YES	NO	YES	NO
Long Dis. Scram.	NO	NO	NO	Fronting	NO	NO	YES
Control	NO	NO	NO	NO	NO	NO	YES
Multiple Parses	NO	NO	NO	YES	YES	YES	NO

Table 2: Comparison and Evaluation

In Table 2 we have contrasted the implemented system with the previous systems for parsing Persian. The system is developed with the goal of complementing the capabilities of the previous systems and it has its limitations. For further details of the above systems see [17].

5 Conclusion

The framework that we discussed in this paper provides a method for adding the notion of graded grammaticality to the principles of the grammar. In traditional approaches to principle based approaches a principle can be satisfied or violated. In our view some of the principles of the grammar can be violated, but the overall relaxation of the principles (when added together) should not reduce the acceptability of the solution below a certain level.

The acceptability of a particular solution is reduced by a factor whenever a principle of the grammar is violated. This factor depends on the contribution and importance of that specific principle.

By adding features to the word order rules, we can introduce more complex word order rules to take into account features such as animacy.

In our work we have introduced performance constraints for scrambling and we discussed the Resource Limitation Principle (RLP) for local scrambling. There is another counterpart to RLP for long distance scrambling - the Resource Barrier Principle (RBP) - that we have also implemented. These performance constraints restrict the scrambling in some free word order languages. They can be used to classify free word order languages.

References

[1] Steven Abney. Partial Parsing via Finite-State Cascades. In *Proceedings of the Workshop on Robust Parsing at Eighth Summer School in Logic, Language and Information*, pages 8–15, August 1996.

[2] Noam Chomsky. Degrees of Grammaticalness. In *Jerry A. Fodor and Jerrold J. Karz (eds), The Structure of Language: Readings in the Philosophy of Language*, pages 384–389. Printice Hall, 1964.

[3] Gerald Gazdar, Ewan Klein, Geoff Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar*. Harvard University Press, Cambridge,Massachusetts, 1985.

[4] E. Gibson and Brienhier. Optimality Theory and Human Sentence Processing. MIT Working Papers in Linguistics (In press), 1998.

[5] Mark Hale and Charles Reiss. What an OT parser tells us about the initial state of the grammar. In *Proceedings of the GALA'97 conference on Language Acquisition*, pages 352–357, Edinburgh, UK, April 1997.

[6] John A. Hawkins. A Parsing Theory of Word Order Universals. *Linguistic Inquiry*, 21(2):223–264, 1990.

[7] Beryl Hoffman. *The Computational Analysis of the Syntax and Interpretation of Free Word Order in Turkish*. PhD thesis, Dept. of Computer and Information Science, University of Pennsylvania, 1995.

[8] Mark Johnson. Resource Sensitivity in Grammar and Processing. In *The Ninth Annual CUNY Conference on Human Sentence Processing*, New York, 1996.

[9] Ronald M. Kaplan and Annie Zaenen. Long-Distance Dependencies, Constituent Structure, and Functional Uncertainty. In *Mark R. Baltin; Anthony S. Kroch (eds) Alternative Conceptions of Phrase Structure*, Chicago and London, 1989. The University of Chicago Press.

[10] L. Karttunen and M. Kay. Parsing in a Free Word Order Language. In *D. Dowty, L. Karttunen and A. Zwicky (eds.), Natural Language Parsing, Psychological, Computational, and Theoretical Perspective*, pages 279–306, Cambridge University Press, Cambridge, 1985.

[11] Lauri Karttunen. The Proper Treatment of Optimality in Computational Phonology. In *Proceedings of International Workshop on Finite-state Methods in Natural Language Processing (FSMNLP'98)*, Bilkent University, 1998.

[12] Michael B. Kashket. Parsing a Free-Word Order Language: Warlpiri. In *24th proceedings of the ACL*, pages 60–66, 1986.

[13] Alan Prince and Paul Smolensky. Optimality Theory: Constraint Interaction in Generative Grammar. Technical Report RuCCS Technical Report #2, Centere for Coginitve Science, Rutgers University, October 1993.

[14] Owen Rambow and Aravind K. Joshi. A Processing Model for Free Word-Order Languages. In *Perspectives On Sentence Processing, edited by Charels Clifton, Lyn Frazier and Keith Rayner*, pages 267–301, Lawrence Erlbaum Associates, New Jersey, 1994.

[15] Mike Reape. Getting Things in Order. In *Discontinuous Constituency, Harry Bunt and Arthur van Horck (eds.)*, pages 209–253, Mouton de Gruyter, 1996.

[16] Siamak Rezaei and Matthew Crocker. A Distributed Architecture for Parsing Persian. In *Proceedings of ICSCS*, Sharif Univ. of Technology, Tehran, December 1995.

[17] Siamak Rezaei-Durroei. *Linguistic and Computational Analysis of Word Order and Scrambling in Persian*. PhD thesis, Division of Informatics, University of Edinburgh, 2000.

[18] Hans Uszkoreit. Constraints on Order. Technical Report 364, SRI, October 1985.

[19] Arnold M. Zwicky. Concatenation and Liberation. In *Papers from the 22nd Regional Meeting of the Chicago Linguistic Society*, pages 65–74, 1986.

ON THE USE OF GRAMMAR BASED LANGUAGE MODELS FOR STATISTICAL MACHINE TRANSLATION

Hassan Sawaf, Kai Schütz, Hermann Ney
Computer Science Department – Lehrstuhl für Informatik VI
RWTH Aachen – University of Technology
D-52056 Aachen, Germany
`sawaf@cs.rwth-aachen.de`

Abstract

In this paper, we describe some concepts of language models beyond the usually used standard trigram and use such language models for statistical machine translation.

In statistical machine translation the language model is the a-priori knowledge source of the system about the target language. One important requirement for the language model is the correct word order, given a certain choice of words, and to score the translations generated by the translation model $\Pr(f_1^J|e_1^I)$, in view of the syntactic context.

In addition to standard m-grams with long histories, we examine the use of Part-of-Speech based models as well as linguistically motivated grammars with stochastic parsing as a special type of language model. Translation results are given on the VERBMOBIL task, where translation is performed from German to English, with vocabulary sizes of 6500 and 4000 words, respectively.

1 Introduction to Statistical Machine Translation

In this paper, we describe some methods of how to use more structured information in language modelling to improve statistical machine translation (SMT). The organisation of the paper is as follows: In this section a short introduction to SMT is given. The following section gives an overview of the different language model approaches to SMT, then our definition of performance measures and experimental results follow.

The goal of machine translation is the translation of a text given in some source language into a text in the target language. We are given a source string $f_1^J = f_1 \dots f_j \dots f_J$, which is to be translated into a target string $e_1^I = e_1 \dots e_i \dots e_I$. Among all possible target strings, we will choose the string with the highest probability:

$$\begin{aligned}\hat{e}_1^I &= \arg\max_{e_1^I} \{\Pr(e_1^I|f_1^J)\} \\ &= \arg\max_{e_1^I} \{\Pr(e_1^I) \cdot \Pr(f_1^J|e_1^I)\} \quad . \qquad (1)\end{aligned}$$

The argmax operation denotes the search problem, i.e. the generation of the output sentence in the target language. $\Pr(e_1^I)$ is the *language model* (LM) of the target language, which will be investigated in this paper, whereas $\Pr(f_1^J|e_1^I)$ is the *translation model* that is the main topic in [1, 9, 11, 12, 13, 14]. In this work the alignment template approach as described in [11] is used.

2 Language Models for SMT

Especially for the task of translation, the need of more restriction in the LM prove to be necessary. Restriction of the LM of the target language intuitively should improve translation quality in that way that the translation model should allow many alignment possibilities that are then restricted by the language model. Some heuristics help SMT systems to perform better, for example by re-ordering the source sentence as in [13] or by producing permutations in the search that are scored by a LM as in [11, 17, 18]. The advantage of the latter approach is that the reordering is integrated into the search and needs not be done as a preprocessing step. For this approach a more restricting LM, in terms of linguistic constraints and the ability to model long range dependencies is helpful.

In the work of [1], where the translation direction is French to English, the robust m-gram models have been used, as they cover the need for the analysed task well. For French-English the word order difference is mostly of a local nature, LMs that model embedded long range structures seem not to be necessary.

In this paper we show experiments using following types of LMs: word based m-gram models, class based m-gram models and context free grammar based models.

2.1 Standard Word based m-Grams

The m-gram LMs have their strength in their simplicity and reliability in robustness. That is the reason why they are widely used in automatic speech recognition. They are derived as follows:

$$\begin{aligned}\Pr(e_1^I) &= \prod_{i=1}^{I} \Pr(e_i|e_1^{i-1}) = \prod_{i=1}^{I} \Pr(e_i|h_i) \\ &= \prod_{i=1}^{I} p(e_i|e_{i-m+1}^{i-1}) \quad . \end{aligned} \tag{2}$$

In equation (2) an m-gram LM is used. For the experiments shown in this paper we used absolute discounting with interpolation, since this method has succeeded in outperforming the backing-off strategy and linear interpolation [7]. On the other hand the implementation of absolute discounting with interpolation has a low cost regarding calculation complexity [16].

2.2 Part-Of-Speech based m-Grams

To cope with the problem of sparse data, class based LMs have been studied. The classes can be extracted automatically using clustering algorithms as in [6, 8] or they are defined by linguistic experts. Here we want to show the use of the linguistically motivated Part-of-Speech (POS) based m-gram LMs.

The basis of class based m-gram LMs can be written as follows:

$$\begin{aligned}\Pr(e_1^I) &= \sum_{c_1^I} \Pr(e_1^I, c_1^I) = \sum_{c_1^I} \Pr(e_1^I|c_1^I)\Pr(c_1^I) \\ &= \sum_{c_1^I} \prod_{i=1}^{I} p(e_i|c_i, h_i) p(c_i|h_i) \qquad \text{with } h_i = c_1^{i-1} \\ &\cong \prod_{i=1}^{I} \max_{c_i} \{p(e_i|c_i, h_i) \cdot p(c_i|h_i)\} \quad . \end{aligned} \tag{3}$$

In equation (3) the LM is divided into two submodels: the classification model $p(e_i|c_i, h_i)$ and the class sequence model $p(c_i|h_i)$, where e_i denotes the ith word of the hypothesised sentence, c_i a classification of this word and h_i is the sequence of preceding word classes $h_i = c_1^{i-1}$ or, using an m-gram, $h_i = c_{i-m+1}^{i-1}$.

As shown in formula (3) we use the simplification that we perform the maximisation *before* calculating the probability of the whole word sequence. This means that the found maximum is only a local maximum. Instead the maximisation should have taken place after the calculation for the whole sequence, which would require a search similar to the monotone search [12].

Using non-unique class membership adds the problem of smoothing the class probability $p(e_i|c_i, h_i)$ in formula (3). It shows that absolute discounting for disambiguation of the class here performs best. Formally the absolute discounting of the class probability is as follows:

$$p(e_i|c_i, h_i) = \max\left\{\frac{N(e_i, c_i, h_i) - b_c}{N(c_i, h_i)}, 0\right\} + b_c \cdot \frac{W - n_0(c_i, h_i)}{N(c_i, h_i)} \cdot \beta(e_i|c_i, \overline{h_i}) \quad , \tag{4}$$

where $N(.)$ denotes the count of an event. W is the size of the vocabulary and $n_0(c_i, h_i)$ denotes the number of words not seen with the class c_i and class history h_i. $\beta(e_i|c_i, \overline{h_i})$ describes a less specific distribution with the generalised class history $\overline{h_i}$. The probability distributions for the class probabilities used in the present study have the same order as the class sequence probabilities, for example if the class sequence model is a trigram, the class probability is

$$\begin{aligned} p(e_i|c_i, c_{i-1}, c_{i-2}) &= \max\left\{\frac{N(e_i, c_i, c_{i-1}, c_{i-2}) - b_{c,3}}{N(c_i, c_{i-1}, c_{i-2})}, 0\right\} && (5)\\ &\quad + b_{c,3} \cdot \frac{W - n_0(c_i, c_{i-1}, c_{i-2})}{N(c_i, c_{i-1}, c_{i-2})} \cdot p(e_i|c_i, c_{i-1}) \quad ; \\ p(e_i|c_i, c_{i-1}) &= \max\left\{\frac{N(e_i, c_i, c_{i-1}) - b_{c,2}}{N(c_i, c_{i-1})}, 0\right\} && (6)\\ &\quad + b_{c,2} \cdot \frac{W - n_0(c_i, c_{i-1})}{N(c_i, c_{i-1})} \cdot p(e_i|c_i) \quad ; \\ p(e_i|c_i) &= \max\left\{\frac{N(e_i, c_i) - b_{c,1}}{N(c_i)}, 0\right\} && (7)\\ &\quad + b_{c,1} \cdot \frac{W - n_0(c_i)}{N(c_i)} \cdot p_{c,0} \quad ; \\ p_{c,0} &= \frac{1}{W} \quad . && (8) \end{aligned}$$

As with absolute discounting in standard language modelling the calculation of the probability $p(e_i|c_i, c_{i-1}, c_{i-2})$ consists of a *trigram* (5), a *bigram* (6), a *unigram* (7) and a *zerogram* (8) portion, which is given by the inverse of the number of vocabulary entries.

2.3 Stochastic Context Free Parser

When using m-gram LMs, some linguistic phenomena are not captured very well, because they do not model long range dependencies and embedded structures. A possible solution to this problem the use of context free grammars (CFG) as LMs [2, 18].

Formally a CFG is a quadruple $\mathcal{G} = (V_N, V_T, R, S)$, where V_N is the set of all nonterminals, V_T the set of terminals, R is the set of rules and S denominates the starting symbol that has to cover the analysed sentence.

To use CFG we implemented a stochastic parser using the stochastic version of the algorithm introduced by Cocke, Younger and Kasami [4, 19], by assigning a probability $p(r : A_n \rightarrow A_\alpha A_\beta | A_n)$ to each rule $r : A_n \rightarrow A_\alpha A_\beta$. The best hypothesis is the sequence of rule applications that produces the whole sentence from the nonterminal S and has the highest probability. The probabilities are trained using the Viterbi approximation so that only the best parse is used to calculate the rule probabilities.

If we assume the rule probabilities to be independent of each other and the class probabilities of the Part-of-Speech tags distributed uniformly, the LM probabilities can then be written as:

$$\begin{aligned}
\Pr(e_1^I) &= \sum_{r_1^N} p(r_1^N : S \xrightarrow{N} e_1^I | S) \\
&= \sum_{r_1^N} \prod_{n=1}^{N} p(r_n|A_n) \quad , \quad \text{with } A_n \text{ being the left-hand side of rule } r_n \quad , \\
&\cong \max_{r_1^N} \prod_{n=1}^{N} p(r_n|A_n) \quad . \qquad (9)
\end{aligned}$$

In order to include the tagging process into the stochastic parser, we change the format of the standard Chomsky Normal form (CNF) in the following way. We distinguish three types of rules and associated probabilities:

- **tagging rule**

  $c_i \rightarrow e_i$

  with probability $p(c_i \rightarrow e_i|c_i)$,

- the so-called **lexical rule**

  $A_\alpha \rightarrow c_i$

  with probability $p(A_\alpha \rightarrow c_i|A_\alpha)$, and

- the so-called **structure rule**

  $A_\alpha \rightarrow A_\beta A_\gamma$

  with probability $p(A_\alpha \rightarrow A_\beta A_\gamma|A_\alpha)$.

For words that are not observed in the training corpus, a simple backing-off smoothing technique is used for tagging to be able to tag these unknown words. Note that, according to equation (9), the optimal POS tags are determined only *after* the whole sentence has been parsed. In that sense the tagger uses a global sentence level criterion rather than a local decision criterion.

The search for the best parse is done using dynamic programming over the positions $1 < i, j < I$. The recursion formula for the stochastic Cocke-Younger-Kasami-style parser (SCYK) using the Viterbi approximation is based on partial hypotheses Q derived from (9) with

$$Q(j, i|A_n \rightarrow A_\alpha A_\beta) = p(A_n \rightarrow A_\alpha A_\beta|A_n) \max_{j<l<i-1} \left(Q(j, l|A_\alpha)\, Q(l+1, i|A_\beta)\right) \quad .$$

We use two speed-up methods for the SCYK:

top-down filtering: The parsing proceeds mainly bottom up, apart from a top-down filtering method that does not affect the parsing accuracy. The filtering limits the number of nonterminals that can produce the hypothesised sentence fragment, allowing only nonterminals that can be reached from a special position within the sentence.

bounding: Another implemented feature to improve speed is that a calculation is cancelled, if the scores of partial hypotheses do not reach a lower bound. The main idea is that a product of two probabilities cannot be higher than the smaller of the two probabilities.

To be able to use grammars that are not in CNF, we implemented an algorithm that converts any CFG into CNF. The standard algorithm is sketched in table 1 [3].

Table 1: Standard algorithm for converting a general CFG into CNF.

```
while ∃ rule $r : A \rightarrow A'$ in productions $R$
    foreach occurrence of nonterminal $A$ in $R$
        foreach rule $r'$ with $A$ on the left-hand side
            add rules by applying $r'$ to all rules with $A$ on the
            right hand side into $R$
    remove rule $r$ from $R$
while ∃ rule $r : A \rightarrow A_1A_2A_3 \dots A_I$ in $R$ which violates CNF
    add new nonterminal $A^*$ to $V_N$
    add new rule $r^*$ with $r^* : A^* \rightarrow A_1A_2$ to $R$
    rewrite rule $r$ as $r : A \rightarrow A^*A_3 \dots A_I$
```

The problem of this algorithm is that the number of newly added rules increases rapidly so that the memory requirement for the CNF grammar is very high and also time complexity increases. The reason of this increase is that the complexity of the SCYK is $O(|V_N| \cdot |R| \cdot I^3)$. According to this formula we would prefer a CNF grammar with minimal numbers $|R|$ of rules and $|V_N|$ nonterminals.

Here, we introduce an algorithm that uses bigram frequencies to determine which pair of nonterminals to choose when generating a new rule (see algorithm in table 2). Thus we can reduce the number of rules significantly.

Table 2: Improved algorithm for converting a general CFG into CNF.

```
while ∃ rule $r : A \rightarrow A'$ in productions $R$
    foreach occurrence of nonterminal $A$ in $R$
        foreach $r'$ with $A$ on the left-hand side
            add rules by applying $r'$ to all rules with $A$ on
            the right hand side into $R$
    remove rule $r$ from $R$
while ∃ rule $r : A \rightarrow A_1A_2 \dots A_I$ in $R$ which violates CNF
    create 2-dimensional bigram count table $b(\cdot,\cdot)$ over
    all nonterminals $A_\alpha, A_\beta$ on the right hand side
    add new nonterminal $A^*$ to $V_N$
    add new rule $r^*$ with $r^* : A^* \rightarrow A_\alpha A_\beta$ to $R$,
    where $b(A_\alpha, A_\beta)$ has highest value
    rewrite all rules, replacing successive nonterminals
    $A_\alpha A_\beta$ on the right hand side
```

There must be some considerations as to what to do regarding the rule probabilities when transforming the grammar. The easiest way to handle the probabilities is as follows: when applying a rule, the rule probabilities are multiplied as usual. When adding a rule to the productions R, the new rule r' has probability $p(r' : A' \rightarrow A_\alpha A_\beta | A') = 1$.

The implemented parser is constructed in such a way so that the hypothesised sentence is processed left-to-right that means the chart for the SCYK is constructed along the covered positions of the sentence instead of the standard way, where the chart is constructed along the depth of the parse subtrees. In such a way, we can use the parser as a left-to-right LM that can be embedded into the translation approaches, e.g. described in [9, 11, 12] or in speech recognition systems that build up the recognised sentence incrementally.

2.4 Linear Interpolation of Language Models

Experiments show that every LM type has some advantages compared to the other LMs, but also bears some weakness so that it would be best to use all LMs at the same time. An m-gram LM for instance has its strength in robustness and we expect the grammar based LM to model long range dependencies better.

A very easy method to combine two LMs $p_1(\cdot)$ and $p_2(\cdot)$ is to use *linear interpolation* [8] at the full-sentence level:

$$Pr(e_1^I) = (1-\alpha) \cdot p_1(e_1^I) + \alpha \cdot p_2(e_1^I) \quad \text{with } 0 < \alpha < 1 \quad .$$

3 Parsing Performance

The "VERBMOBIL Task" [15] is a speech translation task in the domain of appointment scheduling, travel planning and hotel reservation. The translation direction is from German to English which poses special problems due to the big difference in the word order of the two languages.

To perform experiments with the SCYK parser, we used the English part of the VERBMOBIL treebank [5]. Table 3 shows some statistics about the investigated corpora. For the performance tests of the parser, we used the standard training and test set that consist of about 9000 and 500 trees, respectively, where every tree corresponds to one sentence. To train the parser for rescoring the translation results, we used all available trees in the VERBMOBIL treebank, i.e. about 21,000 sentences.

Table 3: VERBMOBIL treebank characteristics.

corpus	train	test	all
trees	9126	500	20,889
running words	81,382	4331	185,084
vocabulary size	1710	498	2168
avg. sentence length	8.88	8.66	8.86
avg. tree depth	6.58	6.55	6.58
avg. nodes per tree	11.25	10.8	11.12
$\|V_N\|$	96		99
$\|V_T\|$	72		76
$\|R\|$	4287		6260
avg. rule length	3.54		3.68
unseen words		31	
unseen rules		143	
trees with unseen rules		113	

Results for the time complexity are shown in table 4. As described in chapter 2.3 we used two methods, namely the top-down filtering and bounding to speed up the parsing process. These two methods are included in 4.

Table 4: Time performance of SCYK of the speed-up methods.

	avg. time/sentence
baseline	246.7 ms
+ top-down filtering	226.4 ms
+ bounding	63.78 ms
+ bounding + top-down filtering	62.82 ms

The enhanced transformation method for converting a generic grammar into Chomsky Normal form compared to the standard method is presented in table 5. The corpus used here is the above mentioned VERBMOBIL standard treebank test corpus.

Table 5: Results for conversion into Chomsky Normal form.

method	$\|V_N\|$	$\|R\|$
no conversion	99	6260
standard	10932	17017
improved	1170	7255

Table 6 shows the parsing performance on the VERBMOBIL treebank test corpus. When looking at the results shown in table 6, we should keep in mind that there was a certain number of unseen words (see table 3). Due to this number of unseen words the tagging accuracy decreases from 96.73% to 95.87%, the complete match from 51.6% to 49.6% and bracketing recall and precision drop by about one percent absolute respectively.

Table 6: Parsing results on standard testset of VERBMOBIL.

	SCYK
number of sentences	500
number of unparsed sentences	0
number of valid sentences	500
bracketing recall	84.84 %
bracketing precision	84.82 %
complete match	49.60 %
average crossing	0.43
no crossing	80.40 %
2 or less crossing	93.80 %
tagging accuracy	95.87 %

When using no smoothing technique for tagging, the parser would not parse 31 out of the 500 sentences. It has also to be mentioned that 22.6% of the test sentences contain rules that cannot be matched by the rules generated from the training corpus. That means, the highest possible value for complete match would be 77.4%.

4 Translation Results

4.1 The VERBMOBIL Task

For translation experiments we used the bilingual text corpus of the VERBMOBIL task. The text input was obtained by manually transcribing the spontaneously spoken sentences. There was no constraint on the length of the sentences, and some of the sentences in the test corpus contain more than 50 words. Therefore, for text input, each sentence was split into shorter units using the punctuation marks. The segments thus obtained were translated separately, and the final translation was obtained by concatenation.

Table 7 shows a summary of the corpus used for the experiments. Here the term word refers to full-form word as there is no morphological processing involved.

Table 7: Training and test conditions for the VERBMOBIL Task.

		German	English
Train	sentences	34 465	
	running words	363 514	383 509
	vocabulary size	6 381	3 766
Test	sentences	147	
	running words	1 968	2 173

4.2 Performance Measures

In the translation experiments, we use the following performance measures [10]:

- mWER (multi-reference word error rate):
  A known weakness of the *word error rate* that is widely used for speech recognition is that a translation of a given sentence is not unique so that there can be more than one translation for a source sentence. A possible solution for this is to use a set of several possible translations for each source sentence. The mWER is the Levenshtein distance to the most similar sentence of this set.

- SSER (subjective sentence error rate):
  For a more detailed analysis, subjective judgements by test persons are necessary. Each translated sentence is judged by a human examiner according to an error scale from 0.0 to 1.0 in eleven steps. A rating of 0.0 means that the translation is semantically and syntactically correct and a rating of 1.0 means that the sentence is semantically wrong, i.e. either the produced sentence has no sense at all or the produced sense does not convey the sense of the source sentence. The human examiner was offered the translated sentences for the different LMs at the same time.

4.3 Translation Results for VERBMOBIL

In Table 8 some results for the different LMs for SMT are presented. The results are calculated by rescoring the 100 best hypotheses of each sentence of the test set using the alignment template approach. The hypotheses are then re-calculated by multiplying the translation model scores and the language model scores like in formula 1. The pure POS-based LMs for SMT show relatively poor performance compared to word based LMs, only if using an interpolation of both word and POS-based LMs the performance improves slightly. Table 8 contains the results of using the different LMs in terms of mWER and SSER.

Table 8: Translation performance on VERBMOBIL

LM		mWER[%]	SSER[%]
word based	2-gram	37.76	23.54
	3-gram	35.40	22.59
	4-gram	35.61	23.20
	5-gram	35.45	22.45
	10-gram	34.89	22.11
POS based	2-gram	39.00	24.83
	3-gram	38.34	22.52
	4-gram	38.09	23.47
	5-gram	37.36	23.95
	10-gram	37.43	24.15
stochastic CFG		36.55	22.31
linear interpolation of			
word 5-gram + POS 5-gram		34.13	21.22
POS 5-gram + SCFG		31.95	20.48

Using long history length seems to perform better for m-gram LMs both word and POS-based. The SCFG alone does not improve translation results compared to the word based LM. The interpolation of both POS-based LM and SCFG achieves the best results. Linear interpolation of word and POS-based LMs also achieves better results than POS-based or word based LMs alone but does not reach the performance of the combination of POS-based LM and SCFG.

Analysing the sentences chosen from the different LM types we can observe that the SCFG is superior to the m-gram LMs in modelling nested structures and long range dependencies as can be seen in table 9. In each of the two cases for linear interpolation shown in table 9, the interpolation factors were 0.5, which produced the best results.

The sentences show that the syntactic quality increases when using more linguistic information for SMT. The third sentence in table 9 shows the advantage of using grammar based LMs. The corresponding sentence of the word based model contains the problem that the verb group for this sentence in the source language is composed of two parts: *am* produced from the first part *habe* and *make* produced from *ausgemacht*, are positioned relatively far from each other within the sentence. The coherence of these two words is thus not detected by the m-gram LMs. The SCFG, however, can detect this and constructs a better verb phrase.

For the third translation example in table 9 it should be noted that the list of 100 sentences does not include the correct sentence. After adding the correct sentence to the list, the system produced the correct translation.

5 Conclusion

In this paper we discussed the use of linguistically motivated language models for statistical machine translation, namely Part-of-Speech based m-gram models and stochastic context free grammars. The results of the different language model types and the interpolation of the language models are then

Table 9: Translation examples on VERBMOBIL task and reference translation.

German	also, ich habe Unterlagen über Flüge da.
reference	well, I have information about flights here.
word based	well, I have about brochures flights.
word+POS	well, I have flights about brochures.
POS+SCFG	well, I have papers about flights then.
German	ich könnte erst eigentlich jetzt wieder dann November vorschlagen.
reference	actually I could only suggest November then.
word based	now again then actually I could only suggest November.
word+POS	I could only suggest again now actually of November then.
POS+SCFG	I could suggest again then now actually first of November.
German	also ausgerechnet habe ich am dritten Juli schon einen Zahnarzttermin ausgemacht.
reference	well, on the third of July I have already made a dentist's appointment.
word based	so, I am on the third of July already make a dentist appointment.
word+POS	well, I have the third of July already make a dentist appointment.
POS+SCFG	well, on the third of July I have got already make a dentist appointment.

presented on the VERBMOBIL task. It shows to be a promising approach to use a stochastic context free parser as syntax-restricting language model and to interpolate it with a Part-of-Speech based language model.

In the near future the language models introduced here will be integrated into the translation process itself instead of using rescoring. In [7] different interpolation methods are compared and the linear interpolation was found to be worse than log-linear interpolation methods. Therefore these interpolation methods should be examined for language models for statistical machine translation in the future. Also refined grammar based models should be investigated, for instance usage of lexicalized grammars or stochastic attribute grammars. The usage of morphological analysis should achieve a gain in syntactic quality for the produced sentences.

Acknowledgement

This work has been partly supported as part of the VERBMOBIL project (contract number 01 IV 601 A) by the German Federal Ministry of Education, Science, Research and Technology.

The authors would like to thank Franz Josef Och for his fruitful discussions and Richard Zens for his support in performing the tests with the m-gram LMs.

References

[1] P. F. Brown, V. J. Della Pietra, S. A. Della Pietra, and R. L. Mercer: The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, Vol. 19, No. 2, pp. 263-311, 1993.

[2] C. Chelba, F. Jelinek: Recognition Performance of a Structured Language Model. *6th European Conference on Speech Communication and Technology*, pp. 1567-1570, Budapest, Hungary, September 1999.

[3] J. E. Hopcroft, J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation*, Addison Wesley, Reading, MA, 1979.

[4] T. Kasami: An Efficient Recognition and Syntax Analysis Algorithm for Context Free Languages. Scientific Report AFCRL-65-758, Air Force Cambridge Research Laboratory, Bedford, MA, 1965.

[5] V. Kordoni: Stylebook for the English Treebank in VERBMOBIL. Technical report, University of Tübingen, Tübingen, Germany, 1998.

[6] S. C. Martin, J. Liermann, H. Ney: Algorithms for Bigram and Trigram Word Clustering. *Speech Communication*, Vol. 24, No. 1, pp. 19-37, April 1998.

[7] S. C. Martin, C. Hamacher, J. Liermann, H. Ney: Assessment of Smoothing Methods and Complex Stochastic Language Modelling. *6th European Conference on Speech Communication and Technology*, pp. 1939-1942, Budapest, Hungary, September 1999.

[8] H. Ney, F. Wessel, S. C. Martin: Statistical Language Modelling Using Leaving-One-Out. In S. Young, G. Bloothooft (eds.): *Corpus-Based Methods in Speech and Language*, Kluwer Academic Publishers, Dordrecht, The Netherlands, pp. 174-207, 1997.

[9] S. Nießen, S. Vogel, H. Ney, C. Tillmann: A DP-Based Search Algorithm for Statistical Machine Translation. *36th Annual Conference of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pp. 960-967, Montreal, Canada, August 1998.

[10] S. Nießen, F. J. Och, G. Leusch: An Evaluation Tool for Machine Translation: Fast Evaluation for MT Research. *submitted to 2nd International Conference on Language Resources and Evaluation*, Athens, Greece, May 2000.

[11] F. J. Och, C. Tillmann, and H. Ney: Improved Alignment Models for Statistical Machine Translation. *Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pp. 20-28. University of Maryland, College Park, MD, June 1999.

[12] C. Tillmann, S. Vogel, H. Ney, A. Zubiaga: A DP-based Search Using Monotone Alignments in Statistical Translation. *35th Annual Conference of the Association for Computational Linguistics*, pp. 289-296, Madrid, Spain, July 1997.

[13] C. Tillmann, S. Vogel, H. Ney, A. Zubiaga, H. Sawaf: Accelerated DP Based Search for Statistical Translation. *5th European Conference on Speech Communication and Technology*, pp. 2667-2670, Rhodos, Greece, September 1997.

[14] S. Vogel, H. Ney, C. Tillmann: HMM-Based Word Alignment in Statistical Translation. *International Conference on Computational Linguistics*, pp. 836-841, Copenhagen, Denmark, August 1996.

[15] W. Wahlster: VERBMOBIL: Translation of Face-to-Face Dialogs. *Machine Translation Summit IV*, pp. 127-135, Kobe, Japan, 1993.

[16] F. Wessel, S. Ortmanns, H. Ney: Implementation of Word Based Statistical Language Models. *2nd SQEL Workshop on Multi-Lingual Information Retrieval Dialogues*, pp. 55-59, Pilsen, Czech Republic, April 1997.

[17] D. Wu: A Polynomial-Time Algorithm for Statistical Machine Translation. *34rd Annual Conference of the Association for Computational Linguistics*, pp. 152-158, Santa Cruz, CA, 1996.

[18] D. Wu, H. Wong: Machine Translation with a Stochastic Grammatical Channel. *36th Annual Conference of the Association for Computational Linguistics and the 17th International Conference on Computational Linguistics*, pp. 1408-1415, Montreal, Canada, August 1998.

[19] D. H. Younger: Recognition and Parsing of Context Free Languages in Time n^3. *Information and Control*, 10:2, pp. 189-208, 1967.

ALGEBRAIC CONSTRUCTION OF PARSING SCHEMATA

Karl-Michael Schneider
Department of General Linguistics
University of Passau
Innstr. 40, 94032 Passau, Germany
schneide@phil.uni-passau.de

Abstract

We propose an algebraic method for the design of tabular parsing algorithms which uses parsing schemata [7]. The parsing strategy is expressed in a tree algebra. A parsing schema is derived from the tree algebra by means of algebraic operations such as homomorphic images, direct products, subalgebras and quotient algebras. The latter yields a tabular interpretation of the parsing strategy. The proposed method allows simpler and more elegant correctness proofs by using general theorems and is not limited to left-right parsing strategies, unlike current automaton-based approaches. Furthermore, it allows to derive parsing schemata for linear indexed grammars (LIG) from parsing schemata for context-free grammars by means of a correctness preserving algebraic transformation. A new bottom-up head corner parsing schema for LIG is constructed to demonstrate the method.

1 Introduction

Linear indexed grammars (LIG) [2] and tree adjoining grammars (TAG) [4] are weakly equivalent grammar formalisms that generate an important subclass of the so-called mildly context-sensitive languages (MCSL). In recent publications (see for example [1, 5] and the papers cited there) the design of parsing algorithms for LIG and TAG is based on an operational model of (formal) language recognition. It consists of the construction of some nondeterministic push-down automaton from a grammar, depending on the parsing strategy, and a tabular interpretation of that automaton. This approach is modular because the tabulation of the automaton is independent of the parsing strategy.

Besides its obvious advantages over a direct construction of parsing algorithms (as in [9]), this approach is still dissatisfying in two respects: First, the tabulation of a LIG automaton is motivated only informally, in terms of a certain non-contextuality of LIG derivations (i.e., parts of LIG derivations do not depend on the bottom parts of the dependent stacks) or in terms of an efficient representation of unbounded LIG stacks. Second, because the usual push-down automata read their input sequentially from left to right, this technique cannot be applied straightforwardly to head-corner strategies, which start the recognition of an input string in the middle.

In this paper we present an algebraic approach to the design of parsing algorithms. By this we mean that a parsing algorithm is derived from an algebraic specification of a parsing strategy by means of algebraic operations such as homomorphic images, direct products, subalgebras and quotient algebras. A parsing strategy is expressed through the operations in an algebra where the objects are partial parse trees (called a *tree algebra*). A second algebra (called *yield algebra*) describes how the input string is processed.

Following [7] we do not construct parsing algorithms but rather *parsing schemata*, i.e., high-level descriptions of tabular parsing algorithms that can be implemented as tabular algorithms in a canonical way [8]. A parsing schema describes the items in the table and the steps that the algorithm performs in order to find all valid items, but leaves the order in which parsing steps are performed unspecified.

Our approach picks up an idea originally proposed but not fully developed[1] by Sikkel [7] whereupon a parsing schema could be regarded as the quotient (with respect to some congruence relation) of a *tree-based parsing schema*. A parse item is seen as a congruence class of a partial parse tree for some part of the input string. The problem is that items that do not denote a valid parse tree for some part of the input string cannot be described in this way because they would denote empty congruence classes. In our approach a parse item is seen as a pair $(a_{\simeq}, \xi)$ where $a_{\simeq}$ is a congruence class of trees and ξ denotes a substring of the input string. In this way all items can be characterized algebraically. This allows us to lift the correctness proof from the level of items to the level of trees.

In this paper we construct a new bottom-up head-corner (buHC) parsing schema for LIG to demonstrate the algebraic approach. The construction proceeds in two steps: In the first step we construct a buHC parsing schema for context-free grammars (CFG) algebraically and give a correctness proof. In the second step an algebraic, correctness preserving transformation is applied to the tree algebra of this parsing schema to obtain a buHC parsing schema for LIG. The transformed tree algebra implements the non-contextuality of LIG derivations into the tree operations and thus makes this notion more precise.

Our approach has a series of advantages over the automaton-based construction of parsing algorithms: It is not limited to parsing strategies that process the input string from left to right; it provides a precise characterization of an item in terms of congruence classes; it allows simpler and more elegant correctness proofs by means of general algebraic theorems; it allows to derive parsing schemata for LIG from parsing schemata for CFG by means of algebraic transformations; and finally, it provides a precise explanation for certain characteristics of LIG parsing algorithms.

The paper may be outlined as follows: In Sect. 2 we define the basic algebraic concepts used in this paper. Sect. 3 presents a short introduction to parsing schemata and describes the general method of constructing parsing schemata algebraically. In Sect. 4 we show the algebraic construction of the buHC parsing schema for CFG, and in Sect. 5 we define an algebraic transformation that yields a buHC parsing schema for LIG. Sect. 6 presents final conclusions.

2 Nondeterministic Algebras

In this section we present generalized versions of standard concepts of Universal Algebra [3] for algebras with nondeterministic operations, called *nondeterministic algebras*, which provide the basis for the algebraic description of parsing schemata. The theorems in this section are given without proof, the proofs can be found in [6]. Although nondeterministic variants of algebras have been defined previously, for example relational systems [3], most concepts of Universal Algebra have been fully developed only for algebras with deterministic operations.

An algebra $\mathcal{A}$ is a pair (A, F) where A is a nonvoid set (the *carrier* of $\mathcal{A}$) and F is a family of finitary operations $f : A^n \to A$. An *n-ary nondeterministic operation* is a set-valued function $f : A^n \to \mathcal{P}(A)$, where $\mathcal{P}(A)$ denotes the powerset of A. We use the notation $f(a_1, \ldots, a_n) \vdash a$ iff $a \in f(a_1, \ldots, a_n)$. If

[1] although it must be pointed out that Sikkel's book is not about the algebraic structure of parsing schemata in the first place, but about relations between different parsing schemata.

f is nullary we write $f \vdash a$ instead of $f() \vdash a$. A *nondeterministic algebra* $\mathcal{A}$ is a pair (A, F) where A is a nonvoid set and F is a family of finitary nondeterministic operations. A *partial algebra* is a nonvoid set A together with a family of finitary partial operations on A, i.e., partial functions $f : A^n \rightharpoonup A$. The *type* of a (partial, nondeterministic) algebra is the function that assigns each operation its arity. We write $\mathcal{A} = (A, F)$ and $\mathcal{B} = (B, F)$ to indicate that $\mathcal{A}$ and $\mathcal{B}$ are algebras of the same type, although the operations can be defined differently in $\mathcal{A}$ and $\mathcal{B}$. To indicate whether an operation $f \in F$ belongs to $\mathcal{A}$ or $\mathcal{B}$ we write $f^{\mathcal{A}}$ and $f^{\mathcal{B}}$, but we omit the superscripts if the algebra is understood.

The *restriction* of a nondeterministic operation $f : A^n \to \mathcal{P}(A)$ to a subset $B \subseteq A$ is the operation $f' : B^n \to \mathcal{P}(B)$ defined by $f'(b_1, \ldots, b_n) = f(b_1, \ldots, b_n) \cap B$. Let $\mathcal{A} = (A, F)$ be a nondeterministic algebra. The smallest subset A' of A that is closed under the operations in F (i.e., if $a_1, \ldots, a_n \in A'$, $n \geq 0$, and $f(a_1, \ldots, a_n) \vdash a$ then $a \in A'$) is denoted with $[\emptyset]_{\mathcal{A}}$. $[\emptyset]_{\mathcal{A}}$ is nonempty only if $\mathcal{A}$ has nullary operations. The elements in $[\emptyset]_{\mathcal{A}}$ are said to be *generated* (by $\mathcal{A}$).

We now present some standard concepts of Universal Algebra for nondeterministic algebras. Let $\mathcal{A} = (A, F)$ and $\mathcal{B} = (B, F)$ be nondeterministic algebras of the same type. $\mathcal{B}$ is called a *relative subalgebra* of $\mathcal{A}$ if $B \subseteq A$ and for every $f \in F$, $f^{\mathcal{B}}$ is the restriction of $f^{\mathcal{A}}$ to B. $\mathcal{B}$ is called a *weak subalgebra* of $\mathcal{A}$ if $B \subseteq A$, and for every $f \in F$, for all $b_1, \ldots, b_n, b \in B$: whenever $f^{\mathcal{B}}(b_1, \ldots, b_n) \vdash b$ then $f^{\mathcal{A}}(b_1, \ldots, b_n) \vdash b$. If $[\emptyset]_{\mathcal{A}}$ is nonempty then the relative subalgebra $([\emptyset]_{\mathcal{A}}, F)$ is called the *generated subalgebra* of $\mathcal{A}$.

The *direct product* of $\mathcal{A}$ and $\mathcal{B}$ is the nondeterministic algebra $\mathcal{A} \times \mathcal{B} = (A \times B, F)$, where $f^{\mathcal{A}\times\mathcal{B}}((a_1, b_1), \ldots, (a_n, b_n)) \vdash (a, b)$ iff $f^{\mathcal{A}}(a_1, \ldots, a_n) \vdash a$ and $f^{\mathcal{B}}(b_1, \ldots, b_n) \vdash b$.

A *homomorphism* of $\mathcal{A}$ into $\mathcal{B}$ is a function $h : A \to B$ satisfying the condition: For all $a_1, \ldots, a_n \in A$, if $f^{\mathcal{A}}(a_1, \ldots, a_n) \vdash a$ then $f^{\mathcal{B}}(ha_1, \ldots, ha_n) \vdash ha$. A homomorphism h of $\mathcal{A}$ into $\mathcal{B}$ is called *strong* if for all $a_1, \ldots, a_n \in A$, for all $b \in B$: whenever $f^{\mathcal{B}}(ha_1, \ldots, ha_n) \vdash b$, then there is some element $a \in A$ such that $f^{\mathcal{A}}(a_1, \ldots, a_n) \vdash a$ and $ha = b$.

A *strong congruence relation* of a $\mathcal{A}$ is an equivalence relation $\simeq$ on A satisfying the condition: if $f(a_1, \ldots, a_n) \vdash a$ and $a_i \simeq a_i'$, for $i = 1, \ldots, n$, then there is some $a' \in A$ such that $a \simeq a'$ and $f(a_1', \ldots, a_n') \vdash a'$. The set of all strong congruence relations of $\mathcal{A}$ is denoted with $Cgr(\mathcal{A})$. Strong congruence relations and strong homomorphisms of nondeterministic algebras are related as follows [6]: The *kernel* of a strong homomorphism is a strong congruence relation, and every strong congruence relation is the kernel of some strong homomorphism.

Let $\simeq$ be a strong congruence relation of $\mathcal{A}$. The quotient algebra $\mathcal{A}/\simeq$ is the nondeterministic algebra $(A/\simeq, F)$ where the operations are defined through $f^{\mathcal{A}/\simeq}(a_{1\simeq}, \ldots, a_{n\simeq}) \vdash a_{\simeq}$ iff for some elements $a_1', \ldots, a_n', a' \in A$: $a_i' \simeq a_i$ (for all i) and $a' \simeq a$ and $f^{\mathcal{A}}(a_1', \ldots, a_n') \vdash a'$.

The following theorem describes the connection between homomorphisms and generated subalgebras of direct products:

Theorem 1. *Let $\mathcal{A}$ be a nondeterministic algebra and $\mathcal{B}$ a partial algebra of the same type and $h : \mathcal{A} \to \mathcal{B}$ a homomorphism. Then $[\emptyset]_{\mathcal{A}\times\mathcal{B}} = \{(a, b) \in [\emptyset]_{\mathcal{A}} \times [\emptyset]_{\mathcal{B}} \mid ha = b\}$.*

The next theorem shows that generated subalgebras and quotient algebras commute:

Theorem 2. *If $\simeq$ is a strong congruence relation of $\mathcal{A}$ then $[\emptyset]_{\mathcal{A}/\simeq} = [\emptyset]_{\mathcal{A}}/\simeq$.*

As a corollary, we also get the following

Theorem 3. *If $h : \mathcal{A} \to \mathcal{B}$ is a strong homomorphism then $[\emptyset]_{\mathcal{B}} = \{ha \mid a \in [\emptyset]_{\mathcal{A}}\}$.*

The last theorem can be interpreted thus: Under a strong homomorphism, computations in a homomorphic algebra are homomorphic images of computations in the original algebra.

3 Algebraic Construction of Parsing Schemata

In this section we present a formal definition of parsing schemata and describe the general scheme of the algebraic construction of parsing schemata. This scheme is completely independent of a particular grammar formalism or parsing strategy. Parsing schemata were proposed by Sikkel as a well-defined level of abstraction for the description and comparison of tabular parsing algorithms [7].

A parsing schema[2] is a deduction system (I, D) consisting of a finite set of (parse) items I and a finite set D of deduction steps written in the form $x_1, \ldots, x_n \vdash x$ (meaning x is deducible from $x_1, \ldots, x_n$) where $n \geq 0$ and $x_1, \ldots, x_n, x \in I$. The *inference relation* $\vdash$ is defined by $X \vdash x$ iff for some $x_1, \ldots, x_n \in X$: $x_1, \ldots, x_n \vdash x \in D$. The *reflexive and transitive inference relation* $\vdash^*$ is defined by $X \vdash^* x$ iff $x \in X$ or there are items $y_1, \ldots, y_m$ such that for all i, $X \cup \{y_1, \ldots, y_i - 1\} \vdash y_i \in D$ and $x = y_m$. If $X \vdash^* x$ we say that x is *deducible* from X. If x is deducible from the void set $\emptyset$ then x is called *valid*.

Let (I, D) be a parsing schema for a grammar G and an input string w. Every item $x \in I$ represents a G-derivation of a particular form of some substring of w. If such a derivation actually exists then x is called *correct*. (I, D) is called correct if valid and correct items coincide. If (I, D) is correct then a string w is in the language of G iff an item representing a G-derivation of w from the start symbol of G is valid.

Let G be a grammar. An *(augmented) tree algebra* for G is a nondeterministic algebra $\mathcal{A}_G = (A, F)$ where A is a set of partial derivation trees augmented with some state information (that depends on the parsing strategy) and F is a family of (possibly nondeterministic) tree operations that depend on the grammar G as well as the parsing strategy. We assume that F contains at least one nullary operation. In the sequel we will assume that G is understood and write $\mathcal{A}$ instead of $\mathcal{A}_G$.

Let Σ be an input alphabet. An *(augmented) yield algebra* is a partial algebra $\mathcal{B} = (B, F)$ where B is a set of strings from Σ^* augmented with some state information. A homomorphism $g : \mathcal{A} \to \mathcal{B}$ (where $\mathcal{A}$ is an augmented tree algebra and $\mathcal{B}$ is an augmented yield algebra) that assigns each augmented tree the augmented string of terminal symbols at its leaves is called a *yield homomorphism.*

Let $\mathcal{A}$ be an augmented tree algebra and $\mathcal{B}$ an augmented yield algebra and $g : \mathcal{A} \to \mathcal{B}$ a yield homomorphism. Let $\simeq$ be a strong congruence relation of $\mathcal{A}$. For any string $w \in \Sigma^*$ let $B(w) \subseteq B$ be the set of all augmented substrings of w (the exact definition of *substring* depends on the parsing strategy). Let $\mathcal{B}(w) = (B(w), F)$ be the relative subalgebra of $\mathcal{B}$ with carrier $B(w)$. The nondeterministic algebra $\mathcal{A}/\!\simeq \times \mathcal{B}(w)$, that is, the direct product of the quotient algebra of $\mathcal{A}$ and the relative subalgebra of $\mathcal{B}$ with augmented substrings of w, is called a *parsing algebra* for G, w. The elements of a parsing algebra are pairs $(a_{\simeq}, b)$ where $a_{\simeq}$ is a congruence class of an augmented derivation tree and b is an augmented substring of w. By Theorems 1 and 2, $(a_{\simeq}, b)$ is generated in the parsing algebra iff there is some generated derivation tree a' in $\mathcal{A}$ such that $a'_{\simeq} = a_{\simeq}$ and $ga' = b$.

Let $w \in \Sigma^*$ be an input string. An augmented substring of w may be given by a tuple $\xi \in \mathbb{N}^m$ of positions in w. Two different tuples ξ, ζ may determine the same augmented substring of w. A *parse item* is a pair $(a_{\simeq}, \xi)$ where $a_{\simeq}$ is an equivalence class of augmented derivation trees and ξ is a tuple of positions. A *parse item algebra* is a nondeterministic algebra $\mathcal{I} = (I, F)$ where I is a set of parse items. A *parse item homomorphism* is a strong homomorphism φ from a parse item algebra into a parsing algebra, such that $\varphi(a_{\simeq}, \xi) = (a_{\simeq}, b)$, i.e., φ maps the second component of a parse item to an augmented substring of w. If $\varphi : \mathcal{I} \to \mathcal{A}/\!\simeq \times \mathcal{B}(w)$ is a parse item homomorphism, then

[2]We use a slightly different notation and terminology than that in [7].

by Theorem 3, a parse item $(a_{\simeq}, \xi)$ is generated in $\mathcal{I}$ iff for some $b \in B(w)$, $\varphi(a_{\simeq}, \xi) = (a_{\simeq}, b)$ and $(a_{\simeq}, b)$ is generated in the parsing algebra.

A parsing schema (I, D) is obtained from a finite parse item algebra (I, F) by defining $D = \{x_1, \ldots, x_n \vdash x \mid \exists f \in F : f(x_1, \ldots, x_n) \vdash x\}$. Then by the previous equivalences, $(a_{\simeq}, \xi)$ is deducible in the parsing schema iff for some $a' \in A$, a' is generated in $\mathcal{A}$ and $a'_{\simeq} = a_{\simeq}$ and $ga' \in B(w)$ and $\varphi(a_{\simeq}, \xi) = (a_{\simeq}, ga')$. Note that if φ is a parse item homomorphism of a finite parse item algebra into a parsing algebra, then there are only finitely many congruence classes of *generated* augmented derivation trees.

A nondeterministic algebra $\mathcal{A}$ is called sound (resp. complete) w.r.t. a set $A_0 \subseteq A$ iff $[\emptyset]_{\mathcal{A}} \subseteq A_0$ (resp. $[\emptyset]_{\mathcal{A}} \supseteq A_0$). A nondeterministic algebra is called *correct* iff it is sound and complete (w.r.t. a set A_0). The grammar G defines a subset $A_0 \subseteq A$ of *admissible* augmented derivation trees. Note that A_0 depends on the parsing strategy but not on F. If a parsing schema (I, D) for G, w is constructed as above and the tree algebra is correct w.r.t. admissible derivation trees of G, then a parse item $(a_{\simeq}, \xi)$ is deducible in (I, D) iff $a_{\simeq}$ is the congruence class of some admissible derivation tree a' and ga' is the augmented substring of w denoted by ξ; that is, (I, D) is correct.

By definition, $w \in L(G)$ iff there is a derivation of w from some start symbol of G. An element in A that represents a derivation of a string $w \in \Sigma^*$ from a start symbol is called a *complete (augmented) derivation tree.* An equivalence relation $\simeq$ on A is called *regular* if there are no mixed equivalence classes; that is, if the condition holds: whenever a is complete and $a \simeq a'$ then a' is complete, too. If $\simeq$ is regular then $a_{\simeq}$ is called complete iff a is complete. A parse item $(a_{\simeq}, \xi)$ where $a_{\simeq}$ is complete is called a *final item.* If (I, D) is correct for G, w then $w \in L(G)$ iff there is some final item $(a_{\simeq}, \xi)$ such that $(a_{\simeq}, \xi)$ is deducible in (I, D) and ξ denotes w.

4 Context-Free Bottom-Up Head-Corner Parsing

In this section we present an algebraic description of the bottom-up head-corner (buHC) parsing schema for CFG [7, Schema 11.13], according to the construction scheme described in the previous section. A buHC parser starts the recognition of the right-hand side of a production at a predefined position (the *head* of the production) rather than at the left edge, and proceeds in both directions. In the sequel we will denote terminal symbols with $a, a_1, a_2, \ldots$, nonterminal symbols with $A, B, \ldots$, strings of terminal symbols with u, w and strings of terminal and nonterminal symbols with β, γ, δ. $|\beta|$ denotes the length of β. We borrow a practical notation for trees from [7]: $\langle A \rightsquigarrow \beta \rangle$ denotes an arbitrary tree with root symbol A and yield (i.e., sequence of labels on the leaves, from left to right) β (possibly of height 0, in which case $\beta = A$). $\langle A \to \beta \rangle$ denotes the unique tree of height 1 with root symbol A and yield β. A tree of height 1 is called a *local tree.* Expressions of this form can be nested, thus specifying subtrees of larger trees. We also write $\langle \beta \rightsquigarrow \gamma \rangle$ for a sequence of trees with root symbols β (from left to right) and concatenated yields γ.

A *headed context-free grammar* G is a tuple (N, Σ, P, S, h) such that (N, Σ, P, S) is a CFG without ε-productions, where N, Σ, P are finite sets of nonterminal symbols, terminal symbols and productions, respectively, S is a start symbol and $h : P \to \mathbb{N}$ is a function that assigns each production $p = A \to \beta$ a position $1 \le h(p) \le |\beta|$. The $h(p)$-th symbol in β is called the *head* of p (for simplicity it is assumed that the same production cannot occur twice with different heads). The pair $(p, h(p))$ is called a *headed production.* If $p = A \to \beta X \delta$ and $h(p) = |\beta X|$ then we write $A \to \underline{\beta X} \delta$ for $(p, h(p))$.

A *buHC tree* is a triple (τ, k, l) where $\tau = \langle A \to \beta \langle \gamma \rightsquigarrow u \rangle \delta \rangle$ (for some $A, \beta, \gamma, \delta, u$) is a finite,

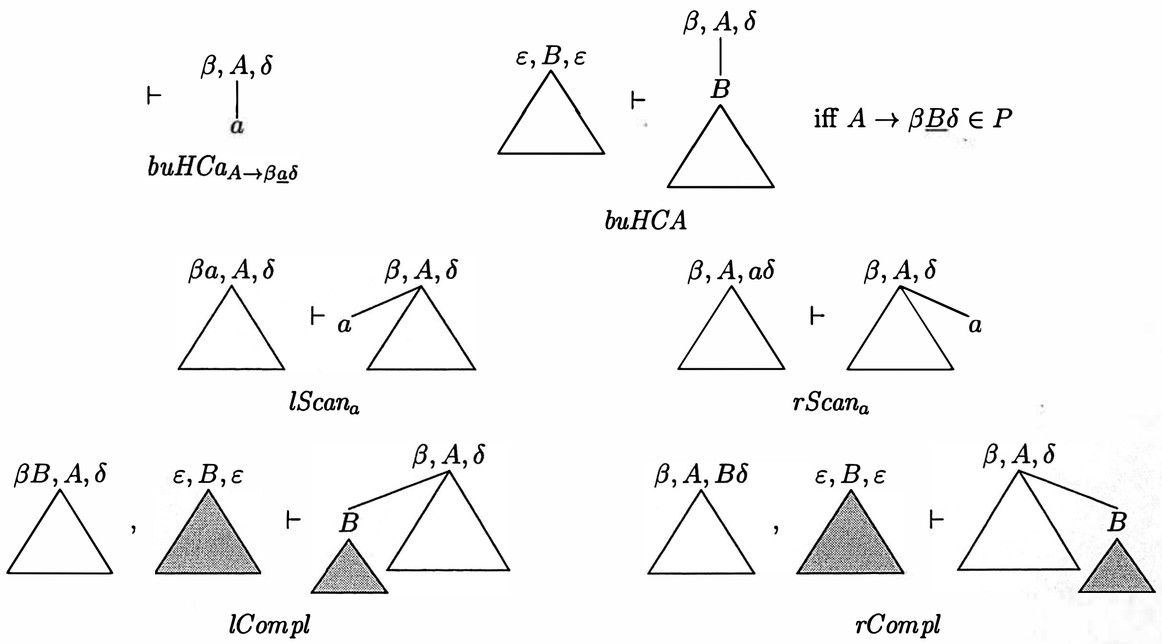


Figure 1: Bottom-up head-corner tree operations.

ordered tree with $k = |\beta|$ and $l = |\beta\gamma|$. k and l are state information; they mark the beginning and end of the recognized part γ of a production. An equivalent representation for (τ, k, l) is the triple (τ', β, δ) where $\tau' = \langle A \to \langle\gamma \rightsquigarrow u\rangle\rangle$ and the subtrees specified by $\langle\gamma \rightsquigarrow u\rangle$ are the same in τ and τ'. We use the second form in graphical representations of buHC trees and write β and δ to the left and right of the root label, respectively.

The *buHC tree operations* are shown in Fig. 1 (the yields of the trees are omitted for simplicity). ε denotes the empty string. The nullary operation $buHCa_{A\to\beta\underline{a}\delta}$ is indexed with a headed production in order to ensure that the corresponding operation in the yield algebra is a partial function. For the same reason, the operations $lScan_a$ and $rScan_a$ are indexed with the symbol a being scanned. The tree rooted by B in *lCompl* and *rCompl* is called *side tree*. We denote the buHC tree algebra with $\mathcal{A}_{buHC}$.

A local tree $\langle A \to \beta\rangle$ is called *admissible* w.r.t. G iff $A \to \beta$ is a production of G. A buHC tree (τ, k, l) where $\tau = \langle A \to \beta\langle\gamma \rightsquigarrow u\rangle\delta\rangle$ is admissible w.r.t. G iff each local tree in τ is admissible w.r.t. G and there is some headed production $(p, h(p))$ with $p = A \to \beta\gamma\delta$ and $k < h(p) \leq l$.

Proposition 1. *$\mathcal{A}_{buHC}$ is correct w.r.t. admissible buHC trees.*

Proof. Soundness is proved by induction on the basis of individual operations. To this end, observe that each operation computes only admissible trees provided that its arguments are admissible. In particular, $buHCa_{A\to\beta\underline{a}\gamma}$ is an admissible tree.

Completeness is proved by induction on the length of computations. Define the function $\lambda : (\tau, k, l) \mapsto (|\tau|, l - k)$ for any buHC tree (τ, k, l) where $|\tau|$ denotes the number of nodes in τ, and define the relation $<_T$ on $\mathbb{N}^2$ by $(m, n) <_T (m', n')$ iff $m < m'$ or $(m = m'$ and $n < n')$. $<_T$ is a well-founded, strict ordering on the λ-values of all buHC trees. Now observe that if (τ, k, l) is an admissible buHC tree then (τ, k, l) is computed by $buHCa_{A\to\beta\underline{a}\gamma}$ or we find admissible buHC trees (τ_i, k_i, l_i) for $1 \leq i \leq j$ such that (τ, k, l) is computed from (τ_i, k_i, l_i) by some j-ary buHC tree operation and $\lambda(\tau_i, k_i, l_i) <_T \lambda(\tau, k, l)$ for all i. Then completeness follows by induction on the values of λ. □

$$
\begin{aligned}
I_{buHC} &= \{[A \to \beta\bullet\gamma\bullet\delta, i, j] \mid A \to \beta\gamma\delta \in P,\ 0 \leq i < j \leq |w|\} \\
D^{buHCa} &= \{\vdash [A \to \beta\bullet a_i\bullet\delta, i-1, i] \mid A \to \beta\underline{a_i}\delta \in P\} \\
D^{buHCA} &= \{[B \to \bullet\gamma\bullet, i, j] \vdash [A \to \beta\bullet B\bullet\delta, i, j]\} \mid A \to \beta\underline{B}\delta \in P\} \\
D^{lScan} &= \{[A \to \beta a_i\bullet\gamma\bullet\delta, i, j] \vdash [A \to \beta\bullet a_i\gamma\bullet\delta, i-1, j]\} \\
D^{rScan} &= \{[A \to \beta\bullet\gamma\bullet a_{j+1}\delta, i, j] \vdash [A \to \beta\bullet\gamma a_{j+1}\bullet\delta, i, j+1]\} \\
D^{lCompl} &= \{[A \to \beta B\bullet\gamma\bullet\delta, i, j], [B \to \bullet\gamma'\bullet, k, i] \vdash [A \to \beta\bullet B\gamma\bullet\delta, k, j]\} \\
D^{rCompl} &= \{[A \to \beta\bullet\gamma\bullet B\delta, i, j], [B \to \bullet\gamma'\bullet, j, k] \vdash [A \to \beta\bullet\gamma B\bullet\delta, i, k]\} \\
D_{buHC}(w) &= D^{buHCa} \cup D^{buHCA} \cup D^{lScan} \cup D^{rScan} \cup D^{lCompl} \cup D^{rCompl}
\end{aligned}
$$

Figure 2: The buHC parsing schema.

A buHC yield is a string in Σ^*. The buHC yield homomorphism is defined as follows: For $(\tau, k, l) = (\tau', \beta, \delta)$, where $\tau' = \langle A \to \langle\gamma \rightsquigarrow u\rangle\rangle$, let $g_{\text{buHC}}(\tau, k, l) = u$. The buHC yield operations are defined as follows: $buHCa_{A\to\beta\underline{a}\delta} = a$, $buHCA(u) = u$, $lScan_a(u) = au$, $rScan_a(u) = ua$, $lCompl(u, v) = vu$, $rCompl(u, v) = uv$. We denote the buHC yield algebra with $\mathcal{B}_{buHC}$. Clearly, g_{buHC} is a homomorphism of $\mathcal{A}_{buHC}$ into $\mathcal{B}_{buHC}$.

Let $\simeq_{\text{buHC}}$ be the equivalence relation defined by $(\tau_1, k_1, l_1) \simeq_{\text{buHC}} (\tau_2, k_2, l_2)$ iff (for some $A, \beta, \gamma, \delta, u_1, u_2$) $\tau_i = \langle A \to \beta\langle\gamma \rightsquigarrow u_i\rangle\delta\rangle$ $(i = 1, 2)$ and $k_1 = k_2 = |\beta|$ and $l_1 = l_2 = |\beta\gamma|$. Then $\simeq_{\text{buHC}}$ is a strong congruence relation of $\mathcal{A}_{buHC}$. We denote the congruence class of (τ, k_i, l_i) with $[A \to \beta\bullet\gamma\bullet\delta]$. It is obvious that $\simeq_{\text{buHC}}$ partitions the set of admissible buHC trees into finitely many congruence classes, for any headed CFG G. For any $w \in \Sigma^*$ let $sub(w)$ be the set of strings u such that for some v, v': $w = vuv'$ (substrings of w), and let $\mathcal{B}_{buHC}(w)$ be the relative subalgebra of $\mathcal{B}_{buHC}$ with carrier $sub(w)$. Clearly, $\mathcal{B}_{buHC}(w)$ is finite for any w.

Now fix some input string $w = a_1 \ldots a_n$. A buHC item for G, w is a triple $((\tau, k, l)_{\simeq_{\text{buHC}}}, i, j)$, written as $[A \to \beta\bullet\gamma\bullet\delta, i, j]$, where $(\tau, k, l)_{\simeq_{\text{buHC}}} = [A \to \beta\bullet\gamma\bullet\delta]$ and $A \to \beta\gamma\delta \in P$ and $0 \leq i < j \leq n$. A pair (i, j) denotes the nonempty substring of w from position i through j. Let $I_{buHC}(w)$ be the set of all buHC items for G, w. Let φ_{buHC} be the function $[A \to \beta\bullet\gamma\bullet\delta, i, j] \mapsto ([A \to \beta\bullet\gamma\bullet\delta], a_{i+1} \ldots a_j)$. The buHC operations can be defined straightforwardly on $I_{buHC}(w)$ such that φ_{buHC} is a strong homomorphism from $I_{buHC}(w)$ into $\mathcal{A}_{buHC}/{\simeq_{\text{buHC}}} \times \mathcal{B}_{buHC}(w)$. Finally, the buHC parsing schema for G, w is obtained by defining the deduction steps as described in Sect. 3. It is shown in Fig. 2.

A buHC tree (τ, k, l) is *complete* iff τ is of the form $\langle S \to \langle\gamma \rightsquigarrow u\rangle\rangle$ and $k = 0$ and $l = |\gamma|$. Thus (τ, k, l) is complete iff $(\tau, k, l)_{\simeq_{\text{buHC}}} = [S \to \bullet\gamma\bullet]$. The following corollary follows directly from the construction:

Corollary 1. *1.* $\vdash^* [A \to \beta\bullet\gamma\bullet\delta, i, j]$ *iff there is some admissible buHC tree* (τ, k, l) *with* $\tau = \langle\beta\langle\gamma \rightsquigarrow a_{i+1} \ldots a_j\rangle\delta\rangle$ *and* $k = |\beta|$ *and* $l = |\beta\gamma|$.

2. $w \in L(G)$ *iff for some* γ, $\vdash^* [S \to \bullet\gamma\bullet, 0, n]$.

5 Bottom-Up Head-Corner Parsing of LIG

A linear indexed grammar (LIG) [2] is an extension of a headed CFG in which the productions are associated with stack operations and where the nonterminal symbols in a derivation are associated

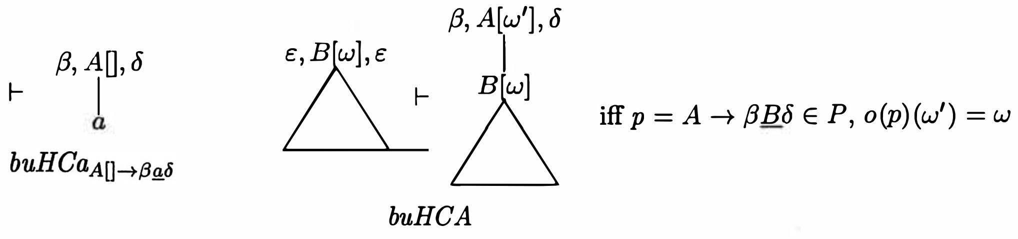

$\beta, A[], \delta$
$\vdash$
a
$buHCa_{A[]\to\beta\underline{a}\delta}$
$\varepsilon, B[\omega], \varepsilon$
$\vdash$
$\beta, A[\omega'], \delta$
$B[\omega]$
$buHCA$
iff $p = A \to \beta\underline{B}\delta \in P$, $o(p)(\omega') = \omega$

Figure 3: buHC-LIG tree operations.

with stacks of symbols. The stacks associated with the head and the left-hand side of a production are related by the stack operation associated with that production while all other descendants have a stack of bounded length. We consider a normal form of LIG where a stack operation either pushes or pops a single symbol, and where the stacks of non-head descendants must be empty.

A LIG in normal form can be represented as a tuple $(N, \Sigma, Q, P, S, h, o)$ where (N, Σ, P, S, h) is a headed CFG, Q is a finite stack alphabet and o is a function that assigns each production $p \in P$ a stack operation push_q or pop_q (where $q \in Q$) if the head of p is a nonterminal symbol, and nop otherwise. Let $q \in Q^*$ be a finite stack. The stack operations are defined as follows: $\text{push}_q(\omega) = \omega q$, $\text{pop}_q(\omega) = \omega'$ if $\omega = \omega' q$, else undefined, $\text{nop}(\omega) = \omega$.

A *headed tree* is a tree such that for each node v that is not a leaf, exactly one child of v is marked and the others are unmarked. The marked child is called the *dependent descendant* of v. A *buHC-LIG tree* is a tuple (τ, k, l) such that τ is a headed tree labeled with pairs $(X, \omega) \in (N \cup \Sigma) \times Q^*$, written as $X[\omega]$, and τ has the form $\langle A[\omega] \to \Lambda\langle\Gamma \rightsquigarrow u\rangle\Delta\rangle$, and $k = |\Lambda|$ and $l = |\Lambda\Gamma|$, where Λ, Γ, Δ denote (finite) sequences of labels in $(N \cup \Sigma) \times Q^*$, and if $X \in \Sigma$ then ω is empty. Instead of $X[]$ we can write X. Let G be a LIG. A local tree $\langle A[\omega] \to \Gamma\rangle$ is admissible w.r.t. G iff there is a production $p = A \to \gamma\underline{X}\gamma' \in P$ such that $\Gamma = \gamma X[\omega']\gamma'$ and $\omega' = o(p)(\omega)$ and the $h(p)$-th child of $A[\omega]$ is marked (note that γ, γ' denote sequences of labels with empty stacks). A buHC-LIG tree (τ, k, l) is admissible w.r.t. G iff every local tree in it is admissible w.r.t. G and, if the m-th child of the root of τ is marked then $k < m \leq l$.

Let (τ, k, l) be a buHC-LIG tree and v a node in τ with a stack of length $n \geq 1$. Consider the unique sequence of dependent descendants beginning at the dependent descendant of v and extending downwards to a leaf. If it exists, the unique node v' closest to v on this sequence with stack length $n - 1$ is called the *dependent stack descendant* of v. This means that on the path from v to v' the stack length does not fall below n except at v'. If v is the root of τ then v' is called the dependent stack descendant of τ. Note that if (τ, k, l) is admissible w.r.t. some LIG then v has a dependent stack descendant.

The buHC-LIG tree operations are obtained from the buHC tree operations by incorporating the stack operations associated with the productions. Fig. 3 shows the *buHCa* and *buHCA* operations. In the resulting trees the nodes labeled with a resp. $B[\omega]$ are marked. The other operations do not mark or unmark nodes. The *Scan* and *Compl* operations do not perform any operations on stacks, however, the *Compl* operations are only defined if the stack on the root of the side tree is empty. The buHC-LIG tree algebra is denoted with $\mathcal{A}_{buHC\text{-}LIG}$. Prop. 1 remains valid if $\mathcal{A}_{buHC}$ is replaced with $\mathcal{A}_{buHC\text{-}LIG}$ and "buHC trees" is replaced with "buHC-LIG trees", if we consider a buHC-LIG tree as a buHC tree over the infinite label domain $(N \cup \Sigma) \times Q^*$ and a LIG production as an abbreviation for an infinite set of context-free productions over this infinite domain. Note that the proof of Prop. 1

does not rely on the finiteness of N and P. However, in $\mathcal{A}_{buHC\text{-}LIG}$ we do not find a strong congruence relation with finitely many congruence classes of admissible buHC-LIG trees:

Proposition 2. *Let $\simeq$ be a strong congruence relation of $\mathcal{A}_{buHC\text{-}LIG}$. If the length of stacks in the derivation trees of G is unbounded, then $[\emptyset]_{\mathcal{A}_{buHC\text{-}LIG}}/\simeq$ is infinite.*

Proof. We give an informal proof. Consider the *buHCA* operation and a production with a push$_q$ operation. First, observe that if two admissible buHC-LIG trees are congruent then they must have the same symbol q on top of the stacks at their root nodes, because of the *buHCA* operation. Let $\omega = q_1 \dots q_m$ and $\omega' = q'_1 \dots q'_n$ be the stacks at the root nodes, where $q_m = q'_n$. By the same operation we can conclude that $q_{m-1} = q'_{n-1}$. By induction it follows that any two congruent, admissible buHC-LIG trees must have the same stack at their root nodes. Thus, if the length of stacks is unbounded, there are infinitely many noncongruent buHC-LIG trees. □

Below we will define an algebraic transformation that preserves the correctness of the transformed nondeterministic algebra under certain conditions. Then we proceed as follows: First, we replace the *buHCA* operation in Fig. 3 with two operations $buHCA_{op}$ for $op \in \{\text{push}, \text{pop}\}$, with the additional condition that $o(p) = op_q$ (for some q) in Fig. 3. Obviously, this does not affect the correctness of the tree algebra. Next, we use the transformation to modify the $buHCA_{push}$ operation. The transformed buHC-LIG tree algebra will have new congruence relations with only finitely many congruence classes of admissible buHC-LIG trees.

We first define some additional algebraic concepts. Let $\mathcal{A} = (A, F)$ be a nondeterministic algebra and $R : A^2 \to \mathcal{P}(A)$ a binary operation on A. A subset $A' \subseteq A$ is called *forward closed* w.r.t. R if for all $a_1, a_2 \in A'$: if $R(a_1, a_2) \vdash a$ then $a \in A'$. A' is called *backward closed* w.r.t. R if there is some function $d : A' \to \mathbb{N}$, such that for all $a_1 \in A'$ there is some $a_2 \in A'$ with $d(a_2) < d(a_1)$ and $R(a_1, a_2) \vdash a_1$. We denote with $\mathcal{A}[R] = (A, F \cup \{R\})$ the algebra where R has been adjoined as a new operation. $f \circ R$ denotes functional composition, i.e., $f \circ R(a_1, a_2) \vdash a$ iff for some $a' \in A$: $R(a_1, a_2) \vdash a'$ and $f(a') \vdash a$.

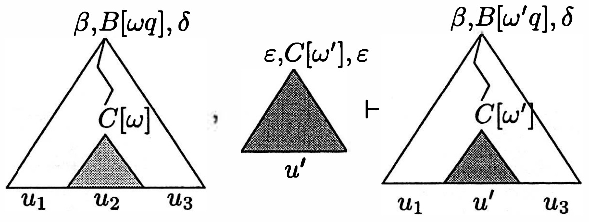
Theorem 4. *Let $\mathcal{A} = (A, F)$ be a nondeterministic algebra and $f \in F$ a unary operation and $R : A^2 \to \mathcal{P}(A)$ and let $\mathcal{A}' = (A, F')$ where $F' = F \setminus \{f\} \cup \{f \circ R\}$.*

1. *If $[\emptyset]_{\mathcal{A}}$ is backward and forward closed w.r.t. R then $[\emptyset]_{\mathcal{A}} = [\emptyset]_{\mathcal{A}'}$.*
2. *$Cgr(\mathcal{A}[R]) = Cgr(\mathcal{A}) \cap Cgr((A, R)) \subseteq Cgr(\mathcal{A}')$.*

Proof. (1) follows from the definition of closure and by induction on the values of d. (2) follows directly from the definitions. □

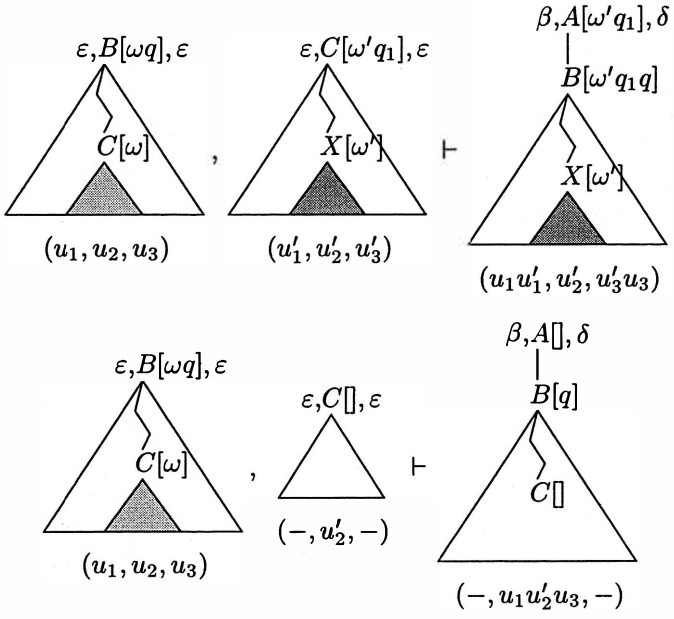
Assume that $\mathcal{A}$ is correct w.r.t. some set of admissible elements A_0, i.e., $[\emptyset]_{\mathcal{A}} = A_0$, let R be a binary operation on A such that A_0 is forward and backward closed w.r.t. R, and define $\mathcal{A}'$ as in Theorem 4. Then by Theorem 4, $\mathcal{A}'$ is also correct w.r.t. A_0. Forward closure of A_0 w.r.t. R preserves the soundness of $\mathcal{A}'$ while backward closure preserves its completeness. The second part of Theorem 4 guarantees that all strong congruence relations of $\mathcal{A}$ are preserved in $\mathcal{A}'$. More importantly, in the example below $\mathcal{A}'$ will have new (interesting) congruence relations.

Define the binary operation R as shown in Fig. 4. The lines indicate sequences of dependent descendants. $C[\omega]$ is the dependent stack descendant of $B[\omega q]$ in the left tree. Note that this implies that the stack ω is not consulted from $B[\omega q]$ to $C[\omega]$. The right tree is obtained by replacing each stack of the form $\omega q_1 \dots q_m$ $(m \geq 0)$ on the path from $B[\omega q]$ to $C[\omega]$ in the left tree with a stack of the form $\omega' q_1 \dots q_m$, and then replacing the subtree rooted by $C[\omega']$ with the middle tree. The substitution of stacks exploits the fact that the application of LIG productions on the path from $B[\omega q]$ to $C[\omega]$ does



$\beta,B[\omega q],\delta$
$C[\omega]$
u_1 u_2 u_3
$\varepsilon,C[\omega'],\varepsilon$
u'
$\vdash$
$\beta,B[\omega' q],\delta$
$C[\omega']$
u_1 u' u_3

Figure 4: Substitution in buHC-LIG trees.



$\varepsilon,B[\omega q],\varepsilon$
$C[\omega]$
(u_1, u_2, u_3)
$\varepsilon,C[\omega' q_1],\varepsilon$
$X[\omega']$
(u'_1, u'_2, u'_3)
$\vdash$
$\beta,A[\omega' q_1],\delta$
$B[\omega' q_1 q]$
$X[\omega']$
$(u_1 u'_1, u'_2, u'_3 u_3)$
$\varepsilon,B[\omega q],\varepsilon$
$C[\omega]$
(u_1, u_2, u_3)
$\varepsilon,C[],\varepsilon$
$(-, u'_2, -)$
$\vdash$
$\beta,A[],\delta$
$B[q]$
$C[]$
$(-, u_1 u'_2 u_3, -)$

Figure 5: $buHCA_{push} \circ R$.

not depend on ω. If (τ, k, l) has no dependent stack descendant then let $R(\tau, k, l)(\tau', k', l') = (\tau, k, l)$ for any buHC-LIG tree (τ', k', l').

Proposition 3. $[\emptyset]_{\mathcal{A}_{buHC\text{-}LIG}}$ *is forward and backward closed w.r.t. R.*

Proof. For forward closure, observe that every node that is not on the path from $B[\omega q]$ to $C[\omega]$ is not a dependent descendent of any node on that path, and hence the stack substitution in the left tree, together with the subtree substitution, does not affect the admissibility of any local tree in the white area.

For backward closure, observe that the right tree in Fig. 4 can be obtained by replacing the subtree rooted by $C[\omega']$ with itself in the right tree (i.e., by doing nothing), and use the number of nodes in a tree as the function d in the definition of backward closure. □

Let $\mathcal{A}'_{buHC\text{-}LIG}$ be the algebra that is obtained from $\mathcal{A}_{buHC\text{-}LIG}$ by replacing the $buHCA_{push}$ operation with $buHCA_{push} \circ R$ (see Fig. 5). By Theorem 4 and Prop. 3, $\mathcal{A}'_{buHC\text{-}LIG}$ is correct w.r.t. admissible buHC-LIG trees. Furthermore, let $\simeq_{\text{buHC-LIG}}$ be the equivalence relation defined as follows: Let (τ_1, k_1, l_1), (τ_2, k_2, l_2) be buHC-LIG trees and for $i = 1, 2$ let $\tau_i = \langle A_i[\omega_i] \to \Lambda_i\langle\Gamma_i \rightsquigarrow u_i\rangle\Delta_i\rangle$. Then $(\tau_1, k_1, l_1) \simeq_{\text{buHC-LIG}} (\tau_2, k_2, l_2)$ iff $A_1 = A_2$ and $\Lambda_1 = \Lambda_2$ and $\Gamma_1 = \Gamma_2$ and $\Delta_1 = \Delta_2$ and $k_1 = k_2$ and $l_1 = l_2$, and if $\omega_1 = \varepsilon$ then $\omega_2 = \varepsilon$, and if $\omega_1 = \omega q$ then $\omega_2 = \omega' q$, and τ_1 has a dependent stack descendant iff τ_2 has a dependent stack descendant, and if $B[]$ is the dependent stack descendant of τ_1 then $B[]$ is the dependent stack descendant of τ_2, and if $B[\overline{\omega} q']$ is the dependent stack descendant

$\vdash [A \to \beta \bullet a_i \bullet \delta, -, -, -, i-1, i, -, -] \quad (A \to \beta \underline{a_i} \delta \in P)$

$[B \to \bullet\gamma\bullet, q, C, q'', i, j, r, s], [C \to \bullet\gamma'\bullet, q_1, X, q_1', r, s, u, v] \vdash [A \to \beta\bullet B \bullet\delta, q_1, X, q_1', i, j, u, v]$
$(p = A \to \beta\underline{B}\delta \in P,\ o(p) = \text{push}_q)$

$[B \to \bullet\gamma\bullet, q, C, q'', i, j, r, s], [C \to \bullet\gamma'\bullet, -, -, -, r, s, -, -] \vdash [A \to \beta\bullet B \bullet\delta, -, -, -, i, j, -, -]$
$(p = A \to \beta\underline{B}\delta \in P,\ o(p) = \text{push}_q)$

$[B \to \bullet\gamma\bullet, q_1, C, q_1', i, j, r, s] \vdash [A \to \beta\bullet B\bullet\delta, q, B, q_1, i, j, i, j] \quad (p = A \to \beta\underline{B}\delta \in P,\ o(p) = \text{pop}_q)$

$[A \to \beta a_i \bullet\gamma\bullet\delta, q, B, q', i, j, r, s] \vdash [A \to \beta\bullet a_i\gamma\bullet\delta, q, B, q', i-1, j, r, s]$

$[A \to \beta\bullet\gamma\bullet a_{j+1}\delta, q, B, q', i, j, r, s] \vdash [A \to \beta\bullet\gamma a_{j+1}\bullet\delta, q, B, q', i, j+1, r, s]$

$[A \to \beta B\bullet\gamma\bullet\delta, q, C, q', i, j, r, s], [B \to \bullet\gamma'\bullet, -, -, -, k, i, -, -] \vdash [A \to \beta\bullet B\gamma\bullet\delta, q, C, q', k, j, r, s]$

$[A \to \beta\bullet\gamma\bullet B\delta, q, C, q', i, j, r, s], [B \to \bullet\gamma'\bullet, -, -, -, j, k, -, -] \vdash [A \to \beta\bullet\gamma B\bullet\delta, q, C, q', i, k, r, s]$

Figure 6: The buHC-LIG deduction steps.

of τ_1 then $B[\overline{\omega}'q']$ is the dependent stack descendant of τ_2, for some $\omega, \omega', \overline{\omega}, \overline{\omega}'$.

Proposition 4. *$\simeq_{buHC\text{-}LIG}$ is a strong congruence relation of $\mathcal{A}'_{buHC\text{-}LIG}$ with only finitely many congruence classes of admissible buHC-LIG trees.*

If (τ, k, l) is as in Fig. 4 (left) then let $g_{\text{buHC-LIG}}(\tau, k, l) = (u_1, u_2, u_3)$, and if $\tau = \langle A[] \to \beta\langle\Gamma \rightsquigarrow u\rangle\delta\rangle$ then let $g_{\text{buHC-LIG}}(\tau, k, l) = (-, u, -)$. Then the buHC-LIG operations can be defined on the buHC-LIG yields in a straightforward way (for example, see Fig. 5 for $buHCA_{push}$), such that $g_{\text{buHC-LIG}}$ is a homomorphism. Using the construction described in Sect. 3 (analogously to Sect. 4) we obtain a (correct!) buHC-LIG parsing schema. The buHC-LIG items are of the form $[A \to \beta\bullet\gamma\bullet\delta, q, B, q', i, j, r, s]$ where $A \to \beta\gamma\delta$ is a production, q, q' are stack symbols, $0 \le i \le r < s \le j \le |w|$ and q, B, q', r, s are $-$ if a buHC-LIG tree has no dependent stack descendant (then $0 \le i < j \le |w|$). The item homomorphism $\varphi_{\text{buHC-LIG}}$ maps a tuple of positions (i, j, r, s) to $(a_{i+1} \dots a_r, a_{r+1} \dots a_s, a_{s+1} \dots a_j)$, resp. to $(-, a_{i+1} \dots a_j, -)$ if $r = s = -$, where $w = a_1 \dots a_n$ is the input string. The deduction steps are shown in Fig. 6.

The transformation defined in Theorem 4 may also be used to account for the form of the steps in the CYK-LIG algorithm in [9, 10]. This algorithm may be seen as the result of a transformation of a CYK tree algebra for CFG in Chomsky normal form using a similar substitution of subtrees as in Fig. 4.

6 Conclusion

We have proposed an algebraic method for the construction of tabular parsing algorithms. A parsing algebra for a grammar G and input string w is a relative subalgebra of a quotient algebra of the direct product of a tree algebra $\mathcal{A}$ (that reflects the parsing strategy) and a yield algebra $\mathcal{B}$ (that describes how the input string is processed) which is homomorphic to $\mathcal{A}$. A parsing schema is the inverse image of a parsing algebra under a strong homomorphism. Correctness of a parsing schema is defined at the level of tree operations. We have demonstrated the construction using a buHC parsing strategy for CFG. Furthermore, we have derived a buHC parsing schema for LIG from the buHC parsing schema for CFG by means of a correctness preserving algebraic transformation.

We have proposed the algebraic construction of tabular parsing algorithms for LIG as an alternative to the automaton-based approach proposed in recent papers [1, 5] because it allows to derive LIG algorithms from CFG algorithms by means of algebraic transformations, allows simpler and more elegant correctness proofs by using general theorems, and is not restricted to left-right parsing strategies. Furthermore, it makes the notion of parse items more precise and thus adds to a better understanding of parsing schemata.

References

[1] Miguel A. Alonso Pardo, Eric de la Clergerie, and David Cabrero Souto. Tabulation of automata for tree adjoining languages. In *Proc. of the Sixth Meeting on Mathematics of Language (MOL 6)*, pages 127–141, Orlando, Florida, USA, July 1999.

[2] Gerald Gazdar. Applicability of indexed grammars to natural languages. Tech. Rep. CSLI-85-34, Center for Study of Language and Information, Stanford, 1985.

[3] George Grätzer. *Universal Algebra.* Springer Verlag, New York, second edition, 1979.

[4] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3*, chapter 2, pages 69–123. Springer, Berlin, 1997.

[5] Mark-Jan Nederhof. Models of tabulation for TAG parsing. In *Sixth Meeting on Mathematics of Language*, pages 143–158, Orlando, Florida USA, July 1999. University of Central Florida.

[6] Karl-Michael Schneider. *Algebraic Construction of Parsing Schemata.* Doctoral dissertation, University of Passau, 1999.

[7] Klaas Sikkel. *Parsing Schemata.* Proefschrift, Universiteit Twente, CIP-Gegevens Koninklijke Bibliotheek, Den Haag, 1993.

[8] Klaas Sikkel. Parsing schemata and correctness of parsing algorithms. *Theoretical Computer Science*, 199(1–2):87–103, 1998.

[9] K. Vijay-Shanker and David J. Weir. Polynomial parsing of extensions of context-free grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology*, pages 191–206. Kluwer, Dordrecht, 1991.

[10] K. Vijay-Shanker and David J. Weir. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19(4):591–636, December 1993.

A SPANISH POS TAGGER WITH VARIABLE MEMORY

Triviño-Rodriguez, J.L. **Morales-Bueno, R.**
Dept. Lenguajes y Ciencias de la Computación
University of Málaga
{trivino,morales}@lcc.uma.es

Abstract
An implementation of a Spanish POS tagger is described in this paper. This implementation combines three basic approaches: a single word tagger based on decision trees, a POS tagger based on variable memory Markov models, and a feature structures set of tags. Using decision trees for single word tagging allows the tagger to work without a lexicon that lists only possible tags. Moreover, it decreases the error rate because there are no unknown words. The feature structure set of tags is advantageous when the available training corpus is small and the tag set large, which can be the case with morphologically rich languages like Spanish. Finally, variable memory Markov models training is more efficient than traditional full-order Markov models and achieves better accuracy. In this implementation, 98.58% of tokens are correctly classified.

1 Introduction

Many words in Spanish function as different parts of speech (POS). Part-of-speech tagging is the problem of determining the syntactic part of speech of an occurrence of a word in context. Because most of the high-frequency Spanish words function as several parts of speech, an automatic system for POS tagging is very important for most other high level natural language text processing.

There are many approaches to the performance of this task, but they could be classified into two main ones:

- The linguistic approach, in which the language model is written by a linguist, generally in the form of rules or constraints. An example of this approach is the tagger developed by Voutilainen [Voutilainen and Järvinen, 1995].

- The automatic approach, in which the language model is automatically obtained from corpora. This corpora can be either raw or annotated. In this way, the automatic taggers are classified into supervised and unsupervised learning:
  - Supervised learning is applied when the model is obtained from annotated corpora.
  - Unsupervised learning is applied when the model is obtained from raw corpora training.

The most noticeable examples of automatic tagging approaches are Markov chains [Church, 1989, Charniak et al., 1993, Jelinek, 1985, Merialdo, 1994, Garside et al., 1987, Cutting et al., 1992], neural networks [Schmid, 1994], decision trees [Daelemans et al., 1996, Márquez and Rodríguez, 1995], and transformation-based error driven learning [Brill, 1994].

The most widely-used methods for POS tagging are stochastic methods based on fixed order Markov chains models and Hidden Markov models. However, the complexity of Markov chains grows exponentially with its order L, and hence only low order Markov chains could be considered in practical applications. Moreover, it has been observed that, in many natural sequences, memory length depends on the context and is not fixed. In order to solve this problem, in 1996 Dana Ron [Ron, 1996, Ron et al., 1996] described a learning model of a subclass of PFAs (*Probabilistic Finite Automatas*) called PSAs (*Probabilistic Suffix Automatas*). A PSA is like a Markov chain with variable memory length. So, this model avoids the exponential complexity relation with the order L and thus can be applied to better effect than Markov chains.

In this paper a Spanish POS tagger based on variable memory Markov chains is described (which will be called *VMM* throughout). This tagger is based on a modified version of Singer's tagger [Schütze and Singer, 1994]. The VMM training algorithm used by Singer's tagger is more efficient than the Markov chains training algorithm. Furthermore, it allows higher order chains than fixed order Markov models. This implies better accuracy and a lower error rate.

On the other hand, the described tagger makes use of a feature structure set of tags like the model described by Kempe [Kempe, 1994]. Usually, the contextual probability of a tag is estimated as the relative frequency in a given training corpus. With a large tag set (resulting from the fact that the tags contain – besides the POS – a lot of morphological information) and with only a small training corpus available, most of these probabilities are too low for an exact estimation of contextual probabilities. Moreover, a large tag set increases the complexity of the Markov chains and it implies low training efficiency. An example of this problem is the tagger developed by Sánchez [Sánchez León and Nieto Serrano, 1995].

Finally, most taggers deal with the problem of determining the syntactic part of speech of an occurrence of a word in context, but they can not determine the set of correct tags for a word without any such context. Instead, most taggers use a lexicon that lists possible tags. In morphologically rich languages, this lexicon must be very large in order to represent the language correctly. In order to solve this problem, a single word tagger has been added to the system described. This single word tagger [Triviño, 1995] is based on a modified version of the ID3 algorithm described by Quinlan [Quinlan, 1986].

This paper is organised as follows: first the single word tagger is described. Next, the VMM is shown and how it is applied to POS tagging. Finally, the results obtained from this model are analysed.

2 Single word tagging

The single word tagger is the algorithm that computes all valid tags for a word if the word is considered out of context. The information used to carry out this task is the morphology of the word. Therefore, this kind of algorithm must be applied to morphologically rich languages where a word may have many derived words and where the lexicon must be very large in order to hold all these words.

Most single word taggers like Padro's tagger [Padró, 1996] are based on the two-level morphology described by Kimmo Koskenniemi [Koskenniemi, 1985]. Koskenniemi's model of two-level morphology was based on the traditional distinction that linguists make between morphotactics, which enumerates the inventory of morphemes and specifies in what order they can occur, and morphophonemics, which accounts for alternate forms or "spellings" of morphemes according to the phonological context in which they occur. For example, the word *chased* is analyzed morphotactically as the stem *chase* followed by the suffix *-ed*. However, the addition of the suffix *-ed* apparently causes the loss of the final *e* of *chase*; thus *chase* and *chas* are allomorphs or alternate forms of the same morpheme. Koskenniemi's model is "two-level" in the sense that a word is represented as a direct letter-for-letter correspondence between its lexical or underlying form and its surface form.

Instead of this, the single word tagger that we have developed is based on a modified version of the ID3 algorithm. This algorithm generates a decision tree for a single out attribute (called *class* in TDIDT terminology). However, the feature structure set of tags of the tagger needs a decision tree for several out attributes. In order to solve this problem the ID3 algorithm has been modified. So, the modified version of the algorithm builds up the tree recursively as follows: first, the algorithm chooses the best out attribute in order to generate the tree; next, the tree is built up recursively following the traditional ID3 algorithm until the selected out attribute is classified; finally, the algorithm goes back and chooses another out attribute for each leaf to build up the tree from that leaf.

The single word tagger has been trained with an 87830 words corpus. From this corpus, the learning algorithm has computed a decision tree with 48317 rules. Next, an example of a generated rule is shown (the symbol '*' means any symbol):

```
**********************************b***dad
  [NomN_ComFemSg#1]
```

A word matches in this rule if the word ends with the symbol 'b' followed by any three symbols followed by the string 'dad'. This rule has been computed from the words: ***accesibilidad, aceptabilidad, adaptabilidad, admisibilidad, afabilidad, agregabilidad, amabilidad, amigabilidad, antiobesidad, aplicabilidad, asociabilidad, autoestabilidad, barbaridad, combustibilidad, comensurabilidad, compatibilidad, comunicabilidad, conductibilidad, confortabilidad, conmutabilidad, contabilidad, contrastabilidad, convertibilidad, corresponsabilidad, corruptibilidad, credibilidad, cuestionabilidad, culpabilidad, debilidad, deseabilidad, disponibilidad, elegibilidad, estabilidad, excitabilidad, factibilidad, falibilidad, fiabilidad, flexibilidad, flotabilidad, globalidad, gobernabilidad, habilidad, habitabilidad, heredabilidad, honorabilidad, impasibilidad, impenetrabilidad, impermeabilidad, imposibilidad, imprevisibilidad, improbabilidad, imputabilidad, inaccesibilidad, incompatibilidad, incomunicabilidad, incorruptibilidad, inestabilidad, infalibilidad, ingobernabilidad, ininteligibilidad, insaciabilidad, insensibilidad, inteligibilidad, inviabilidad, invisibilidad, irresponsabilidad, irreversibilidad, irritabilidad, maniobrabilidad, morbilidad, morbosidad, mutabilidad, nubosidad, obesidad, obviedad, perdurabilidad, permeabilidad, posibilidad, probabilidad, rentabilidad, respetabilidad, responsabilidad, reversibilidad, sensibilidad, sobriedad, sociabilidad, solubilidad, subunidad, susceptibilidad, urbanidad, verbosidad, viabilidad, volubilidad, vulnerabilidad.***

3 Training algorithm

3.1 Variable memory Markov models (VMM)

In this section, the variable memory Markov model is described. This model is called PSA (*Probabilistic Suffix Automata*) and has been described by Ron [Ron, 1996]. This model is hard to learn. Another model exists: the PST model (*Prediction Suffix Trees*) that can be learned more easily. It can be shown that every distribution generated by a PSA can be equivalently generated by a PST which is not much larger. So, the PST model is used as learning hypothesis instead of the PSA model.

Definition 3.1 (PST (*Prediction Suffix Tree*)) *A PST T over an alphabet Σ is a tree of degree $|\Sigma|$. Each edge in the tree is labeled by a single symbol in Σ such that from every internal node there*

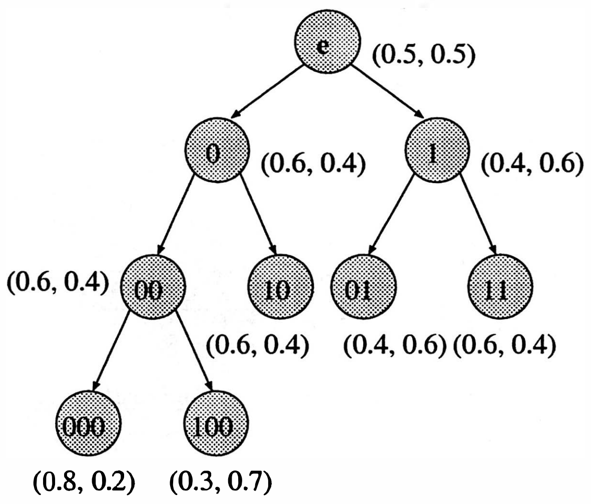


Figure 1: Example of PST over the alphabet $\{0, 1\}$. The prediction probabilities of the symbols '0' and '1' are next to the nodes, in parentheses, respectively.

is exactly $|\Sigma|$ edges each one labeled with a different symbol. The nodes of the tree are labeled by pairs (s, γ_s) where s is the string associated with the walk starting from that node and ending in the root of the tree, and $\gamma_s : \Sigma \to [0,1]$ is the next symbol probability function related with s. It is necessary that for every string s labeling a node in the tree, $\sum_{\sigma \in \Sigma} \gamma_s(\sigma) = 1$. An example of PST can be seen in figure 1.

An interesting feature of the PST model is that the transition probabilities are smoothed along the learning task. This avoids transition probabilities with value 0. For this reason, the model must not be interpolated after the learning task unlike the Markov model.

The PST learning algorithm generates a PST hypothesis that holds the ϵ-good hypothesis property with respect to a given PSA.

Definition 3.2 (ϵ-good hypothesis) *Let M be a PSA and let T be a PST and let P_M and P_T be the two probability distributions they generate respectively. We say that T is an ϵ-good hypothesis with respect to M if $\forall N > 0$, it is held that:*

$$\frac{1}{N} * \underset{KL}{D}[P_M^N || P_T^N] \leq \epsilon$$

Where

$$\underset{KL}{D}[P_M^N || P_T^N] \overset{def}{=} \sum_{r \in \Sigma^N} \overset{N}{\underset{M}{P}}(r) * \log \frac{P_M^N(r)}{P_T^N(r)}$$

Definition 3.3 (Parameters of the learning model)

- *L is the maximum length of the string labeling the states.*
- *n is an upper bound on the number of states in M.*
- *δ is the confidence parameter, $0 < \delta < 1$.*
- *ϵ is the approximation parameter, $0 < \epsilon < 1$*
- *The training sample is compounded by m' strings generated by M each of length $L + 1$ at least.*

Definition 3.4 (Observation probabilities) *The probability of a state s and a symbol σ in the learning sample is calculated as the relative frequency of the state s (number of times the state s*

appears in the sample divided by the length of the sample) and the relative frequency of the symbol σ in the context of the state s (number of times the symbol σ appears in the sample after the state s divided by the number of times the state s appears in the sample). This is computed by the following equations:

$$\begin{aligned}
\widetilde{\Pr}(s) &= \frac{1}{m'(\ell - L)} \sum_{i=1}^{m'} \sum_{j=L}^{\ell-1} \chi_j^i(s) \\
\widetilde{\Pr}(\sigma|s) &= \frac{\sum_{i=1}^{m'} \sum_{j=L}^{\ell-1} \chi_{j+1}^i(s\sigma)}{\sum_{i=1}^{m'} \sum_{j=L}^{\ell-1} \chi_j^i(s)}
\end{aligned}$$

Where m' is the number of strings in the sample and ℓ is the length of the strings, such that $\ell \geq L+1$ and $\chi_j^i(s)$ is defined as 1 if $r_{j-|s|+1}^i \cdots r_j^i = s$ and 0 otherwise.

Definition 3.5 (Internal parameters) *The following parameters are used internally by the learning algorithm:*

$$\begin{aligned}
\epsilon_2 &= \frac{\epsilon}{48 * L} \\
\gamma_{min} &= \frac{\epsilon_2}{|\Sigma|} \\
\epsilon_0 &= \frac{\epsilon}{2 * n * L * \log(\frac{1}{\gamma_{min}})} \\
\epsilon_1 &= \frac{\epsilon_2 * \gamma_{min}}{8 * n * \epsilon_0}
\end{aligned}$$

The learning algorithm generates a PST from a training sample following a top-down approach. It starts with a tree consisting of a single root node labeled with the empty string. Progressively, new nodes and edges are added to the tree. Each node is added depending on its observation probability. In this way, a new node is added when it has a significant probability of being in the sample and the transition probabilities from this node differ substantially from the transition probability of its parent. An example of the learning task can be seen in figure 2.

3.2 Tagging with VMM (Variable Memory Markov Models)

Part-of-speech tagging is the problem of determining the syntactic part of speech of an occurrence of a word in context. Therefore, given a sequence of words $w_1, \ldots, w_n$, POS tagging is the problem of determining the tag sequence $t_1, \ldots, t_n$ that is most likely for $w_1, \ldots, w_n$. This could be computed by maximizing the joint probability of $w_1, \ldots, w_n$ and $t_1, \ldots, t_n$:

$$\begin{aligned}
\tau(w_1, \ldots, w_n) &= \underset{t_1, \ldots, t_n}{argmax}\, Pr(t_1, \ldots, t_n | w_1, \ldots, w_n) \\
&= \underset{t_1, \ldots, t_n}{argmax} \frac{Pr(t_1, \ldots, t_n, w_1, \ldots, w_n)}{Pr(w_1, \ldots, w_n)} \\
&= \underset{t_1, \ldots, t_n}{argmax}\, Pr(t_1, \ldots, t_n, w_1, \ldots, w_n) \qquad (1)
\end{aligned}$$

The equation 1 could be laid out as follows:

$$\begin{aligned}
&Pr(t_1, \ldots, t_n, w_1, \ldots, w_n) = \\
&\prod_{i=1}^{n} Pr(t_i | t_1, \ldots, t_{i-1}) * Pr(w_i | t_1, \ldots, t_i, w_1, \ldots, w_{i-1}) \qquad (2)
\end{aligned}$$

Algorithm 3.1 (Learning a PST)

```
1. Let $\bar{T}$ be a single root node (labeled with e) and let
   $\bar{S} = \{\sigma \mid \sigma \in \Sigma \wedge \widetilde{\Pr}(\sigma) \geq (1-\epsilon_1) * \epsilon_0\}$.
2. While $\bar{S} \neq \emptyset$, pick up any $s \in \bar{S}$ and do:
   (a) Remove s from $\bar{S}$
   (b) If there exists $\sigma \in \Sigma$ such that
       $$\widetilde{\Pr}(\sigma|s) \geq (1+\epsilon_2) * \gamma_{min} \quad \wedge \quad \frac{\widetilde{\Pr}(\sigma|s)}{\widetilde{\Pr}(\sigma|\,suffix(s))} > 1 + 3 * \epsilon_2$$
       then add to $\bar{T}$ the node s and all the nodes on the path from the deepest node in $\bar{T}$ that is a suffix of s, to $\bar{S}$.
   (c) If $|s| < L$ then, for every $\sigma' \in \Sigma$, if $\widetilde{\Pr}(\sigma' \cdot s) \geq (1-\epsilon_1) * \epsilon_0$ then add $\sigma' \cdot s$ to $\bar{S}$.
3. Let $\hat{T} = \bar{T}$
4. Extend $\hat{T}$ by adding all missing sons of internal nodes.
5. For each s labeling a node in $\hat{T}$ let:
   $$\hat{\gamma}_s(\sigma) = \widetilde{\Pr}(\sigma|s') * (1 - |\Sigma| * \gamma_{min}) + \gamma_{min}$$
   where $s'$ is the longest suffix of s in $\bar{T}$
```

Up to this point no assumptions about the probabilities have been made, but the probabilities required by equation 2 are not empirically collectible. In order to solve this problem, it is necessary to make the following assumptions about these probabilities [1]:

$$Pr(t_i|t_1,\ldots,t_{i-1}) = Pr(t_i|t_{i-L},\ldots,t_{i-1}) \tag{3}$$
$$Pr(w_i|t_1,\ldots,t_i,w_1,\ldots,w_{i-1}) = Pr(w_i|t_i) \tag{4}$$

By equations 3 and 4, equation 2 can be expressed as follows:

$$Pr(t_1,\ldots,t_n,w_1,\ldots,w_n) = \prod_{i=1}^{n} Pr(t_i|t_{i-L},\ldots,t_{i-1}) * Pr(w_i|t_i) \tag{5}$$

Where L is the maximum length of the model.

This tagger is based on a feature structure set of tags. Therefore, the term t_i in equation 5 is a vector of a symbols[2] (a symbol for each attribute). However, the Viterbi algorithm [Viterbi, 1967] can only compute the probability of a sequence of a single attribute. In order to solve this problem, a set of a trees (a tree for each attribute) has been computed independently. The equation 5 is computed from this set of trees as follows:

$$Pr(t_1,\ldots,t_n,w_1,\ldots,w_n) =$$

[1] These assumptions were described by Charniak [Charniak et al., 1993]. That is, that the probability of the tag given the past depends only on the last two tags, and the probability of the word given the past depends only on its tag.

[2] We take into account 18 attributes for each word, thus $a = 18$ in our tagger

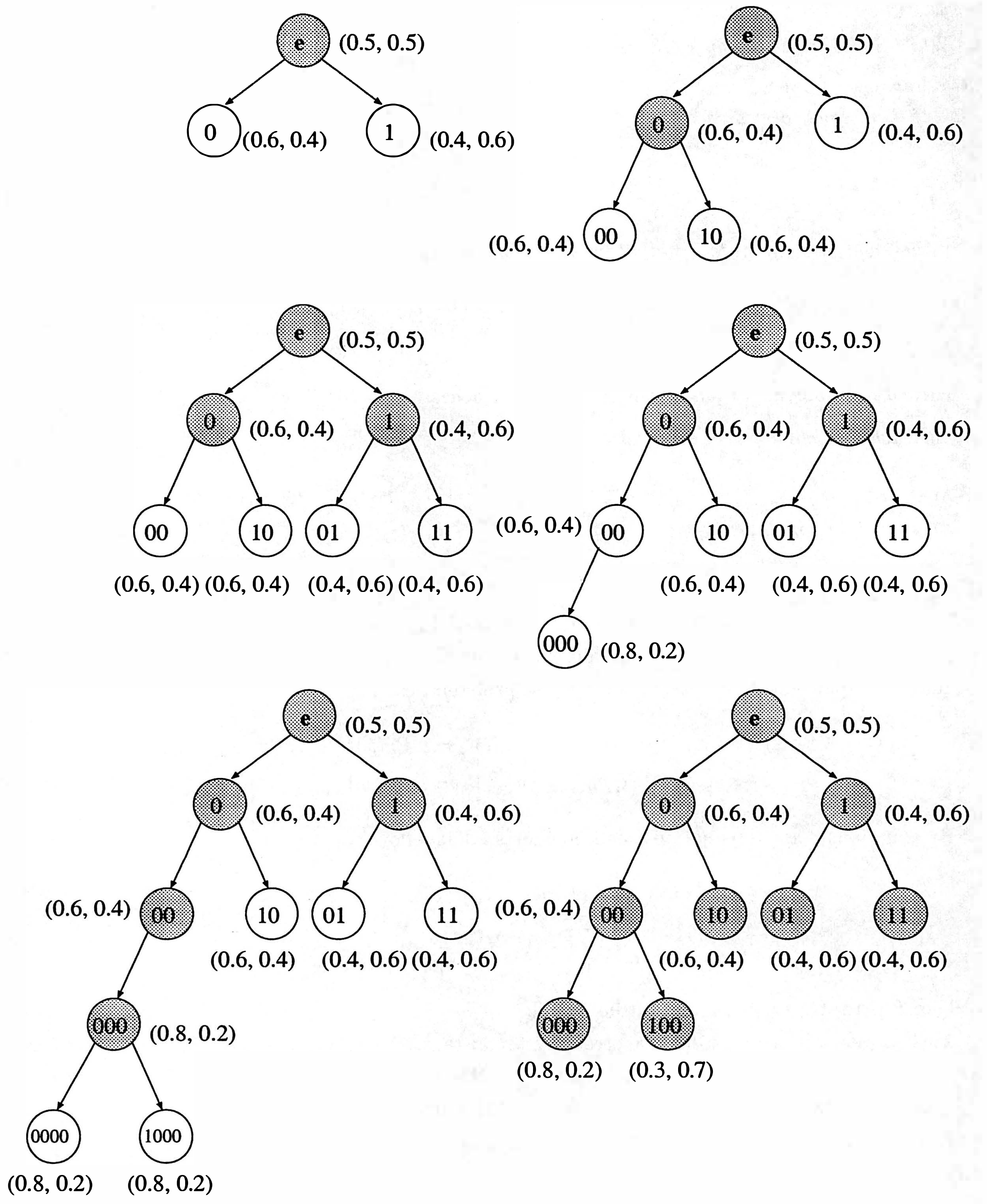


Figure 2: An illustrative run of the learning algorithm

$$= \prod_{j=1}^{a} Pr(t_1^j, \ldots, t_n^j, w_1, \ldots, w_n)$$

$$= \prod_{i=1}^{n} (Pr(w_i|t_i) * \prod_{j=1}^{a} Pr(t_i^j | t_{i-L}^j, \ldots, t_{i-1}^j)) \tag{6}$$

Given a set of PSTs T, $Pr(t_i^j | t_{i-L}^j, \ldots, t_{i-1}^j)$ is estimated by $\gamma_s^j(t_i^j)$ where s is the longest suffix of $t_{i-L}^j, \ldots, t_{i-1}^j$ labeling a state in T^j and γ_s^j is the next symbol probability function related with the state s in T^j.

The static probability $Pr(w_i|t_i)$ is estimated indirectly from $Pr(t_i|w_i)$ using Bayes' Theorem:

$$Pr(w_i|t_i) = \frac{Pr(w_i) * Pr(t_i|w_i)}{Pr(t_i)}$$

The terms $Pr(w_i)$ are constant for a given sequence w_i and can therefore be omitted from the maximization. The values of $Pr(t_i)$ can be computed as follows:

$$Pr(t_i) = \prod_{j=1}^{a} \gamma_e^j(t_i^j) \tag{7}$$

The static parameters $Pr(t_i|w_i)$ are computed by the single word tagging.

4 Analysis of results

The order L of the VMM tagger has been set to 4. The tagger has been trained on a Spanish 45000 words corpus tagged with the set of tags shown in table 1. An example of tagged text [3] can be seen in figure 3.

The accuracy obtained is 98.58%. This accuracy is better than that achieved by any other tagger we know of. In table 2 a comparison between the accuracy of several taggers is shown.

Our results (98.58%) are better than Singer's tagger (95.81%) based on VMM because the single word tagger has added lexical information to our tagger. This can decrease the error rate when errors due to bad tags for rare words are avoided by the single word tagger. However, it is difficult to compare these results to other works, since the accuracy varies greatly depending on the corpus, tag set, etc. More exact performance could be measured by increasing the size of training a testing text. Thus, the performance could decrease or increase if different corpus styles are used.

5 Conclusions and future work

In this paper, a high accuracy Spanish tagger has been presented. This tagger has three main features: a single word tagger based on decision trees, a feature structure set of tags, and a variable memory Markov model.

The results obtained show that joining an accurate single word tagger with a feature structure set of tags and a high order VMM produces an improvement in the performance of the tagger.

[3]Many spanish verbs have the same form to several times. In example, the verb *'tomar'* have the same word (*'toma'*) for the third person in present time and second person in imperative time. The rest of persons in these times are different. In order to determine the valid tag, it must be necessary a syntactic analysis. Therefore, the POS tagger keeps two different tags to this word. In order to measure the accuracy of the tagger, this word is considered to be correctly tagged only if it has its two tags.

Word	Valid tags	Selected tag
Él	[ProP_PerMasP3Sg#1]	[ProP_PerMasP3Sg#1]
toma	[NomN_ComFemSg#0.333333] [VerImvImpSim2Sg#0.333333] [VerIndPrs3Sg#0.333333]	[VerImvImpSim2Sg#0.5] [VerIndPrs3Sg#0.5]
el	[ArtAr_DetMasSg#1]	[ArtAr_DetMasSg#1]
tomo	[VerIndPrs1Sg#0.5] [NomN_ComMasSg#0.5]	[NomN_ComMasSg#1]
del	[Pre#1]	[Pre#1]
libro	[VerIndPrs1Sg#0.5] [NomN_ComMasSg#0.5]	[NomN_ComMasSg#1]
bajo	[Pre#0.2] [AdvAv_Modo#0.2] [NomN_ComMasSg#0.2] [AdjAd_CalMasSg#0.2] [VerActIndPrs1Sg#0.2]	[Pre#1]
la	[ArtAr_DetFemSg#0.333333] [ProP_PerFemP3Sg#0.333333] [NomN_ComMasSg#0.333333]	[ArtAr_DetFemSg#1]
mesa	[NomN_ComFemSg#0.333333] [VerImvImpSim2Sg#0.333333] [VerIndPrs3Sg#0.333333]	[NomN_ComFemSg#1]
del	[Pre#1]	[Pre#1]
hombre	[NomN_ComMasSg#1]	[NomN_ComMasSg#1]
bajo	[Pre#0.2] [AdvAv_Modo#0.2] [NomN_ComMasSg#0.2] [AdjAd_CalMasSg#0.2] [VerActIndPrs1Sg#0.2]	[AdjAd_CalMasSg#1]
.	[SimbSPunto]	[SimbSPunto]

Figure 3: An example of the output of the tagger

Attribute	# Tags
Class	12
Class of noun	4
Class of adjective	13
Class of adverb	9
Class of pronoun	13
Function of pronoun	4
Class of conjunction	13
Class of Article	3
Gender	5
Active/Passive	3
Form	6
Tense	19
Person	4
Number	4
Prefix	2
Suffix	2
Class of Symbol	22

Table 1: Set of tags in the corpus

Tagger	**Language**	**Corpus (# words)**	**Accuracy (%)**
Triviño	Spanish	45000	98.58
Padró	Spanish/english	10^6	97.45
Charniak	English	10^6	96.45
Kempe	French	$2 * 10^6$	96.16
Brill	English	350000	96.0
Singer	English	10^6	95.81
Kempe	French	10000	88.89

Table 2: Comparison between several taggers

On the other hand, most of the errors arise because the tagger does not take into account relations between out attributes. Thus, the main development line of this work is to change the variable memory model for a model that takes relations between attributes into account.

References

[Brill, 1994] Brill, E. (1994). Some advances in transformation-based part of speech tagging. In *Proceedings of AAAI94*, page 6.

[Charniak et al., 1993] Charniak, E., Hendrickson, C., Jacobson, N., and Perkowitz, M. (1993). Equations for part-ofspeech tagging. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, pages 784–789.

[Church, 1989] Church, K. (1989). A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of ICASSP.*

[Cutting et al., 1992] Cutting, D., Kupiec, J., Pederson, J., and Sibun, P. (1992). A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing.* ACL.

[Daelemans et al., 1996] Daelemans, W., Zavrel, J., Beck, P., and S.Gillis (1996). Mtb: A memory-based part-of-speech tagger generator. In *Proceedings of 4th Workshop on Very Large Corpora*, Copenhagen, Denmark.

[Garside et al., 1987] Garside, R., Leech, G., and Sampson, G. (1987). *The Computational Analysis of English.* London and New York: Longman.

[Jelinek, 1985] Jelinek, F. (1985). Robust part-of-speech tagging using a hidden markov model. Technical report, IBM.

[Kempe, 1994] Kempe, A. (1994). Probabilistic tagging with feature structures. In *Coling-94*, volume 1, pages 161–165.

[Koskenniemi, 1985] Koskenniemi, K. (1985). A general two-level computational model for word-form recognition and production. In Departament of Linguistics, editor, *Karlsson, F.*, pages 1–18. Computational Morphosyntax, University of Helsinki.

[Merialdo, 1994] Merialdo, B. (1994). Tagging english text with a probabilistic model. *Computational Linguistics*, 2(20):155–171.

[Márquez and Rodríguez, 1995] Márquez, L. and Rodríguez, H. (1995). Towards learning a constraint grammar from annotated corpora using decision trees. ESPRIT BRA-7315 Acquilez II, Working Paper.

[Padró, 1996] Padró, L. (1996). Pos tagging using relaxation labelling. In *Proceedings of 16th International Conference on Computational Linguistics*, Copenhagen, Denmark.

[Quinlan, 1986] Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, (1):81–106.

[Ron, 1996] Ron, D. (1996). *Automata Learning and its Applications.* PhD thesis, MIT.

[Ron et al., 1996] Ron, D., Singer, Y., and Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, page 34.

[Schmid, 1994] Schmid, H. (1994). Part-of-speech tagging with neural networks. In *Proceedings of 15th International Conference on Computational Linguistics*, Kyoto, Japan.

[Schütze and Singer, 1994] Schütze, H. and Singer, Y. (1994). Part-of-speech tagging using a variable memory Markov model. In *ACL 32'nd.*

[Sánchez León and Nieto Serrano, 1995] Sánchez León, F. and Nieto Serrano, A. (1995). Development of a spanish version of the Xerox Tagger. Technical report, Facultad de Filosofía y Letras (Universidad Autónoma de Madrid).

[Triviño, 1995] Triviño, J. (1995). SEAM. Sistema experto para análisis morfológico. Master's thesis, Universidad de Málaga.

[Viterbi, 1967] Viterbi, A. (1967). error bounds for convolutional codes and an asymptotical optimal decoding algorithm. In *Proceedings of IEEE*, volume 61, pages 268–278.

[Voutilainen and Järvinen, 1995] Voutilainen, A. and Järvinen, T. (1995). Specifying a shallow grammatical representation for parsing purposes. In *Proceedings of the 7th meeting of the European Association for Computational Linguistics*, pages 210–214.

PARSING A LATTICE WITH MULTIPLE GRAMMARS

Fuliang Weng[1]* **Helen Meng**[2]
Po Chui Luk[2]
[1]Speech Technology and Research Laboratory
SRI international
333 Ravenswood Ave, Menlo Park, CA 94025
fuliang@speech.sri.com
[2]Human-Computer Communications Laboratory
Department of Systems Engineering and Engineering Management
the Chinese University of Hong Kong
Shatin, NT, Hong Kong, China
{hmmeng,pcluk}@se.cuhk.edu.hk

Abstract

Efficiency, memory, ambiguity, robustness and scalability are the central issues in natural language parsing. Because of the complexity of natural language, different parsers may be suited only to certain subgrammars. In addition, grammar maintenance and updating may have adverse effects on tuned parsers. Motivated by these concerns, [25] proposed a grammar partitioning and top-down parser composition mechanism for loosely restricted Context-Free Grammars (CFGs). In this paper, we report on significant progress, i.e., (1) developing guidelines for the grammar partition through a set of heuristics, (2) devising a new mix-strategy composition algorithms for any rule-based grammar partition in a lattice framework, and 3) initial but encouraging parsing results for Chinese and English queries from an Air Travel Information System (ATIS) corpus.

1 Introduction

Efficiency, memory, ambiguity, robustness, and scalability are the central issues in natural language parsing [5, 21, 7, 19, 12, 24, 14]. Among others, Earley's algorithm is often used for its superior performance due to a good combination of top-down prediction and bottom-up bookkeeping. Schabes' algorithm precompiles all the prediction steps and some completion steps in Earley's algorithm, and has resulted in improved average performance [19] while maintaining the cubic time complexity and square space complexity of Earley's algorithm. Tomita [21] generalized Knuth's LR(k) parsers to allow nondeterminism, and reported faster performance than Earley's parser. However, in the worst case, as different researchers [23, 7] have pointed out, (G)LR parsers have exponential time and space complexity. By utilizing a lookup table for *reduce* steps in a GLR parser, Kipps [9] was able to limit the time complexity to $O(n^3)$, but this approach led to a less practical system. Nederhof and Satta [14] further improved the GLR parser through splitting a *reduce* step into more primitive operations that could remove

*The author is currently with Intel China Research Center, No.1 GuangHua Road, Beijing 100020, China. His new e-mail address is fuliang.weng@intel.com.

redundancy in parsing. To deal with real-world natural language parsing, others [18, 12, 24] proposed ways to handle extra-grammatical cases for GLR parsers.

As reported in this paper, we have continued our work in grammar partition and parser composition [25] that explores advantages of different parsing algorithms. We have also investigated guidelines for grammar partitioning, the relationship between partitioning and state space selection, and practical ways for composing different parsing algorithms, in the context of robust parsing of Chinese and English queries from an ATIS corpus.

In Section 2, we briefly review some concepts of grammar partitioning, and discuss a few guidelines for the task and their relationship with state space selection for the parsers mentioned above. Section 3 presents an improved parser composition algorithm. Section 4 describes some initial but encouraging results on parsing Chinese and English queries. The final section presents our conclusion and describes our future research directions.

2 Grammar Partitioning

To facilitate our discussion, we first briefly review the motivation and concept for grammar partition [25]. Then, we give some intuitive guidelines for the partitioning and their relationship with state space selection for the parsers analyzed by [13].

2.1 Motivation and Concept

Earley [4] and Ukkonen [23] proved that the following artificial grammar G_n is exponential for LR(k) parsers, and even for any right parsers, respectively:

$S \rightarrow A_i \quad (1 \leq i \leq n)$
$A_i \rightarrow a_j A_i \quad (1 \leq i \neq j \leq n)$
$A_i \rightarrow a_i B_i \mid b_i \quad (1 \leq i \leq n)$
$B_i \rightarrow a_j B_i \mid b_i \quad (1 \leq i \leq n)$

These results present a potentially severe problem for LR(k) parsers and their applications to natural language processing. However, as illustrated in [25], if we partition the grammar into n subgrammars, parse each subgrammar separately, and combine subparsers for the subgrammars, we can obtain a satisfactory compromise in terms of speed and memory.

From a practical point of view, in addition to the possibility of causing an exponential number of compiled states, sublanguages in a large parser may be described by pre-existing subgrammars using specialized parsing algorithms. Implementation and maintenance constraints may prevent integration of the sublanguage modules into a monolithic system. A framework for partitioning grammars and composing parsers of different types would be useful for the scalability of a parsing system.

In the rest of this paper, we use the definitions in [25] for grammar partition. Intuitively, we partition the production set into subsets, and create subgrammars for the subsets, accordingly. The interaction among different subgrammars is through nonterminal sets, i.e., INPUT and OUTPUT, and a virtual terminal technique. The INPUT to a subgrammar is a set of nonterminals that were previously parsed by other subgrammars. The OUTPUT of a subgrammar is those nonterminals that were parsed based on this subgrammar and used by other subgrammars as their INPUT symbols. For every nonterminal A in the INPUT set of a subgrammar, there is an augmented rule $A \longrightarrow vt_A$ for that grammar, where vt_A is called a *virtual terminal* acting as if it were a terminal [10, 24, 25]. A directed calling graph for the subgrammar set of G is defined as (V, E), where V is the subgrammar set and $E = \{(A, B)\}$, with the overlap of the OUTPUT of B and the INPUT of A being nonempty. It can be proved that the partitioned grammar will lead to the same recognizable language. It is straightforward to see that this grammar partition definition is more general than the one defined with respect to nonterminal symbol set [10].

2.2 Guidelines for Grammar Partitioning

Despite the fact that grammar partitioning is desirable, it is not clear how it should be done in any general case. We describe properties that may help in understanding this subject.

First, let us look at the two extreme cases: the coarsest and finest partitions. The coarsest partition is the grammar itself, and all the production rules are included in the set. The finest grammar partitioning is the set of all singletons with one exact grammar rule in each set. The INPUT/OUTPUT set members are the nonterminal symbols on the right-hand side/left-hand side (RHS/LHS) of the rules in these singletons. So, an inverse problem of grammar partitioning is the clustering of smaller subgrammars into bigger subgrammars based on calling relations among different subgrammars.

Intuitively, we may want to create clusters that have frequent interaction within cluster members, but fewer interactions between members of different clusters. This will reduce the possibility of combining common prefix computations for highly ambiguous grammars. However, it may increase the chance of computing a common prefix repetitively. Our criterion is to form as big a cluster as possible within an affordable size limit.

First we create a weighted graph describing the connections among different clusters. Its node set consists of all the clusters in the grammar. When no training data is available, its edge set and weights are obtained through the following procedure. For each pair of clusters, intersect one cluster's INPUT and the other cluster's OUTPUT, take the total size of the two symmetric intersections as the weight on the connecting edge, and create an edge between the two clusters only when the weight is above a certain threshold.

When training data is available, probability weights on the edges can be computed based on the actual calls between the corresponding clusters, in a way, related to the grammar chunking approach [16].

For better cluster quality, we present a set of simple heuristics for the refinement of grammar clusters. There are two types of refinements, merge to form bigger clusters, and split to form smaller clusters. The merge operations are:

1. Compute the transitive closures of the calling relations for clusters, and replace the original clusters with their closures.
2. If a cluster has an empty INPUT set, duplicate the cluster for each of its parent clusters, and merge the copies with its parents.

The split operations can be through removing the edges with low weights or the ones that lead to minimum changes with respect to its original graph using *Kullback-Leibler* distance.

If dotted rules are considered, the state space of the viable prefix recognizer NFA [1, 3] (VP-NFA) is one extreme partition with each dotted rule as a state (or, a partitioned subgrammar). In the VP-NFA, for each pair of states $A \rightarrow \alpha.X\beta$ and $A \rightarrow \alpha X.\beta$, there is a transition with X as its label, and for each pair of states $A \rightarrow \alpha.B\beta$ and $B \rightarrow .\gamma$, there is a ϵ-transition. If we collapse the end states of ϵ transitions in the VP-NFA with their corresponding start states (if multiple ϵ transitions go to a state node, we duplicate it and its out-going transitions for each state to be merged), we obtain the state space of Schabes' algorithm. The resulting state transition automaton is still nondeterministic with each state containing only one kernel dotted rule. On the other hand, if we determinize the VP-NFA, we obtain another extreme partition, i.e., the state transition automaton for GLR parsers. Two measurements can be used for the state merge decisions. One is the maximum degree of nondeterminism for each state. In other words, one can merge two out-going transitions with the same label and from the same state to reduce the nondeterminism until the nondeterministic degree reaches a preset limit. The other measure is the maximum length of common prefix paths. Notice that the common prefix may contain nonterminals, therefore it is not simply the lookahead. If training data is available for the state space, similarly, we can search for the two parameters empirically, and decide the clusters. Our two measurements provide some details to the observation in [13] that allowing different degrees of nondeterminism in the state transition graphs would lead to different time and space complexity: Schabes' algorithm allows a maximum degree of nondeterminism in the state transition, while Tomita's algorithm allows a minimum degree of nondeterminism.

3 Parsing with Multiple Grammars

Weng and Stolcke [25] presented a top-down style algorithm that combines several parsers with partitioned subgrammars, with a restriction that the calling graph must not have left recursion, a somewhat stronger constraint than Tomita's original parser [15]. Here, we will present a flexible parsing composition algorithm that does not have such restriction on the grammar.

The intuition behind this new algorithm is that each subparser with a subgrammar will try to parse from the current input position and place its output nonterminal (virtual terminal) onto the same lattice, if successful. A subparser is invoked by a caller subparser only when certain predictive pruning conditions are satisfied to avoid excessive parser invoking. The representation used for interfacing different parsers is a special chart or lattice. The nodes of the lattice are spatial positions, and its transitions represent terminals/virtual terminals that cover positions indicated by their corresponding start and end nodes. Usually, a terminal covers one position, while a virtual terminal can cover multiple positions. During parsing, both terminals and virtual terminals are represented in the same lattice, but, partial constituents and nonterminals that are not the output of any subgrammar cannot be placed there. This implies that only the granularity represented by subgrammars is present in the lattice, and therefore, is termed as a lattice with multiple granularity (LMG). In addition, this representation gives us a good control over the lattice density for parsing. These features differentiate LMG from the traditional chart used in other parsing algorithms [8, 2, 17, 27].

Dealing with word lattice input for a single GLR parser was solved by Tomita [22]. Two steps in Tomita's algorithm are different from the standard GLR parsing algorithm: the lattice is topologically sorted for synchronization; for *shift* action, an additional procedure tests whether the current word *word*$_i$ is adjacent to any word on the top of the graph-structured stack. This same idea can be used for the Earley parser, left-corner parser, and shift-reduce parser. That is, instead of moving from i to $i+1$, the new algorithm will look for the *next* adjacent symbol to the current node in the lattice. However, direct application of this idea would force topological sorting to be performed each time a virtual terminal is added to the lattice. In the new composition algorithm, we use a stack to store the initially sorted lattice input, and dynamically add newly found virtual terminals to the stack in such a way that the topological order is always preserved for the changing LMG. By doing this, we avoid performing a constant topological sorting required by the simplistic approach.

In the original definition [25], we duplicated multiple subgrammars for a partition element with multiple output nonterminals. However, it is not necessary to make such duplication for the purpose of parsing because we can easily overcome the difficulty in handling multiple output symbols. To do so, we let $\{O^i_j\}^k_{j=1}$ be the output of a partition element G_i. If we add rule set $\{S_i \rightarrow O^i_j\}^k_{j=1}$ to G_i, use S_i as its start symbol for partition element G_i, and label the output node in the lattice with corresponding O^i_j during parsing, the correctness of derivation is still guaranteed. In the rest of this paper, without loss of generality, we can safely adopt the more general definition.

We have developed a new composition algorithm for parsers with different subgrammars, given a word lattice as its input. Without loss of generality, we use GLR parsers. Other parsers, such as Earley's parser or the shift-reduce parser, can be used in combination.

Let the word lattice be $L = (N, E)$, where N is a node set, indicating spatial positions, and E is a set of directed transitions, indicating the ranges the (virtual) terminals cover. Functions $start(dt)$, $end(dt)$, and $word(dt)$ return the start, the end, and the word label of directed transition dt, respectively. Let σ be the stack that stores the topologically sorted transition sequence from L.

Suppose that psr_i is the parser for subgrammar G_i, psr_0 is the parser for the top grammar G_0, and the largest subgrammar index is K. Each psr_i takes a (virtual) terminal stack σ as input (call by value), and returns a list of pairs with form (v, d), where d is the end node for virtual terminal v of subgrammar G_i if parsed successfully. If no parse is found, it returns $\emptyset$.

Parser Composition Algorithm: This algorithm replaces PARSE() and PARSEWORD() in Farshi's version of Tomita's recognition algorithm [15]. In this algorithm, ACTOR(A, Q, w), COMPLETER(R, Γ), and SHIFTER(Q, Γ) are similar to that version, with a few minor changes: w, the to-be-processed word, replaces a_{i+1} as the next word for processing in ACTOR; U_j, the state set at node j in L, replaces U_{i+1} as a state set next to U_i when w corresponds to the transition from U_i to U_j. U_i's and L are shared by all the parsers. However, graph-structured stack Γ and shared-packed forest are private to each parser. SP is a list of triples (pid, dt, σ), where pid is a parser id, dt is the transition being processed, and σ is the sorted transition stack. SP records all the parsers started at a particular location in L to avoid infinite recursion.

1. Topologically sort L into $dt_0, ..., dt_l$, and push all the transitions onto stack σ in the reverse order[1].

2. $\Gamma = \emptyset$. (initializing graph-structured stack).

[1]Notice that the resulting stack σ has $l+1$ element with w_0 on the top of σ.

```
3. Create in $\Gamma$ a vertex $v_0$ labeled with state id $s_0$, $U_0 = \{v_0\}$.
4. $SP = \emptyset$ (initializing the started parser list).
5. $dt = pop(\sigma)$, $SP = \{(0, dt, \sigma)\}$, $psr_0(dt, \sigma)$.
```

PARSER $psr_m(dt, \sigma)$, $m = 0, ..., K$:

```
1. Loop until $dt$ is null do
   (a) $i = start(dt)$, $j = end(dt)$, $w = word(dt)$.
   (b) $A = U_i$; $R, Q = \emptyset$.
   (c) repeat
         i. if $A \neq \emptyset$ then do ACTOR($A, Q, w$).
         ii. else if $R \neq \emptyset$ then do COMPLETER($R, \Gamma$).
       until $A == \emptyset \cap R == \emptyset$ (done with reductions).
   (d) PREDICTIVE-SUB-PARSING($dt, \sigma$).
   (e) SHIFTER($Q, \Gamma$).
   (f) $dt = pop(\sigma)$.
```

Predictive Sub-parser: The parameters for procedure PREDICTIVE-SUB-PARSING are call-by-name. In the procedure, $OUTPUT_k$ is the OUTPUT set of subgrammar G_k, and $PVT(U_i, w)$ is a function that takes a set of states and a (virtual) terminal, and returns a set of legitimate virtual terminals to be parsed at that point.

PREDICTIVE-SUB-PARSING(dt, σ):

```
1. $flag$ = true.
2. while $flag$ do
   (a) $flag$ = false.
   (b) $i = start(dt)$, $j = end(dt)$, $w = word(dt)$.
   (c) For $k$ = 1 to $K$ do
       if $OUTPUT_k \cap PVT(U_i, w) \neq \emptyset$ and $(k, dt, \sigma) \notin SP$ then do
         i. $SP = (k, dt, \sigma) \cup SP$.
         ii. $lvt = psr_k(dt, \sigma)$.
         iii. For $(m, d) \in lvt$, $m \in PVT(U_i, w)$ and $m$ is not in $L$ to cover from $i$ to $d$ then[2]
              A. add $m$ to $L$: create a transition $mdt$ with label $m$ that connects from node
                 $i$ to node $d$.
              B. push $mdt$ onto stack $\sigma$.
              C. $flag$ = true.
```

[2]For getting all the possible parse trees, somewhat more complicated test needs to be made, that is, to see whether the top level structures (rules) have already been built.

Notice that function PREDICTIVE-SUB-PARSING only adds virtual terminals to the stack and lattice, and these virtual terminals have the same start node as the transition being processed. Therefore, the updated stack still maintains the topological order for the updated lattice. This leads to a straightforward correctness proof of the algorithm.

In PREDICTIVE-SUB-PARSING, function $PVT(U_i, w)$, serving as a subparser filter, has not been defined. The occurrence of $PVT(U_i, w)$ in the algorithm is not essential for its correctness. However, without PVT it would start multiple parsers at every node in the lattice, that is, a big efficiency concern. Our technique for addressing this problem resembles predictive calling of subparsers in the strict top-down composition algorithm [25], or $FIRST$ in the conventional parsing table generator.

Let $VT(i) = \{vt | i \text{ is a state}, vt \text{ is a virtual terminal}, \exists s, \textit{shift } s \in \text{ACTION}(i, vt)\}$. The predictive set of virtual terminals for a given state i and a (virtual) terminal w is defined as $PVT(i, w) = \{vt | vt \in VT(i), w \text{ is a (virtual) terminal}, VT(i), vt \overset{*}{\Rightarrow} w ...\}$. To overload the notation a little bit, we define the predictive set of virtual terminals for a set of given states Q and a (virtual) terminal w as: $PVT(Q, w) = \bigcup_{i \in Q} PVT(i, w)$. The algorithm to realize the definition is straightforward, and therefore omitted.

Intuitively, for any state in U_i, if there is a *shift s* step under a virtual terminal for that state in the ACTION table of the caller parser, we add the virtual terminal to $PVT(U_i, w)$. Those virtual terminals that do not have w as their left corner can be pruned.

For other types of parsers, we can also compute a set of virtual terminals, given a nonterminal (or a set of nonterminals) and a left corner, similar to the way PVT or $FIRST$ is computed. Thus, we can avoid invoking subparsers excessively.

In addition to predictive pruning, parallelization is an approach to the efficiency problem. Notice that all legitimate subparsers in PVT for a particular transition in the lattice can be parallelized without affecting other parts of the algorithm.

To address both parsing accuracy and efficiency simultaneously, we can incorporate probabilistic models into lattice. Estimation of ngram word transition models is standard. In our lattice framework, head words can be passed to virtual terminals for more detailed probabilistic estimation to better capture certain long distance dependency without lexicalizing the grammar. The latter approach leads to a huge grammar in any practical system [6]. However, a potential sparse data problem for the introduction of lexicalized virtual terminals can be partially addressed by the relatively easy-to-control granularity of grammar partitioning.

To address the robustness for real-world applications, the techniques used in the EGLR parser [24] can be adapted in the lattice framework, because, in the EGLR parser, the three editing operations (insertion, deletion, and substitution) can be applied to both terminals and virtual terminals. Modifying input is equivalent to adding transitions in the lattice [11, 26]. The only additional issue is that the cost function for the optimal path selection may not be a uniform unit cost, but should be associated with the number of words changed when virtual terminals are in the editing operations.

4 Experiments

Ideally, we want to test grammar partitioning and parser composition in its full scale. However, before the full implementation of the algorithms described above, we took a shortcut to evaluate the feasibility in a real-world natural language application. We describe a concrete partition of Chinese and English grammars in an ATIS domain, and a robust cascading parsing and composition. We then report the parsing and understanding results for the ATIS domain.

4.1 Cascading Parsing for the ATIS Grammars

English ATIS tasks contain queries regarding air travel information, including flights leaving from or arriving at locations at various dates and times, services of certain flights, and airline routes. The Chinese queries used in the experiments are translated from an English ATIS corpus.

Based on the guidelines for grammar partitioning, we manually partitioned the rule set into subgrammars, such as STATE-NAMES, CITY-NAMES, AIRPORT-CODE, with corresponding virtual terminals prefixed with *vt*. In this particular case, the calling graphs of the subgrammars are acyclic and, therefore, they can be assigned with level indices (ids). Their corresponding subparsers are GLR parsers, compiled separately, and assigned with the same ids. The top-level sentence subgrammar is not yet completed and deployed, because of the flexible word order problem in Chinese and many variations of sentence structures in English. However, the LMG representation allows a robust parser to compose all the legal sequences of virtual terminals and choose the best path that covers the most words in the input string.

The whole parsing process proceeds as follows. The control mechanism takes a query sentence and converts it into an LMG. Then, the subparsers at the lowest level are activated to parse the LMG, and leave their corresponding virtual terminals on the LMG when parsing succeeds. If no more virtual terminals can be added to the LMG, the subparsers at one level higher are activated and start to parse the same LMG[3]. This process continues until the highest level is reached, when the robust sentence-level parser finishes its job of determining the best virtual terminal sequence. For the semantic evaluation described in subsection 4.2, a semantic interpreter converts syntactic virtual terminals into semantic frames of key-value pairs.

The main difference between the parsing composition algorithm described in Section 3 and the one used here is the adoption of a viterbi-style robust master parser instead of a GLR parser. This gives us a flexible way to deal with extra-grammatical queries even without any sentence-level grammar rules. As a consequence, it also derives a preliminary set of sentence-level grammar rules for future refinements as we will see in Section 4.2. On the other hand, when both predictive pruning and robust handling are integrated and tuned in a GLR parser, it can replace the current one without affecting the other components, which is the desired feature of the approach described in this paper.

4.2 Experimental Results

Our experiments use the English ATIS-3 corpus and its translated Chinese version, specifically the Class A queries. It contains 1564 queries from the training set, 448 queries from the 1993 test set and 444 queries from the 1994 test set. Each utterance is labeled with a corresponding SQL query for information retrieval.

[3]When there is a recursion at the same level of the calling graph, we can repetitively activate the subparsers at that level until no more virtual terminals can be added to the LMG, which corresponds to the 'while *flag* loop' in Predictive sub-parser algorithm.

For the Chinese experiment, the non-sentence level grammar rules used for parsing were developed based on 375 queries from the MIT subset of the training set, and the remaining 1189 queries of the training set were held for tuning. When parse errors occur in any of the held-out sentences, we modify the grammar rules to incorporate the changes. Evaluations were conducted on 1993 and 1994 test sets. Table 1 shows the general statistics of the hand-crafted grammar. As we can see from the table, the majority of the rules (825 out of 1013) are directly related to the lexicon, and only about 178 rules are related to phrase structures.

Applying the parsing algorithm described in Section 4.1 and the above grammar on the training set and the two test sets, we obtained the results shown in Table 2, where the error rates are measured against the Chinese translation of the standard semantic key-value pairs supplied in the ATIS-3 corpus by Linguistic Data Consortium. *Full understanding* refers to utterances with full matches between the semantic categories in our frame and those in the standard answer with no error. *Part. understanding* refers to partial matches with an error rate between zero and one. *No understanding* refers to no matches.

In addition to the incomplete grammar coverage, parsing errors can have sources in translation errors or implicit information. The former are mainly caused by wrong or imprecise translation from English to Chinese. The latter are due to a discrepancy in the semantic meaning translation between the standard answers and our semantic interpreter. For example, *tonight* has semantic meaning of $time \geq 1800 \& time \leq 2359$, while our interpreter has only a symbolic value. We are developing a correction for this discrepancy.

We also summarized the parsing statistics in Table 3. From the table, we observe that the speed of our parser is very fast for this application. In addition, the number of rules reduced and states visited in the best paths and in the successful sub-parses are less than those in the overall parsing process, which means that further pruning is possible.

Because the current master parser does not use explicit sentence-level grammar rules, we do not have direct parsing table size comparison between partitioned and unpartitioned grammars. However, as an approximation, we can take as the sentence level rules those best paths from all the training data that occur more than once. In the Chinese experiment, we obtained 205 such rules, in addition to the original 1,013 lower level rules. For the unpartitioned grammar, the total number of GLR states is 10,395, while for the partitioned grammars, the total number of GLR states is only 2,825, out of which 852 states are used by the sentence-level subgrammar.

To speed up the English experiment, we constructed English lower level rules based on the Chinese rules and repeated the same processes for parsing and for obtaining English sentence level rules as we did for Chinese. English parsing results are listed next to its Chinese counterpart in Tables 1 to 3. There are additional 198 sentence level English rules. For the unpartitioned grammar, the total number of GLR states is 10,233, while for the partitioned grammars, the total number of GLR states is only 2,128, out of which 671 states are used by the sentence-level subgrammar. Both Chinese and English experiments show that, at least in the ATIS domain, grammar partitioning leads to the desirable effect of reducing parser sizes.

5 Conclusions and Future Work

We have presented guidelines for grammar partitioning through a set of heuristics, and proposed a new mix strategy parser composition algorithm based on a flexible LMG representation, so that we can alleviate size, speed, and robustness problems in real-world natural language applications. We also reported our initial but encouraging parsing results in the ATIS domain.

We plan to further integrate the predictive robust parsing algorithm [24] into the current system, and to use probabilistic modeling for better pruning and selecting of correct parses. To alleviate the current grammar writing bottleneck in our parsing experiments, we will also work with semi-automatically induced grammars [20].

Acknowledgments

Part of this work was done while the first author was visiting the Chinese University of Hong Kong. This work is mostly supported by the Direct Grant from the Chinese University of Hong Kong, and, in part, supported by SRI International, under Mediated Space project.

References

[1] A. Aho and J. Ullman. *Principles of Compiler Design.* Addison-Wesley, Reading, MA, 1977.

[2] J. Amtrup. Parallel Parsing: Different Distribution Schemata for Charts. In *IWPT1995*, 1995.

[3] H. Chen, J. Qian, and Y. Sun. *Principles of Compilation for Programming Languages (in Mandarin).* XinHua Publisher, Beijing, 1984.

[4] J. Earley. *An Efficient Context-free Parsing Algorithm.* Ph.D. thesis, Carnegie Mellon University, 1968.

[5] J. Earley. An Efficient Context-free Parsing Algorithm. *Communications of ACM*, 13(2), 1970.

[6] J. Eisner and G. Satta. Efficient parsing for bilexical context-free grammar and head automaton grammars. In *Proceedings ACL 1999*, 1999.

[7] M. Johnson. The Computational Complexity of Tomita's Algorithm. In *Proceedings of the 1st International Workshop on Parsing Technologies*, 1989.

[8] M. Kay. Algorithm Schemata and Data Structures in Syntactic Processing. In B. Grosz, K. Jones, and B. Webber, editors, *Readings in Natural Language Processing.* Morgan Kaufmann Publishers, Inc., 1986.

[9] J. Kipps. Analysis of Tomita's Algorithm for General Context-Free Parsing. In *Proceedings of the 1st International Workshop on Parsing Technologies*, 1989.

[10] A. Korenjak. A Practical Method for Constructing LR(k) Processors. *Communications of ACM*, 12(11), 1969.

[11] B. Lang. A Generative View of Ill-Formed Input Processing. In *ATR Symposium on Basic Research for Telephone Interpretation (ASTI)*, 1989.

[12] A. Lavie and M. Tomita. An Efficient Noise-Skipping Parsing Algorithm for Context-Free Grammars. In *Proceedings of the 3rd International Workshop on Parsing Technologies*, 1993.

[13] M.-J. Nederhof. An Optimal Tabular Parsing Algorithm. In *Proceedings of ACL 1994*, 1994.

[14] M.-J. Nederhof and G. Satta. Efficient Tabular LR Parsing. In *Proceedings of ACL 1996*, 1996.

[15] R. Nozohoor-Farshi. GLR Parsing for e-Grammars. In Masaru Tomita, editor, *Current Issues in Parsing Technology*. Kluwer Academic Publishers, 1991.

[16] M. Rayner and D. Carter. Fast Parsing Using Pruning and a Grammar Specialization. In *Proceedings of 1996 ACL*, 1996.

[17] T. Ruland, C. Rupp, J. Spilker, H. Weber, and K. Worm. Making the Most of Multiplicity: A Multi-Parser Multi-Strategy Architecture for the Robust Processing of Spoken Language. In *Proceedings of ICSLP 1998*, 1998.

[18] H. Saito and M. Tomita. GLR Parsing for Noisy Input. In *Proceedings of the 2nd International Workshop on Parsing Technologies*, 1991.

[19] Y. Schabes. Polynomial Time and Space Shift-Reduce Parsing of Arbitrary Context-Free Grammars. In *Proceedings of 1991 ACL*, 1991.

[20] K. Siu and H. Meng. Semi-automatic Acquisition of Domain-specific Semantic Structures. In *Proceedings of the 6th European Conference on Speech Communication and Technology*, 1999.

[21] M. Tomita. *Efficient Parsing for Natural Language*. Kluwer Academic Publishers, 1985.

[22] M. Tomita. An Efficient Word Lattice Parsing Algorithm for Continuous Speech Recognition. In *Proceedings of ICASSP-86*, 1986.

[23] E. Ukkonen. Lower Bounds on the Size of Deterministic Parsers. *Journal of Computer and System Sciences*, 26:153–170, 1983.

[24] F. Weng. Handling Syntactic Extra-grammaticality. In *Proceedings of the 3rd International Workshop on Parsing Technologies*, 1993.

[25] F. Weng and A. Stolcke. Partitioning Grammars and Composing Parsers. In *Proceedings of the 4th International Workshop on Parsing Technologies*, 1995.

[26] F. Weng and Y. Wang. *Introduction to Computational Linguistics (in Mandarin)*. Sino Social Sciences Publisher, Beijing, 1998.

[27] K. Worm. A Model for Robust Processing of Spontaneous Speech by Integrating Viable Fragments. In *Proceedings of COLING/ACL 1998*, 1998.

Types of rules	sizes		Remarks
	Chinese	English	
non-top-level rules	1013	953	all the rules in the grammar
lexical rules	825	773	rules deriving lexical entries
nonterminals	77	81	all the nonterminals
virtual terminals	50	52	all the virtual terminals
terminals	419	473	all the words

Table 1: Grammar rule statistics.

	Training set 1564 queries		1993 test set 448 queries		1994 test set 444 queries	
	Chinese	English	Chinese	English	Chinese	English
Ave. sent. length (# of chars/words)	19.33	11.24	16.63	10.26	20.17	11.46
Ave. error rate (%)	7.94	7.30	11.22	9.69	13.61	12.40
Full understanding (%)	80.56	83.95	77.90	87.05	70.95	76.35
Part. understanding (%)	17.26	14.13	17.41	10.49	23.87	21.40
No understanding (%)	2.17	1.92	4.69	2.46	5.18	2.25

Table 2: Error rate and accuracy statistics.

	Training set 1564 queries		1993 test set 448 queries		1994 test set 444 queries	
	Chinese	English	Chinese	English	Chinese	English
Total time for entire set (in seconds, Ultra sparc 5)	21	14	5	4	7	5
Ave. no. of states visited incl. failed subparsers	64.9	36.6	53.3	33.3	68.5	38.8
Ave. no. of rules reduced incl. failed subparsers	13.4	14.6	11.3	13.2	13.7	15.0
Ave. no. of states visited, only successful subparsers	39.0	30.2	31.3	27.0	40.1	31.1
Ave. no. of rules reduced, only successful subparsers	12.7	13.9	10.4	12.5	12.9	14.3
Ave. no. of states visited in the best paths	30.7	24.1	26.0	21.6	31.1	24.7
Ave. no. of rules reduced in the best paths	9.5	11.0	8.2	9.9	9.5	11.1

Table 3: Statistics of the grammar used for the experiments.

Modular Unification-Based Parsers

Rémi Zajac and Jan W. Amtrup
Computing Research Laboratory, New Mexico State University
`zajac,jamtrup@crl.nmsu.edu`

Abstract
We present an implementation of the notion of modularity and composition applied to unification-based grammars. Monolithic unification grammars can be decomposed into sub-grammars with well-defined interfaces. Sub-grammars are applied in a sequential manner at runtime, allowing incremental development and testing of large coverage grammars. The modular approach to grammar development leads us away from the traditional view of parsing a string of input symbols as the recognition of some start symbol, and towards a richer and more flexible view where inputs and outputs share the same structural properties.

1 Introduction

We present a modular unification-based parsing architecture which allows to build a parser as a sequence of sub-components that can be developed, tested and executed one at a time. This modularization reduces the overall complexity of the grammar and enables incremental development of grammars. The motivation comes from practical problems encountered in developing broad coverage grammars where linguists face the same problems as those addressed by software engineering [Zajac 92b] and we seek similar solutions: modular approach to 'lingware' construction, encapsulation, typing, incremental compilation.

Unification-based grammars represent a definite advance in comparison to previous procedural parsing systems such as ATNs but are still notoriously difficult to develop and debug. The complexity stems mainly from the interaction between a large set of rules, where a modification to one rule can cause the parser to fail several steps after the successful application of the rule, making it difficult to trace back the source of the failure. Of course, this problem is shared by procedural parsing systems as well, where tracing back is even more difficult since the state of the system may have changed and may be irrecoverable. In a unification-based system, data is changed monotonically, making it possible, at least in theory, to recover any previous state of the system. However, most systems do not record states of computation, and previous states are only partially recoverable.

Simply inspecting the final result of a parse, although it typically contains all the derivation information, is very hard since the structures are fairly large and include a lot of detailed linguistic information. The other alternative, tracing the parsing process at runtime is also very difficult since most parsing algorithms (e.g., chart-based parsing algorithms), are non-deterministic and, for some parsers, do not proceed in a left-to-right fashion (e.g. island-driven parsers).

Some experimental Prolog environments include debuggers that allow to trace back through any branches of a computation. This approach is attractive but fairly complex to implement, and does not solve one of the main problems, which is the debugging of a single monolithic grammar.

In this paper, we present a modular parsing architecture that attempts to address some of these problems. The architecture is derived from ideas developed in the Tipster project [Zajac, Casper & Sharples 97, Zajac 98a] as well as several transformational parsing systems, including Q-systems [Colmerauer 71] and tree-transducers such as GRADE [Nakamura, Tsujii & Nagao 84] or ROBRA [Vauquois & Boitet 85]. The grammars that are used in the system are rule-based unification grammars using a context-free skeleton (extended context-free grammars): LFG is an example of this type of grammars. Grammars are combined using a composition operator similar to the regular relation as described in [Kaplan & Kay 94] (see also [Wintner 99] for related ideas on modular context-free grammars), and each grammar specifies its interface as a set of input and output types.

Section 1 presents an overview of the parsing architecture, and how sub-grammars can be combined to build a parsing application.

Section 2 presents the unification-based parser and grammar formalism used in the system along with examples from a parser developed for a Persian-English machine translation system.

Section 3 discusses the modular organization of grammars from a linguistic point of view ands mention some runtime complexity figures of a modular parsing system.

2 Parsing Architecture

The parsing architecture is derived from previous work on NLP architectures within the Tipster framework [Zajac, Casper & Sharples 97, Zajac 98a, Bird & Liberman 99] and combines ideas from early modular NLP systems such as Q-systems [Colmerauer 71] and tree-transducers such as GRADE [Nakamura, Tsujii & Nagao 84] or ROBRA [Vauquois & Boitet 85], which provide the linguist which very flexible ways of decomposing a complex system into small building blocks which can be developed, tested and executed one by one. It uses a uniform central data structure which is shared by all components of the system, much like in blackboard systems [Boitet & Seligman 94], and incorporating ideas on chart-based NLP [Kay 73, Kay 96, Amtrup 95, Amtrup 97, Amtrup & Weber 98, Amtrup 99, Zajac 99a]. The traditional chart structure, which stores linguistic information on edges and where nodes represent a time-point in the input stream, are augmented, following Tipster ideas on 'annotations', with tags which define the kind of content an edge bears, and with spans (pairs of integers) pointing to a segment of the input stream covered by the edge. Spans are used for example in debugging and displaying a chart with edges positioned relative to the input text they cover. Tags identify for example edges built by a tokenizer, morphological analyzer, or a syntactic parser, and define sub-graphs of the whole chart that are input to some component. Linguistic information on edges is encoded using a weaker version of the Typed Feature Structure formalism as presented for example in [Zajac 92a].

Using a centralized data structure has several advantages: a chart gives a precise idea on the state of the system at any point in the computation, and integrating components to build larger systems become easy since all components implement the same interface.

In this architecture, all parsers operate in a bottom-up fashion on the same data structure: the input of a parser is a chart, maybe reduced to a linear sequence of tokens, and its output is also a chart, where active edges have been removed (and possibly edges representing sub-constituents). To operate, the first parser obviously needs a component that can read the input text and convert it to a chart.

In current applications, this is done by a tokenizer and/or a morphological analyzer which produce a chart of word structures. The tokenizer reads the input stream, classifies each token as (potential) words, numbers, symbols, or punctuation characters,[1] and produces a chart where each edge contains the analyzed token. A morphological analyzer component reads the token chart and produces for each input token one or several analyses, each stored in a different edge. A morphological analyzer can be some external program for which a wrapper implementing the chart interface has been developed, or be a pre-built component parameterized by a unification-based morphological grammar [Zajac 98b]. In turn, each parser operating on the chart is a component parameterized with a unification-based grammar.

A working system can easily be assembled from a set of components by writing a resource file, called an application definition file, which describes instantiations of components, the calling sequence of components and various global variables and parameters. Assembling components together is done using a composition operator which behaves much like the Unix pipe. When, in a Unix command, data between programs is transmitted using files (stdin/stdout) and programs are combined using the pipe '|' command, in MEAT, the data transmitted between components is a chart, and syntactically the sequence of component is combined using the ':' composition operator. In effect, the MEAT system is a specialized shell for building NLP systems. The implementation language is C++ but external components can be integrated in the system by writing wrappers (as done for several morphological analyzers previously built or used at CRL).

Each component is an instance of a C++ class which implements a standard interface. The application definition file that parameterizes the system and defines actual applications consist of three sections:

- A variable section defines global variables that typically identify locations of resources or the value of some parameters shared by several components.
- In the parameter section, global parameters are component parameters shared by several components. For example, a global parameter used by almost all components is the file containing the set of TFS definitions defining types and associated feature structures that are built by components and stored on edges of the chart.
- In the component section, each component is described as an instance of a C++ class and a list of parameters for that component (e.g. a grammar and a start symbol for a parser).
- In the application section, an application is simply defined as a sequence of components.

This application definition file is read at runtime by the system and the user provides the name of the application on the command line together with the required parameters (e.g., the input text file for a parser). For example, an application definition file would look like:

```
$ROOT=$HOME/arabic/

$MORPH=$ROOT/src/arabic.samba
$SYN=$ROOT/src/arabic.bolero
```

[1]The system includes a Unicode-base tokenizer that can operate on most scripts using spacing characters, and a large number of codeset converters for most writing systems.

```
tangoModule = $ROOT/runtime/arabic.tangoc

module Tok {
 class =  Tokenizer
 targetTag = TOKEN
 verbose = true
 inputFile = $ifile
 encoding = cp1256
 }

module Ama {
  class = MorphAnalyzer
  grammar = $MORPH
  rule = Word
  type = chart
  sourceTag = TOKEN
  targetTag = MORPH
  }

module Syn {
  class = Parser
  grammar = $SYN
  successTypes = ara.LSign
  inTag = MORPH
  outTag = SYN
  }

application parse = Tok($ifile=$1):Ama:Syn:SaveChart
```

The user would then use this application definition file (called say 'myapp') as a parameter of the meat executable:

```
% meat myapp parse test/alf.txt
```

The system would save the chart in some file that can they be browsed using the chart viewer (also a component of the system).

3 Modular Unification-Based Parsers

The parser component of the MEAT system implements a bottom-up island-driven parsing algorithm [Stock et al. 88], with a few modifications geared towards efficiency. In each rule, it is possible to identify one element of the right-hand side as the island which will be searched before attempting to apply the rule. The default behavior when no island is specified is a left-to-right regular chart-parsing algorithm.

Islands can be used to avoid triggering rules on the first element of the left-hand side when this element can appear much more frequently than the island. For example, the left-hand side of a rule describing the coordination of noun phrases would start with a noun phrase, and be triggered for each noun phrase in the default left-to-right parsing strategy. Specifying the coordinator as the island allows the rule to be triggered only in the presence of a coordination.

Elements of the right-hand side can also be specified as optional, reducing the number of disjunctive rules in the grammar.

It is also possible to specify that when a rule successfully applies, its constituents can be erased from the graph. This is used in the Persian system for example when parsing complex predicates with auxiliaries. This grammar, although written using an extended context-free formalism, is actually a regular deterministic grammar. Therefore, when an auxiliary can be integrated in a larger constituent, we can be sure that this auxiliary will never be needed to form another constituent, and we can delete the corresponding edge from the graph. This does not influence the correctness of the result but results in a reduced number of edges that have to be examined by the parsing algorithm, and reduces the number of useless constituents that will never be integrated in the final result (spurious intermediary constituents). Reducing the number of edges has also a direct benefit in terms of debugging, since the chart becomes less cluttered with useless information.

The system also includes a component that removes all edges that do not belong to a shortest path in the graph, thereby implementing a maximum coverage heuristics. This component can be used instead of the erase facility that is local to a rule, in particular when all intermediary results are needed to ensure completeness.

Each parser is created as an instance of the component Parser such as in the following definition:

```
module Syn {
  class = Parser
  grammar = $HOME/arabic/ara.bolero %$
  inTag = MORPH
  outTag = SYN
  successTypes = ara.LSign
  }
```

This definition specifies that the grammar parameterizing the parser is **ara.bolero**, that the input edges that the parser should consider are all edges tagged as **MORPH** (output of the morphological analyzer), and that the edges built by the parser as tagged as **SYN**. Moreover, the edges to be considered as part of the final result must be subsumed by the type **ara.LSign** and all other **SYN** edges are erased.

A parser's output is not an edge or a set of edges covering the whole input sentence but a chart containing all edges built by the parser that belong to a particular type. The measure of parsing success is therefore not the recognition of a single edge spanning the input, but the production of a graph, preferably connected, where edges belong to a success type. Thus, a parse is not a boolean answer anymore, but a graded result where graph connectivity, graph topology and edge type play a role. This approach proves extremely useful in building robust NLP systems which performance degrades gracefully on 'ungrammatical' input [Zajac 99b].

Since input and output are of the same kind, parsers can be chained together, the output edges of one parser becoming the input to the next parser. In the Persian system for example, there is one[2] sub-grammar used to analyze complex verbal predicates (conjugated forms with auxiliaries). For the next level of parsing, each complex verbal predicate is then viewed as a single word and sub-constituents of a complex predicate are erased from the chart.

A rule is specified in the feature structure notation and each syntactic element follows the general feature structure syntax. Although this makes it sometimes a little bit awkward, it allows to compile rules as feature structures which are themselves compiled as compact arrays of integers[3] and enable

[2]Actually 2, the second being used to parse light verb constructions.

[3]The unification algorithm operates on this data structure. Arrays of integers can also be written or loaded from a

very fast access of the rule at runtime (see for example [Wintner 97]).

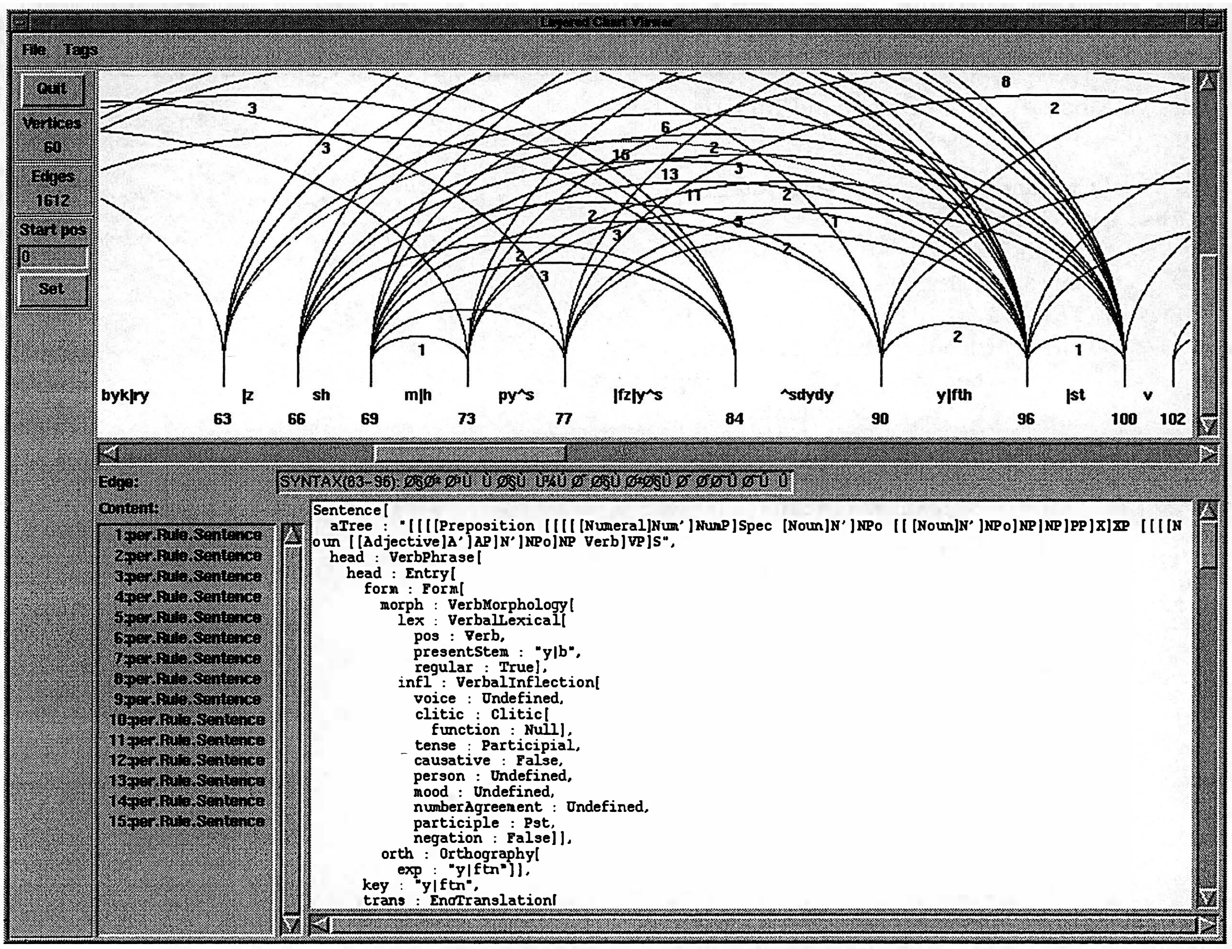
A simple rule describing the Persian Passive Compound Imperfect as a Past Participle followed by Passive Auxiliary in the Compound Imperfect tense:

```
PassiveCompoundImperfect = per.Rule[
  lhs: per.Verb[
         trans: #eng,
         exp: #orth,
         lex: #lex,
         infl: [mood: per.Indicative,
                tense: per.CompoundImperfect,
                voice: per.Passive,
                person: #per,
                numberAgr: #num,
                causative: #caus,
                participle: #part,
                negation: #neg]],
  rhs: <:
    per.Verb[
         trans: #eng = Top,
         exp: #orth = Top,
         lex: #lex= Top,
         infl: [participle: #part= per.Pst,
                causative: #caus = Top,
                negation: False]]
    per.Verb[
         exp: "^sdn",
         infl: [tense: per.CompoundImperfect,
                mood: per.Indicative,
                voice: per.Active,
                person: #per = Top,
                numberAgr: #num = Top,
                negation: #neg = Top]]
      :>,
   remove: True
   ];
```

In this rule, unlike grammars written in the LFG or HPSG-style, a feature structure is created anew for the left-hand side: the left-hand side is not simply the head of the construction percolated up. Therefore, all head features that need to be transmitted need to be specified in the rule. This style makes each rule heavier than its LFG counterpart[4] but allows to categorize each constituent as belonging to a different class, e.g. a different bar-level. For example, the following rule builds an N' from a noun. Rules for noun phrases will then consider only N' and not Nouns, and so on. Recategorization of heads is not possible if the head itself becomes the left-hand-side constituent. Note that this style also make grammars inherently more reversible (assuming that generation would start with a syntactic structure) since we would only need to specify the head in this rule (specified here as the island).

file very efficiently, a facility which is used to pre-compile grammars as arrays of integers as well.

[4]It is of course possible to write LFG-style rules in this formalism.



```
SYNTAX(63-96): Ø§Ø* Ø¹Ù Ù Ø§Ù Ù¼Ù Ø Ø§Ù Ø÷Ø§Ù Ø ØØ Ù Ø Ù Ù
```

```
Sentence[
  aTree : "[[[[Preposition [[[[[Numeral]Num']NumP]Spec [Noun]N']NPo [[[Noun]N']NPo]NP]NP]PP]X]XP [[[[N
oun [[Adjective]A']AP]N']NPo]NP Verb]VP]S",
    head : VerbPhrase[
      head : Entry[
        form : Form[
          morph : VerbMorphology[
            lex : VerbalLexical[
              pos : Verb,
              presentStem : "y|b",
              regular : True],
            infl : VerbalInflection[
              voice : Undefined,
              clitic : Clitic[
                function : Null],
              tense : Participial,
              causative : False,
              person : Undefined,
              mood : Undefined,
              numberAgreement : Undefined,
              participle : Pst,
              negation : False]],
          orth : Orthography[
            exp : "y|ftn"]],
        key : "y|ftn",
        trans : EngTranslation[
```

Figure 1: Viewing a complex analysis with the chart viewer.

```
NounBarNoEzafe = per.Rule[
  lhs: per.NounBar[
      head: #head,
      boundary: per.NPtrue],
  rhs: <:
      #head= per.NounOrNounCompound[
             infl: [ezafe: per.EzFalse,
                    indefEncl: False,
                    clitic.function: per.Null]]
      :>
  island: #head
  ];
```

4 Modularizing a Grammar

As already hinted in the previous section, a monolithic grammar can be decomposed in manageable sub-parts that can be developed and tested separately. Two orthogonal ways of classifying constituents can be used to identify sub-grammars:

1. A classification of constituents in levels of increasing complexity, reminiscent of the X-bar theory. Level 0 would represent so-called pre-terminal elements built directly and exclusively from lexical elements. Level 1 would provide the last interface between morphology and syntax, analyzing free morphemes (e.g. clitics), auxiliaries, non-lexicalized compounds, etc., and build new constituents from Level 0 and Level 1 constituents only. Level 1 grammars would typically fall in the class of regular grammars. Level 2 would analyze modifiers of heads, where the modifier would be from a different category (e.g. adverbs modifying adjectives, adjectives modifying nouns, but not nouns as complements of nouns). Level 2 grammars do not build recursive structures and therefore also fall in the regular class (although the form of the rules would be the one of general context-free rules). Finally, Level 3 would include recursive structures (nominal complements, relative clauses, coordination, etc.)

2. A classification according, broadly, to the category of the head: the traditional division between noun phrases, verb phrases, adjectival phrases, etc. This classification is a partial order since for example an adjective can modify a noun, but a noun will not be a subconstituent of an adjectival phrase. It can be used to order the application of sub-grammars. For example, adjectival phrases must be built before noun phrases, adverb phrases before adjectival phrases, and so on. At the higher level, level 3 constituents include about any other constituent, but at lower levels, for example adverb phrases, level 3 involve mostly or solely coordination of constituents of the same category.

In a classification of constituents into adverb phrases, adjective phrases, determiner phrases, noun phrases, verb predicates, verb phrases, and clauses, a complex monolithic grammar could be ideally be decomposed into 28 sub-grammars, each of them describing a well defined set of constructions. The grammar writer could then define a parser for each of these grammars, define classes of input/output using a set of corresponding tags, and test separately each sub-grammar in turn.

Additionally, after most of grammars from level 0, 1 and 2, it is possible to remove sub-constituent edges from the chart since level 0 and level 1 constituents are typically not ambiguous (contrary to level 3, where we find the traditional PP attachment problems and the likes), thereby reducing the number of hypotheses that the parsing algorithm needs to consider. Edge reduction has also the advantage of facilitating the inspection of the chart for debugging purposes.

5 Conclusion

The MEAT parsing architecture has been developed within the Corelli project at CRL. The system has been implemented in C++ and is used to deliver applications running on Unix and Windows platforms. Strings are internally represented using Unicode, and a set of Unicode converters is also included in the system. The system has been used for developing the Shiraz Persian-English machine translation system (see *http://crl.nmsu.edu/shiraz/*) and for the Turkish-English MT system developed within the Expedition project. It has also been used to reimplement previous glossary-based MT systems for Arabic, Japanese, Russian, Spanish and Serbo-Croatian developed in the Temple project and to develop new MT systems (e.g, Korean-English, see *http://crl.nmsu.edu/corelli/*). Finally, it is also used as the machine translation infrastructure for the Expedition project (*http://crl.nmsu.edu/expedition/*).

The two main parsers developed using this architecture are a Persian parser (used in the Shiraz MT system) and a Turkish parser (used in the Expedition Turkish MT system). The Persian grammar has been developed in 4 months and includes a hundred rules. It covers noun phrases, relative clauses, and basic sentential constructions. The bilingual Persian-English dictionary contains about 42,000 entries (lemmas). The morphological analyzer has been developed using the Samba unification-based system [Zajac 98b]. The Persian MT system has been developed and tested on a corpus of 3000 sentences extracted from on-line news articles. The Turkish grammar (still being developed) includes about ninety rules; it has been developed in 2 months and covers noun phrases, nominalized relative clauses, some nominalized subordinate clauses, basic sentential constructions including free word order. The Turkish morphological analyzer is the two-level system developed by Kemal Oflazer [Oflazer 94]. The bilingual Turkish dictionary contains about 32,000 entries. The MT system is being developed and tested on a corpus of 260 sentences also extracted from on-line news articles.

Performances compare favorably with other parsing systems (see e.g. the comparison between ALE and AMALIA in [Wintner, Gabrilovitch & Francez 97]). Compilation time for a Persian sub-grammar of 80 rules takes about .25 seconds on a Sparc5 and parsing an average sentence (25 words) takes about .22 seconds. This time does not take into account pre-processing of input text (tokenization, morphological analysis and dictionary lookup). Total parsing times below includes the execution of edge removal executed after G2.

	G1: 5 rules	G2: 16 rules	G3: 80 rules	Total: 101 rules
Compilation	0.040s	0.120s	0.210s	0.370s
Parsing 25 words	0.020s	0.040s	0.220s	0.300s
Parsing 80 words	0.050s	0.100s	0.660s	0.820s
Parsing 700 words	0.210s	0.520s	10.100s	10.880s

The MEAT environment is publicly available under the general GNU license agreement.

Acknowledgments

The MEAT system has been implemented by Jan Amtrup and Mike Freider with contributions from many other people at CRL. The Persian and Turkish grammars have been developed by Karine Megerdoomian.

This research has been funded in part by DoD, Maryland Procurement Office, MDA904-96-C-1040.

References

[Amtrup 95] Jan W. Amtrup. 1995. “Chart-based Incremental Transfer in Machine Translation”. Proceedings of the *6th International Conference on Theoretical and Methodological Issues in Machine Translation - TIM'95*, 5-7 July 1995, Leuven, Belgium. pp188-195.

[Amtrup 97] Amtrup J. (1997). "Layered Charts for Speech Translation". In Proceedings of the Seventh International Conference on Theoretical and Methodological Issues in Machine Translation, TMI '97, Santa Fe, NM, July 1997.

[Amtrup 99] Amtrup J. (1999). *Incremental Speech Translation.* Lecture Notes in Artificial Intelligence 1735, Springer Verlag, 1999.

[Amtrup & Weber 98] Jan Amtrup & Volker Weber (1998). "Time Mapping with Hypergraphs". In Proceedings of COLING'98, Montreal, Canada.

[Colmerauer 71] Alain Colmerauer (1971). "Les systèmes-Q: un formalisme poyr analyser et synthétiser des phrases sur ordinateur. Groupe TAUM, Université de Montréal.

[Bird & Liberman 99] Steven Bird & Mark Liberman (1999). "Annotation graphs as a framework for multidimensional linguistic data analysis". http://xxx.lanl.gov/abs/cs.CL/9907003.

[Boitet & Seligman 94] Christian Boitet and Mark Seligman. 1994. "The Whiteboard Architecture: a Way to Integrate Heterogeneous Components of NLP Systems". Proceedings of the *15th International Conference on Computational Linguistics – COLING'94*, August 5-9 1994, Kyoto, Japan. pp426-430.

[Kaplan & Kay 94] Ronald Kaplan and Martin Kay. 1994. *Regular Models of Phonological Rule Systems* Computational Linguistics 20(3), pp331-378.

[Kay 73] Martin Kay. 1973. "The MIND system". In R. Rustin (ed.), *Courant Computer Science Symposium 8: Natural Language Processing.* Algorithmic Press, New-York, NY. pp155-188.

[Kay 96] Martin Kay. 1996. "Chart Generation". Proceedings of the *34th Meeting of the Association for Computational Linguistics* ACL'96. pp200-204.

[Nakamura, Tsujii & Nagao 84] Jun-Ichi Nakamura, Jun-Ichi Tsujii, Makoto Nagao (1984). "Grammar Writing System (GRADE) of Mu-Machine Translation Project and its Characteristics" In Proceedings of Coling'84, Stanford University, CA, 2-6 July 1984. pp338-343.

[Oflazer 94] Kemal Oflazer. 1994. "Two-level Description of Turkish Morphology", *Literary and Linguistic Computing* 9/2.

[Stock et al. 88] Oliviero Stock, Rino Falcone and Patrizia Insinnamo (1988). "Island Parsing and Bidirectional Charts". In Proceedings of COLING'88. pp636-641.

[Vauquois & Boitet 85] Bernard Vauquois & Christian Boitet (1985). Automated Translation at Grenoble University. Computational Linguistics 11(1), pp28-36, January-March 1985.

[Wintner 99] Shuly Wintner (1999) "Modularized context-free grammars". In MOL6 - Sixth Meeting on Mathematics of Language, Pages 61-72, Orlando, Florida, July 1999.

[Wintner 97] Shuly Wintner (1997). *An Abstract Machine for Unification Grammars.* PhD Thesis, Technion – Israel Institute of Technology, Haifa, Israel.

[Wintner, Gabrilovitch & Francez 97] Shuly Wintner, Evgeniy Gabrilovitch and Nissim Francez (1997). "AMALIA – A Unified Platform for Parsing and Generation" In Proceedings of RANLP'97, Tzigov Chark, Bulgaria. pp135-142.

[Zajac 92a] Rémi Zajac (1992). *Inheritance and Constraint-based Grammar Formalism.* Computational Linguistics 18(2), pp 159-182.

[Zajac 92b] Rémi Zajac (1992). "Towards Computer-Aided Linguistic Engineering". In Proceedings of Coling'92, Nantes, France. pp828-834.

[Zajac 98a] Rémi Zajac (1998). "Reuse and Integration of NLP Components in the Calypso Architecture". In First International Conference Language Resources and Evaluation. Granada, Spain, May 1998.

[Zajac 98b] Rémi Zajac. 1998. "Feature Structures, Unification and Finite-State Transducers". FSMNLP'98, International Workshop, on Finite State Methods in Natural Language Processing, June 29 - July 1, 1998. Bilkent University, Ankara, Turkey.

[Zajac 99a] Rémi Zajac. 1999. "Graphs and feature structures in a component-based architecture." EPSRC Workshop on NLP Architectures and Language Resources, Baslow, UK, December 1998.

[Zajac 99b] Rḿi Zajac. 1999. "A Multilevel Framework for Incremental Development of MT Systems". Machine Translation Summit VII, National University of Singapore, September 13-17, 1999, Singapore.

[Zajac, Casper & Sharples 97] Rémi Zajac, Mark Casper and Nigel Sharples (1997). "An Open Distributed Architecture for Reuse and Integration of Heterogeneous Components". ANLP'97.

Posters

Hypergraph Unification-Based Parsing for Incremental Speech Processing

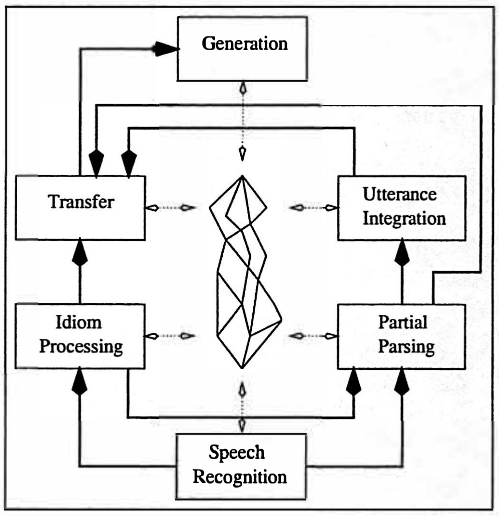
Jan W. Amtrup
Computing Research Laboratory
New Mexico State University
email: `jamtrup@crl.nmsu.edu`

Parsing word graphs is hard. Even though they provide an extremely compact way of representing recognition results (Aubert and Ney, 1995), they still consist of a large number of edges. The ambiguity on the lowest level (words of an utterance) can be high, depending on the experimental setting for the speech recognition system. Parsing word graphs that were produced using an incremental speech recognition system is even harder, since they can be ten times larger than their non-incremental counterparts. However, for highly sophisticated and natural applications, incremental recognition is a sine qua non.

The complexity of incremental word graphs can be partly overcome by converting them to hypergraphs. Here, edges may have several start and end vertices. Following Weber (1995), we call these sets *families of edges.*

Word graphs and their generalizations as hypergraphs are instances of a chart-like structure (Kay, 1980). Similar to standard charts, hypergraphs constructed by inserting word hypotheses are directed, acyclic, rooted graphs. Thus, the hypergraph resulting from this process can be viewed as the lowest level of representation for chart-based speech processing — it relates to the graph that contains only preterminal edges.

We are using a hypergraph as underlying data structure in the speech translation system MILC (*M*achine *I*nterpreting by *L*ayered *C*harts) (Amtrup, 1999). This system was designed as a prototype to demonstrate the feasibility of incremental speech translation and as an experimental platform for the research into the interaction between different processing stages. The figure shows an outline of the architecture. MILC uses typed feature structures with appropriateness for the representation of all linguistic structures in the system. In order to be applicable for a distributed system, we developed an array-based encoding for feature structures (Carpenter and Qu, 1995). This kind of representation allows for an easy exchange of feature structures across machine boundaries.



The structural analysis of hypergraphs constructed incrementally while the parsing processes are already running poses some interesting questions with regard to the consistency of the overall operation of the system. Within such a system, the quality (score) of a piece of information may change over time. This is partly due to further incoming recognition results, and partly due to penalties inflicted upon certain hypotheses by individual parsing processes.

The structural analysis within MILC consists of three components with different purposes and descriptive power. First, idioms are recognized. In our case, idioms are fixed expressions without any inflectional or order variation. This stage performs a fast incremental graph search by intersecting the input hypergraph with a graph describing the known idioms and compounds.

The second component performs an incremental left-to-right parse of an utterance, with an emphasis on the syntactic structure of limited size constituents (Abney, 1991).

The following integration component is implemented as a bidirectional island-parser, capable of extending hypotheses to the left. This is useful, e.g., if the subcategorization frame of a verb is known late due to a verb-last position in an utterance. In our experiments, employing a second parser for incorporating complements to the left resulted in a speedup of ten as compared to a strict left-to-right approach. The principle question with this method, however, is how much of the incrementality is lost using islands due to delayed attachment of elements to the left. In practice, this delay involves only a few calls to the fundamental chart parsing rule, mainly to incorporate complements into verb phrases. In theory, the worst case would arise if the island was the rightmost element of every rule. Suppose that a rule has n elements on the right hand side, then $n - 1$ more elements would have to be incorporated than with the left-to-right approach (the standard approach has to incorporate these as well, but does that before the last element is reached). Thus, at any given point in the chart, the additional effort is bound by the number of elements on the right hand side of a rule, which in turn is bound by the number of words to the left.

The combination of three methods for structurally analyzing an input utterance yields a reasonably high performance, in our tests six-fold real time for a machine translation of spontaneous speech input.

References

Steven Abney. 1991. Parsing By Chunks. . In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-Based Parsing.* Kluwer Academic Publishers, Dordrecht.

Jan W. Amtrup. 1999. *Incremental Speech Translation.* Number 1735 in Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin, Heidelberg, New York.

Xavier Aubert and Hermann Ney. 1995. Large Vocabulary Continuous Speech Recognition Using Word Graphs. In *ICASSP 95.*

Bob Carpenter and Yan Qu. 1995. An Abstract Machine for Attribute-Value Logics. In *Proceedings of the 4th International Workshop on Parsing Technologies (IWPT95)*, pages 59–70, Prague. Charles University.

Martin Kay. 1980. Algorithmic Schemata and Data Structures in Syntactic Processing. Technical Report CSL-80-12, Xerox Palo Alto Research Center, Palo Alto, CA.

Hans Weber. 1995. *LR-inkrementelles, Probabilistisches Chartparsing von Worthypothesengraphen mit Unifikationsgrammatiken: Eine Enge Kopplung von Suche und Analyse.* Ph.D. thesis, Universität Hamburg.

PARSING MILDLY CONTEXT-SENSITIVE RMS

Tilman Becker **Dominik Heckmann**
German Research Center for Artificial Intelligence (DFKI GmbH)
{becker,heckmann}@dfki.de

Abstract
We introduce Recursive Matrix Systems (RMS) which encompass mildly context-sensitive formalisms and present efficient parsing algorithms for linear and context-free variants of RMS. The time-complexities are $O(n^{2h+1})$, and $O(n^{3h})$ respectively, where h is the height of the matrix. It is possible to represent Tree Adjoining Grammars (TAG [1], MC-TAG [2], and R-TAG [3]) as RMS uniformly.

1 Recursive Matrix Systems (RMS)

$RMS = (G, I)$ is a two-step formalism. In a first step, a grammar $G = (N, T, S, P)$ generates a set of recursive matrices. In a second step, a yield function maps, according to the interpretation I, the recursive matrices to a set of strings $L(G, I)$. The recursive matrix generating grammars are ordinary phrase structure grammars, where the terminal symbols are replaced by vertical vectors. These vertical vectors are filled with terminal symbols, nonterminal symbols, or they are left empty.
The two-step formalism: $S \in N \overset{derive}{\Longrightarrow^*_G} m \in \mathcal{M} \overset{yield}{\longmapsto_I} w \in T^*$.
Example: $G_1 = (N, T, S, P) = (\{S\}, \{a, b, c\}, S, \{S \to \begin{array}{|c|}\hline a\\\hline b\\\hline c\\\hline\end{array}\, S\ ,\ S \to \begin{array}{|c|}\hline a\\\hline b\\\hline c\\\hline\end{array}\})$. Every time the first rule is applied, a new column with terminals is added to the matrix. The last rule terminates the process.
A possible derivation: $S \Longrightarrow^1_{G_1} \begin{array}{|c|}\hline a\\\hline b\\\hline c\\\hline\end{array}\, S \Longrightarrow^1_{G_1} \begin{array}{|c|c|}\hline a & a\\\hline b & b\\\hline c & c\\\hline\end{array}\, S \Longrightarrow^1_{G_1} \begin{array}{|c|c|c|}\hline a & a & a\\\hline b & b & b\\\hline c & c & c\\\hline\end{array}$.
The product of the derivation process is a matrix with terminals as elements. Finally, these terminal symbols are combined into one string. There are many possible interpretation functions. However, it seems reasonable to read the terminal symbols within the matrix row by row from top to bottom, and each row alternating from left to right and from right to left. This interpretation condition for height 3 could be visualized by $\begin{smallmatrix}\rightarrow\\ \leftarrow\\ \rightarrow\end{smallmatrix}$. The grammar G_1 together with this interpretation generates strings of the form *aaa...bbb...ccc...*. Thus, the generated language is $L(G_1, \begin{smallmatrix}\rightarrow\\ \leftarrow\\ \rightarrow\end{smallmatrix}) = \{a^n b^n c^n \mid n \in \mathbb{N}\}$.
Definition: h is a positive integer. A *recursive matrix grammar* is a four-tuple $G = (N, T, S, P)$, where N and T are disjoint alphabets, $S \in N$, P is a finite set of ordered pairs $(u, v) \in N \times (N \cup \mathcal{C}_h)^*$, where G is *context-free*, if $P \subset N \times (N \cup \mathcal{C}_h)^*$, *linear*, if $P \subset N \times \mathcal{C}_h N^\epsilon \mathcal{C}_h^\epsilon$, *regular*, if $P \subset N \times \mathcal{C}_h N^\epsilon$.

$$\mathcal{C}_h = \{ \begin{array}{|c|}\hline v_1\\\hline \vdots\\\hline v_h\\\hline\end{array} \mid \forall_{1 \le x \le h} : v_x \in V^\epsilon \}.$$ $\mathcal{C}_h$ is the set of *columns* of height h over $V^\epsilon = N \cup T \cup \{\epsilon\}$.

An *interpretation* $I \in \mathcal{I}_h$ is a vertical vector with left and right arrows as elements, where

$$\mathcal{I}_h = \{ \begin{array}{c} d_1\\ \vdots\\ d_h\end{array} \mid \forall_{1 \le x \le h} : d_x \in \{\rightarrow, \leftarrow\} \}.$$ $\mathcal{I}_h$ is the *set of Interpretations* of height h.

2 Parsing Schemata for Recursive Matrix Systems (RMS)

The notation for the parsing schemata is close to [4]. An Earley parser for RMS can be found in [2].

RMS offer brief vector notations: Let $\vec{i} = \begin{matrix} i_1 \\ \vdots \\ i_h \end{matrix}$, $\vec{i'} = \begin{matrix} i_2 \\ \vdots \\ i_{h+1} \end{matrix}$, $\vec{j} = \begin{matrix} j_1 \\ \vdots \\ j_h \end{matrix}$, $\vec{k} = \begin{matrix} k_1 \\ \vdots \\ k_h \end{matrix}$, $\vec{v} = \begin{array}{|c|}\hline v_1 \\ \hline \vdots \\ \hline v_h \\ \hline\end{array}$.

The set of hypothesis: $\mathcal{H} = \{\begin{array}{|c|c|c|}\hline a & i-1 & i \\ \hline\end{array} \mid a = a_i\ ,\ 1 \le i \le n\} \cup \{\begin{array}{|c|c|c|}\hline \epsilon & i & i \\ \hline\end{array} \mid 0 \le i \le n\}$.

The set of items $\mathcal{J}$ depends on the height h: $\mathcal{J}(h) = \{\begin{array}{|c|c|c|}\hline A & i & j \\ \hline\end{array}, \begin{array}{|c|c|c|}\hline A & \vec{i} & \vec{j} \\ \hline\end{array}\}$.

The parsing schema $\mathbb{P}(CYK)(h\mathcal{RMS}(CFG, \rightrightarrows) \equiv MC-TAG)$ is defined by the deduction rules:

$$D^{init} = \{\begin{array}{|c|c|c|}\hline v_1 & i_1 & k_1 \\ \hline\end{array}, \dots, \begin{array}{|c|c|c|}\hline v_h & i_h & k_h \\ \hline\end{array} \vdash \begin{array}{|c|c|c|}\hline A & \vec{i} & \vec{k} \\ \hline\end{array} \mid (A \to \vec{v}) \in P\}$$

$$D^{combine} = \{\begin{array}{|c|c|c|}\hline B & \vec{i} & \vec{j} \\ \hline\end{array}, \begin{array}{|c|c|c|}\hline C & \vec{j} & \vec{k} \\ \hline\end{array} \vdash \begin{array}{|c|c|c|}\hline A & \vec{i} & \vec{k} \\ \hline\end{array} \mid (A \to BC) \in P\}$$

$$D^{linearize} = \{\begin{array}{|c|c|c|}\hline A & \vec{i} & \vec{i'} \\ \hline\end{array} \vdash \begin{array}{|c|c|c|}\hline A & i_1 & i_{h+1} \\ \hline\end{array}\}$$

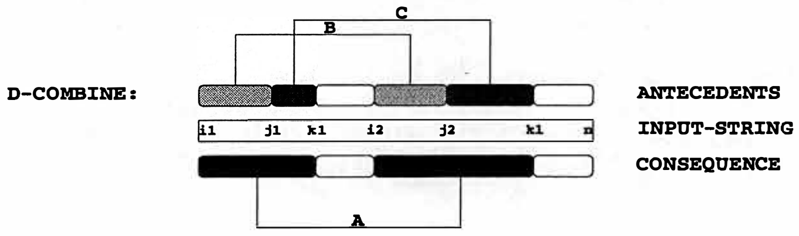
The parsing schema $\mathbb{P}(CYK)(2\mathcal{RMS}(LIN, \rightrightarrows) \equiv R-TAG)$ is defined by the deduction rules:

$$D^{init} = \left\{ \begin{array}{|c|c|c|}\hline v_1 & i_1 & j_2 \\ \hline v_3 & i_3 & j_4 \\ \hline\end{array} \vdash \begin{array}{|c|c|c|}\hline & i_1 & j_2 \\ A & & \\ & i_3 & j_4 \\ \hline\end{array} \mid (A \to \begin{array}{|c|}\hline v_1 \\ \hline v_3 \\ \hline\end{array}) \in P \right\}$$

$$D^{adjoin} = \left\{ \begin{array}{|c|c|c|}\hline v_1 & i_1 & j_1 \\ \hline v_3 & i_3 & j_3 \\ \hline\end{array} \begin{array}{|c|c|c|}\hline & j_1 & i_2 \\ B & & \\ & j_3 & i_4 \\ \hline\end{array} \begin{array}{|c|c|c|}\hline v_2 & i_2 & j_2 \\ \hline v_4 & i_4 & j_4 \\ \hline\end{array} \vdash \begin{array}{|c|c|c|}\hline & i_1 & j_2 \\ A & & \\ & i_3 & j_4 \\ \hline\end{array} \mid (A \to \begin{array}{|c|}\hline v_1 \\ \hline v_3 \\ \hline\end{array} B \begin{array}{|c|}\hline v_2 \\ \hline v_4 \\ \hline\end{array}) \in P \right\}$$

$$D^{linearize} = \left\{ \begin{array}{|c|c|c|}\hline & i_1 & i_2 \\ A & & \\ & i_2 & i_3 \\ \hline\end{array} \vdash \begin{array}{|c|c|c|}\hline A & i_1 & i_3 \\ \hline\end{array} \right\}$$

The time–complexity of the algorithms can be derived from the number of relevant indices, where each vector $\vec{i}$ represents h indices. Step $D^{combine}$ in algorithm $\mathbb{P}(CYK)(h\mathcal{RMS}(CFG, \rightrightarrows))$ has three relevant vectors $\vec{i}, \vec{j}, \vec{k}$, which each represent h indices, resulting in a time–complexity of $O(n^{3h})$. Step $D^{linearize}$ has a $O(n^{h+1})$ time-complexity. The correctness is inherited from the CFG cases. The following figure explains the variables of the deduction rule $D^{Combine}$ for $\mathbb{P}(CYK)(2\mathcal{RMS}(CFG, \rightrightarrows)$:



References

[1] Tilman Becker and Dominik Heckmann. 1998. Recursive Matrix Systems (RMS) and TAG. *Proceedings of the Fourth International Workshop on TAG (TAG+4),Philadelphia, PA.*

[2] Dominik Heckmann. 1999. Recursive Matrix Systems. *Diploma Thesis, University of Saarland.*

[3] Giorgio Satta and William Schuler. 1998. Restrictions on Tree Adjoining Languages. *Proceedings of COLING-ACL'98*, pages 1176–1182.

[4] Klaas Sikkel and Anton Nijhold. 1997. Parsing of Context–Free Languages. *Handbook of Formal Languages, Vol. 2, Springer, Berlin.*

PROPERTY GRAMMARS: A SOLUTION FOR PARSING WITH CONSTRAINTS

Philippe Blache
LPL – Université de Provence, 29 Av. Robert Schuman, 13621 Aix-en-Provence, France
pb@lpl.univ-aix.fr

1. Situation: Guidelines for an Actual Constraint-Based Approach

Modern linguistic theories often make use of the notion of *constraint* to represent information. This notion allows a fine-grained representation of information, a clear distinction between linguistic objects and their properties, and a better declarativity. Several works try to take advantage of a constraint-based implementation (see for example [Maruyama90], [Carpenter95], [Duchier99]). However, the parsing process cannot be interpreted as an actual constraint satisfaction one. This problem mainly comes from the generative conception of grammars on the linguistic analysis. Indeed, in constraint-based theories, constraints can appear at different levels: lexical entries, grammar rules, universal principles. However, during a parse, one has first to select a local tree and then to verify that this tree satisfies the different contraints. This problem comes from the generative interpretation of the relation between grammars and languages. In this case, the notion of derivation is central and parsing consists in finding a derivation generating an input. We propose then a new formalism called *Property Grammars* representing the linguistic information by means of constraints. These constraints constituting an actual system, it becomes possible to consider parsing as a satisfaction process.

An optimal use of constraints should follow some requirements. In particular, all linguistic information has to be represented by means of constraints. This information constitutes a system of constraints, then all the constraints are at the same level and the order of verification of the constraints is not relevant. Encapsulation is another important characteristics which stipulates that a constraint must represent homogeneous information.

The last important point concerns the notion of grammaticality. In the particular problem of parsing, finding an exact solution consists in associating a syntactic structure to a given input. In the case of generative approaches, this amounts to finding a derivation from a distinguished symbol to this input. However, the question when parsing real natural language inputs should not be the grammaticality, but the possibility of providing some information about the input. We propose then to replace the notion of grammaticality with that of *characterization* which is much more general: a characterization is the state of the constraint system for a given input.

2. Property Grammars

Property Grammars (cf. [Blache99]) provide a framework implementing these requirements: declarativity, encapsulation, satisfiability. We use for the representation of syntactic information 7 properties defined as follows :

• **Constituency** (noted *const*)	Set of categories constituting a phrase.
• **Obligation** (noted *oblig*)	Set of compulsory, unique categories (heads).
• **Unicity** (noted *uniq*)	Set of categories which cannot be repeated in a phrase.
• **Requirement** (noted $\Rightarrow$)	Cooccurrency between sets of categories.
• **Exclusion** (noted $\not\Leftrightarrow$)	Restriction of cooccurrence between sets of categories.
• **Linearity** (noted $\prec$)	Linear precedence constraints.
• **Dependency** (noted $\rightsquigarrow$)	Dependency relations between categories.

It is interesting to notice that properties can be expressed over sets of categories, allowing then to represent contextual information.

3. Parsing as Constraint Satisfaction

The parsing process (1) takes as input the set of elementary trees (in fact *unary quai trees*, as described in [Blache98]) that can be associated to the sentence and (2) builds all the characterizations of this input. A *characterization* is then defined over a set of categories by P^+ (the set of satisfied properties) and P^-(the set of unsatisfied properties). The following example describes a grammar of the *NP* in french.

Properties of the *NP*: (1) *Const* = {*Det, N, AP, Sup*} (2) *Oblig* = {*N, AP*} (3) *N[com]* $\Rightarrow$ *Det* (4) *Det* $\prec$ *N* (5) *Det* $\prec$ *AP* (6) *Det* $\prec$ *Sup* (8) *N* $\prec$ *Sup* (9) *AP* $\not\Leftrightarrow$ *Sup* (10) *Det* $\rightsquigarrow$ *N* (11) *AP* $\rightsquigarrow$ *N* (12) *Sup* $\rightsquigarrow$ *N*
Properties of the *AP*: (1) *Const* = {*Adj, Adv*} (2) *Oblig* = {*Adj*} (3) *Adv* $\prec$ *Adj* (4) *Adv* $\rightsquigarrow$ *Adj*
Properties of the *Sup* (superlative): (1) *Const* = {*Det, Adv, Adj*} (2) *Oblig* = {*Adj*} (3) *Adj* $\Rightarrow$ *Det* (4) *Adj* $\Rightarrow$ *Adv* (5) *Det* $\prec$ *Adv* (6) *Det* $\prec$ *Adv* (7) *Adv* $\prec$ *Adj* (8) *Det* $\rightsquigarrow$ *Adj* (9) *Adv* $\rightsquigarrow$ *Adj*

The following example presents the elementary trees associated to the words of the noun phrase *le livre le plus rare (the rarest book)*.

NP		*NP*	*AP*	*AP*
Sup	*NP*	Sup	Sup	Sup
|	|	|	|	|
Det	*N*	*Det*	*Adv*	*Adj*
|	|	|	|	|
le	*livre*	*le*	*plus*	*rare*
(1)	(2)	(3)	(4)	(5)

The general approach consists in characterizing all subsets of categories belonging to the set of elementary trees, whatever their level. Considering all the possible subsets of categories has an exponential cost and some simple controls can be applied. In the following, we will only take into account the sets of juxtaposed categories. Building the characterizations consists for each subset of categories of verifying the satisfiability of the properties.

The following example illustrates the construction of such characterizations for the grammar presented above. The properties are represented by their indexes. For example, the characterization of the set $\{Det_1/N_2\}$ indicates that all the properties specified for these categories (i.e. constituency, obligation, requirement, precedence and dependency) are satisfied. The set $\{Adv_4/AP_5\}$ presents two characterizations: this sequence can characterize an *AP* (all the properties of the grammar are verified) and a *Sup* in which a requirement property is not satisfied.

Characterization of 2-uples:	- Det_1/N_2 :	$P^+(NP) = \{1, 2, 3, 4, 10\}$;	$P^-(NP) = \emptyset$
	- Adv_4/AP_5 :	$P^+(AP) = \{1, 2, 3, 4\}$;	$P^-(AP) = \emptyset$
		$P^+(Sup) = \{1, 2, 4, 9\}$;	$P^-(Sup) = \{3\}$
	...		
Characterization of 3-uples:	- $Det_1/N_2/Sup_3$:	$P^+(NP) = \{1, 2, 3, 4, 6, 8, 9, 10, 12\}$;	$P^-(NP) = \emptyset$
	- $Sup_1/N_2/Det_3$:	$P^+(NP) = \{1, 2, 3, 9, 10, 12\}$;	$P^-(NP) = \{4, 6, 8\}$
	...		
Characterization of 4-uples	- $Det_1/N_2/Sup_3/AP_4$:	$P^+(NP) = \{1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12\}$;	$P^-(NP) = \{9\}$
	...		

Characterizations are associated to a given set of categories without taking into consideration the characterization of each of these categories. However, the information contained by the set of characterizations for a given input allow the reconstitution of a hierarchical structure. The mechanism consists in finding, if possible, a characterization covering the input, then finding characterizations for the embedded constituents. The following example describes such a process. For clarity, only positive characterizations are examined here.

Among the characterizations given in the figure (2), 4 are positive:

- Det_1/N_2 :	$P^+(NP) = \{1, 2, 3, 4, 10\}$;	$P^-(NP) = \emptyset$
- Adv_4/Adj_5 :	$P^+(AP) = \{1, 2, 3, 4\}$;	$P^-(AP) = \emptyset$
- $Det_1/N_2/Sup_3$:	$P^+(NP) = \{1, 2, 3, 4, 8, 9, 10, 12\}$;	$P^-(NP) = \emptyset$
- $Det_3/Adv_4/Adj_5$:	$P^+(Sup) = \{1, 2, 3, 4, 5, 6, 8, 9\}$;	$P^-(NP) = \emptyset$

There is in this case only one solution covering all the input (using only positive characterization): the set of categories $Det_1/N_2/Sup_3$ and $Det_3/Adv_4/Adj_5$ which characterize respectively a *NP* and a *Sup*.

A naive implantation of these mechanisms is obviously not efficient. Indeed, let m be the number of words of the input, c the maximum number of categorizations for each word, then the number n of categories to analyze is bounded by $n = 2mc$. Then, the number of sets of categories to analyse is bounded by 2^n.

Several controls can be added in order (1) to reduce the number of sets to analyze and (2) to control the satisfaction process. We can for example choose to build only sets of juxtaposed categories or consider only sets actually characterized by properties. The satisfaction process itself can be controlled using several heuristics: it is possible to filter the satisfiability according to a threshold given by the cardinality of the set of unsatisfied properties P^-: it is possible to build only characterizations with less than a certain amount of unsatisfied constraints. At the extreme, we can reduce the satisfiability to positive characterizations (reducing characterization to grammaticality).

4. Conclusion

The representation of linguistic information by means of constraints is interesting both for knowledge representation (no implicit information, encapsulation) and for implementation point of view (parsing is seen as a constraint satisfaction process). Property Grammars offer a framework taking advantage of these characteristics and present several interests for natural language processing. The first one concerns the generality and the robustness of the approach: constraints allow the introduction of the notion of characterization to replace that of grammaticality. The parsing process consists in verifying constraints for the possible set of categories associated to the input instead of trying to build a structure according to the grammar. One other interesting point concerns the integration of different information sources: the properties can concern all kind of information (prosodic, syntactic, pragmatiç etc.).

References

[Archangeli97] Archangeli D. & D.T. Langendoen eds. (1997) *Optimality Theory*, Blackwell.

[Blache98] BLACHE P. (1998) "Parsing Ambiguous Structures using Controlled Disjunctions and Unary Quasi-Trees." in proceedings of *ACL-COLING'98*.

[Blache99] Blache P. (1999) *Filtering and Fusion: A Technique for Parsing with Properties*, in proceedings of NLPRS'99.

[Carpenter95] Carpenter B. & Penn G. (1995) "Compiling Typed Attribute-Value Logic Grammars", in H. Bunt and M. Tomita (eds.), *Current Issues in Parsing Technologies*, Kluwer.

[Duchier99] Duchier D. & Thater S. (1999) "Parsing with Tree Descriptions: a constraint based approach", in proceedings of NLULP'99.

[Maruyama90] Maruyama H. (1990), "Structural Disambiguation with Constraint Propagation", in proceedings of *ACL'90*.

[Shieber92] Shieber S. (1992) *Constraint-Based Grammar Formalisms*, MIT Press.

Grammar Organization for Cascade-Based Parsing in Information Extraction

Fabio Ciravegna **Alberto Lavelli**
ITC-irst Centro per la Ricerca Scientifica e Tecnologica
via Sommarive 18, 38050 Povo (TN) ITALY
{cirave|lavelli}@irst.itc.it

Recently there has been an increasing interest in finite-state techniques for NLP. Finite-state transducers have been used for a number of NLP tasks, from morphological analysis and shallow parsing to semantic processing. In [3] we have proposed a parsing approach for Information Extraction (IE) based on finite-state cascades. The parser aims at building a syntactic representation equivalent (from an IE point of view) to the ones built by systems based on full parsing. The approach is inspired by the work of Abney on finite-state cascades for parsing unrestricted text [1] and provides: (i) a method for efficiently and effectively performing full parsing on texts for IE purposes; (ii) a way of organizing generic grammars that simplifies changes, insertion of new rules and integration of domain-oriented rules. Parsing is based on the application of a fixed sequence of cascades of rules. It is performed deterministically in three steps: (1) chunking; (2) clause recognition and nesting; (3) modifier attachment.

The approach has been developed as part of LE-FACILE, a successfully completed EU project for multilingual text classification and IE [4]. The approach is currently used in Pinocchio, an environment for developing and running IE applications [2]. The proposed approach has been tested mainly for Italian, but proved to work also for English and partially for Russian.

Due to space limitations, in this paper we concentrate exclusively on the issues connected with grammar organization. For further details on the adopted formalism, the parsing approach, and some experimental results, see [3].

Rules are grouped into **cascades** that are finite, ordered sequences of rules. Cascades represent elementary logical units, in the sense that all the rules in a cascade deal with some specific construction (e.g., subcategorization of verbs). From a functional point of view a cascade is composed of three segments: s**1** contains rules that deal with idiosyncratic cases for the construction at hand; s**2** contains rules dealing with the regular cases; s**3** contains default rules that fire only when no other rule can be successfully applied.

The initial generic grammars for chunking and clause recognition are designed to cover the most frequent phenomena in a restrictive sense. Additional rules can be added to the grammar (when necessary) for coping with the uncovered phenomena, especially domain-specific idiosyncratic forms. The limited size of the grammar makes modifications simple (e.g., our clause recognition grammar for Italian is composed of 66 rules).

The deterministic approach combined with the use of cascades and segments makes grammar modifications simple, as changes in a cascade (e.g., rule addition/modification) influence only the following part of the cascade or the following cascades. This makes the writing and debugging of grammars easier than in approaches based on context-free grammars, where changes to a rule can in principle influence the application of any rule in the grammar.

The grammar organization in cascades and segments allows a clean definition of the grammar parts. Each cascade copes with a specific phenomenon (modularity of the grammar). All the rules for the specific phenomenon are grouped together and are easy to check.

The segment structure of cascades is suitable for coping with the idiosyncratic phenomena of restricted corpora. As a matter of fact domain-oriented corpora can differ from the standard use of language (such as those found in generic corpora) in two ways: (i) in the frequency of the constructions for a specific phenomenon; (ii) in presenting different (idiosyncratic) constructions. Coping with different frequency distributions is conceptually easy by using deterministic parsing and cascades of rules, as it is sufficient to change the rule order within the cascade coping with the specific phenomenon, so that more frequently applied rules are first in the cascade. Coping with idiosyncratic constructions requires the addition of new rules.

Finally from the point of view of grammar organization, defining segments brings some added value to ordered cascades. Generic rules (in s2) are separated from domain specific ones (in s1); rules covering standard situations (in s2) are separated from recovery rules (in s3). In s2, rules are generic and deal with unmarked cases. In principle s2 and s3 are units portable across the applications without changes. Domain-dependent rules are grouped together in s1 and are the resources the application developer works on for adapting the grammar to the specific corpus needs (e.g., coping with idiosyncratic cases). Such rules generally use contexts and/or introduce domain-dependent (semantic) constraints in order to limit their application to well defined cases. S1 rules are applied before the standard rules and then idiosyncratic constructions have precedence with respect to standard forms.

Segments also help in parsing robustly. S3 deals with unexpected situations, i.e. cases that could prevent the parser from continuing. For example the presence of unknown words is coped with after chunking by a cascade trying to guess the word's lexical class. If every strategy fails, a recovery rule includes the unknown word in the immediately preceding chunk so to let the parser continue. Recovery rules are applied only when rules in s1 and s2 do not fire.

References

[1] Abney S. Partial parsing via finite-state cascades. In *Proceedings of the ESSLI '96 Robust Parsing Workshop*, 1996. Also available at the URL `http://www.sfs.nphil.uni-tuebingen.de/~abney/Papers.html`.

[2] Ciravegna F. and A. Lavelli. The Pinocchio Information Extraction Toolkit. `http://ecate.itc.it:1024/projects/pinocchio.html`.

[3] Ciravegna F. and A. Lavelli. Full text parsing using cascades of rules: An information extraction perspective. In *Proceedings of EACL99*, Bergen, Norway, 1999.

[4] Ciravegna, F., Lavelli, A., Mana, M., Gilardoni, L., Mazza, S., Ferraro, M., Matiasek, J., Black, W. J., Rinaldi, F. and D. Mowatt. FACILE: Classifying texts integrating pattern matching and information extraction. In *Proceedings of IJCAI99*, Stockholm, Sweden, 1999.

A BIDIRECTIONAL BOTTOM-UP PARSER FOR TAG [*]

Víctor J. Díaz **Vicente Carrillo**
Dpto. de Lenguajes y Sistemas Informáticos, Universidad de Sevilla
Avda. Reina Mercedes s/n, 41012 Sevilla (Spain)
{vjdiaz,carrillo}@lsi.us.es

Miguel A. Alonso
Departamento de Computación, Universidad de La Coruña
Campus de Elviña s/n, 15071 La Coruña (Spain)
alonso@dc.fi.udc.es

1 Introduction

Several parsing algorithms have been proposed for Tree Adjoining Grammars (TAGs), most of them derived from context-free tabular parsers [1]. However, only a few bidirectional parsing algorithms have been proposed [3, 5] although this kind of strategy seems to be naturally adapted for TAG. In this paper, we present a new bidirectional bottom-up parser for TAG derived from the context-free parser defined by de Vreught and Honig [4], which has several interesting characteristics: (i) the bidirectional strategy allow us to implement the recognition of the adjoining operation in a simple way, (ii) the bottom-up behavior allow us to take advantage of lexicalization, reducing the number of trees under consideration during the parsing process, and (iii) in the case of ungrammatical input sentences, the parser is able to recover most of partial parsings according to lexical entries.

2 The parser

A Tree Adjoining Grammar is a 5-tuple $\mathcal{G} = (V_N, V_T, S, \boldsymbol{I}, \boldsymbol{A})$, where V_N and V_T are finite sets of non-terminal and terminal symbols, S the axiom of the grammar, and $\boldsymbol{I}$ and $\boldsymbol{A}$ are finite sets of *initial trees* and *auxiliary trees*, respectively. $\boldsymbol{I} \cup \boldsymbol{A}$ is the set of *elementary trees*. The reader is referred to [2] for details in the definition and properties of TAGs. Given an elementary tree γ, a node N^γ and its g children $N_1^\gamma \ldots N_g^\gamma$ are represented by a production $N^\gamma \rightarrow N_1^\gamma \ldots N_g^\gamma$. $\mathcal{P}(\gamma)$ is the set of productions formed in such a way for every node in γ. Terminal symbols and ε will be represented in the productions with the proper symbol. Also, we consider additional productions: $\top \rightarrow \mathbf{R}^\alpha$, $\top \rightarrow \mathbf{R}^\beta$ and $\mathbf{F}^\beta \rightarrow \bot$ for each initial tree α and each auxiliary tree β, where $\mathbf{R}^\alpha$ is the root node of α and $\mathbf{R}^\beta$ and $\mathbf{F}^\beta$ are the root node and foot node of β, respectively. The nodes $\top$ and $\bot$ are considered no-adjoining nodes.

We will define the parser using the *Parsing Schemata* framework [4]. To this end, we assume the reader is familiar with this kind of parser specification. Let be the input string $w = a_1 \ldots a_n$ with

This work was partially supported by the FEDER of EU (project 1FD97-0047-C04-02) and Xunta de Galicia (projects PGIDT99XI10502B and XUGA20402B97).

$n \geq 0$, the parser works with items $[N^\gamma \to \nu \bullet \delta \bullet \omega, i, j, p, q]$, where $N^\gamma \to \nu\delta\omega \in \mathcal{P}(\gamma)$ is decorated with two dots indicating the part, δ, of the subtree dominated by N^γ that has been recognized. When ν and ω are both empty, the whole subtree has been recognized. The two indices i and j indicate that part of w spanned by δ. If $\gamma \in \boldsymbol{A}$, the indices p and q indicate that part of w spanned by the foot node of γ. In other case $p = q = -$, representing they are undefined.

The deduction steps of the parser are the following:

$$\mathcal{D}^{\text{Ini}} = \frac{[a, j-1, j]}{[N^\gamma \to \nu \bullet a \bullet \omega, j-1, j, -, -]} \qquad \mathcal{D}^{\text{Con}} = \frac{\begin{array}{c}[N^\gamma \to \nu \bullet \delta_1 \bullet \delta_2\omega, i, j', p, q] \\ [N^\gamma \to \nu\delta_1 \bullet \delta_2 \bullet \omega, j', j, p', q']\end{array}}{[N^\gamma \to \nu \bullet \delta_1\delta_2 \bullet \omega, i, j, p \cup p', q \cup q']}$$

$$\mathcal{D}^{\epsilon} = \frac{}{[N^\gamma \to \bullet\bullet, j, j, -, -]} \qquad \mathcal{D}^{\text{Foot}} = \frac{[M^\gamma \to \bullet\delta\bullet, k, l, p, q]}{[\mathbf{F}^\beta \to \bullet\bot\bullet, k, l, k, l]}$$

$$\mathcal{D}^{\text{Inc}} = \frac{[M^\gamma \to \bullet\delta\bullet, i, j, p, q]}{[N^\gamma \to \nu \bullet M^\gamma \bullet \omega, i, j, p, q]} \qquad \mathcal{D}^{\text{Adj}} = \frac{\begin{array}{c}[\top \to \bullet\mathbf{R}^\beta\bullet, j, m, k, l] \\ [M^\gamma \to \bullet\delta\bullet, k, l, p, q]\end{array}}{[N^\beta \to \nu \bullet M^\gamma \bullet \omega, j, m, p, q]}$$

The steps $\mathcal{D}^{\text{Init}}$ and $\mathcal{D}^\epsilon$ are the *initializer* deduction steps in charge of starting the parsing process. The steps $\mathcal{D}^{\text{Inc}}$ and $\mathcal{D}^{\text{Con}}$ traverse elementary trees when no adjoining is performed. Finally, given $\beta \in \text{adj}(M^\gamma)$, the steps $\mathcal{D}^{\text{Foot}}$ and $\mathcal{D}^{\text{Adj}}$ are used to recognize the adjoining operation, where $p \cup q$ is equal to p if q is undefined and is equal to q if p is undefined, being undefined in other case.

Given $\alpha \in \boldsymbol{I}$, the input string w belongs to the language defined by the grammar when an item $[\top \to \bullet\mathbf{R}^\alpha\bullet, 0, n, -, -]$ is deduced. From the set of derived items (the chart), a parser of the input can be constructed retracing the recognition steps in reverse order or annotating the items computed by the recognition algorithm with information about how they were obtained.

The time complexity of the algorithm with respect to the length n of the input is $O(n^6)$. This complexity is due to adjoining step since presents the maximum number of relevant input variables (j, m, k, l, p, q). With respect to the size of the grammar, the time-complexity is $O(|G|^2)$, since at most two elementary trees are simultaneously considered in a single step. The space-complexity of the parser is $O(n^4)$ since every item is composed of four input positions ranging on the length of the input string.

References

[1] Alonso, M. A., D. Cabrero, E. de la Clergerie and M. Vilares. 1999. Tabular algorithms for TAG parsing. *Proc. of EACL'99*, pages 150–157, Bergen, Norway.

[2] Joshi, A. K. and Y. Schabes. 1997. Tree-adjoining grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages. Vol 3: Beyond Words*, chapter 2, pages 69–123. Springer-Verlag, Berlin/Heidelberg/New York.

[3] Lavelli, A. and G. Satta. 1991. Bidirectional parsing of lexicalized tree adjoining grammars. *Proc. of EACL'91*, Berlin, Germany.

[4] Sikkel, K. 1997. Parsing Schemata — A Framework for Specification and Analysis of Parsing Algorithms. Springer-Verlag, Berlin/Heidelberg/New York.

[5] van Noord, G. 1994. Head-corner parsing for TAG. *Computational Intelligence*, 10(4):525–534.

A FINITE-STATE PARSER WITH DEPENDENCY STRUCTURE OUTPUT

David Elworthy
Canon Research Centre Europe Ltd., Occam Road, Guildford GU2 5YJ, UK
dahe@acm.org

Abstract
We show how to augment a finite-state grammar with annotations which allow dependency structures to be extracted. There are some difficulties in determinising the grammar, which is an essential step for computational efficiency, but they can be overcome. The parser also allows syntactically ambiguous structures to be packed into a single representation[1].

1 A finite-state parser with dependency output

Finite-state parsing and dependency grammars both offer advantages to implementers of practical NLP systems. Finite-state parsers have good computational properties, typically $O(n^2)$ in the number of words if the finite-state machines are determinised. Finite-state grammars can also be partly modularised by using cascading (Abney, 1996), and can be implemented so as to tolerate ungrammatical input. Dependency grammars, on the other hand, allow a pseudo-semantic analysis, by extracting relationships such as modification between words. In general, dependency grammars can be implemented with $O(n^3)$ complexity, although ungrammaticality and certain constructs can lead to NP complexity (Neuhaus and Bröker, 1997).

We have developed a simple technique for making a finite-state parser produce dependency structures. As well as outputting the phrase bracketing, the parser also stores words in "variables", in response to annotations in the grammar rules. Variables may be unindexed, storing lists of words, or indexed, providing a mapping from words to lists of words. Indexed variables stand for dependency relations; unindexed ones contain non-dependent elements such as the overall head. Each variable has a name, corresponding to the type of dependency relationship. Thus the dependencies for *a big brown dog barked at the man* could be represented as

```
head = barked; agent[barked] = dog; nomMod[dog] = big, brown
spec[dog] = a; pmod[barked] = at; pcomp[at] = man; spec[man] = the
```

Variable assignments are established by augmenting the rules of the finite-state grammar with annotations which indicate the variable into which the corresponding words are to be stored. In the case of indexed variables, the annotation also includes the name of an unindexed variable, indicating the word at which the dependency link begins. Variables can be set from a lexical item or from an existing variable. Here is an example grammar:

```
NP(nhead, spec[], nomMod[])
     = Det:spec{nhead} (Adj:nomMod{nhead})* (N:nomMod{nhead})* N:nhead
VP(vhead, patient[], spec[], nomMod[])
      = V:vhead NP:(patient{vhead}=head,^)?
S(head, agent[], patient[], spec[], nomMod[])
```

[1]A full version of this paper can be obtained from http://research.microsoft.com/~davidelw/. Author's current address: Microsoft Research Ltd., St. George House, 1 Guildhall Street, Cambridge CB2 3NH, UK. Thanks to Aaron Kotcheff, Tony Rose and Amanda Clare.

```
      = NP:(agent{head}=nhead,^) VP:head=vhead^
```

The rules can be glossed as follows. In the first rule, for noun phrases, the final noun is stored in the *nhead* variable, the determiner in the specifier (*spec*) of the head, and the adjectives and nouns as modifiers of the head (*nomMod*). The verb phrase rule stores the verb in the *vhead* variable of the VP. If a noun phrase object is present, its head is copied to *patient*, indexed on the VP's head, thus making a dependency link between the verb and its object. $\wedge$ is shorthand for copying all other variables (*spec*, *mod*) up to the VP. The sentence rule links the subject NP as the agent of the verb, and copies the other dependencies and the head of the VP to the sentence's variables[2].

1.1 Implementation

Most aspects of the implementation are straightforward. We wait until a phrase has been recognised, and then set the unindexed variables followed by the indexed ones. This is implemented by maintaining an agenda of what variables need to be set, during execution of the finite-state machines.

In order to obtain $O(n^2)$ complexity, the finite-state machines must be deterministic. Standard algorithms for determinising finite-state machines will not work here, since the transitions are not uniqely determined by the input symbols. Thus, in `Det:spec (N:nomMod{spec})* N:head`, the `N` input can correspond to different variable outputs. The algorithm described by Roche and Schabes (1995) will solve this in most cases, by deferring output symbols are deferred until enough right-hand context is accumulated to decide which output to produce.

2 Ambiguity and packing

Determinising the finite-state machines only allows a single output, even if the input is ambiguous. Lexical ambiguity can be controlled by a pre-processing step such as tagging, but structural ambiguity is more of a problem. However, the use of variables to represent dependency allows a scheme in which structural ambiguities are packed into a single structure. This will not solve all problems to do with structural ambiguity, but two of the more common ones in English – PP attachment and noun compounding – can easily be handled this way.

To construct packed dependency structures, we place a word in the range (value) of more than one indexed variable, so that it can be accessed via multiple dependencies. The grammar below shows how this is done for prepositional phrase attachment ambiguity.

```
NP(nhead) = Det N:nhead
PP(prep, pcomp[], phead) = Prep:prep NP:(pcomp{prep}=nhead, phead=nhead)
PP(prep, pcomp[], phead) = PP:^ PP:(^,pmod{phead}=prep)
NP(nhead, pmod[], pcomp[]) = NP:^ (PP:pmod{nhead}=prep,pcomp=pcomp,pmod=pmod)*
```

The first PP rule builds simple prepositional phrases, in which *phead* provides access to the head of embedded NP, while *pcomp* provides access to it via the preposition. In the second PP rule, a *pmod* link from this *phead* is then made to the preposition of the second PP. When the phrase *the man in the park on a hill* is parsed, *on a hill* is attached in the final structure to both *man* and *park*. A similar approach can be used for noun compounds.

References

Abney, Steven (1996). Partial Parsing via Finite-State Cascades. In *Proceedings of the ESSLLI '96 Robust Parsing Workshop*. Also available from `http://www.sfs.nphil.uni-tuebingen.de/~abney/`.

Neuhaus, Peter and Norbert Bröker (1997). The Complexity of Recognition of Linguistically Adequate Dependency Grammars. In *Proceedings of the 35th ACL and 8th EACL*, pp. 337–343.

Roche, Emmanuel and Yves Schabes (1995). Deterministic Part-of-Speech Tagging with Finite-State Transducers. *Computational Linguistics*, volume 21, number 2, pp. 227–253.

[2]We are assuming, for the sake of this example, that the subject realises the agent role and the object the patient role

DISCRIMINANT REVERSE LR PARSING OF CONTEXT-FREE GRAMMARS

Jacques Farré
Laboratoire I3S – CNRS and Université de Nice - Sophia Antipolis
jf@essi.fr

1 Introduction

In *Discriminant Reverse LR(k)* parsing[1], actions are determined by scanning a stack *suffix*, thanks to a small deterministic finite automaton (dfa). Thus, opposite to (direct) LR parsers, DRLR parsers do not need the whole content of the stack in order to determine actions. DRLR parsers can, in few cases, deeply explore the stack, but it has been proven they are linear in time.

When the grammar is not of the required class, this feature of DRLR parsers allows to defer actions in conflict and to push a *conflict mark* onto the stack. The DRLR parser can keep on parsing the input right of the conflict, that is to say it can perform shifts *and reductions* as long as these actions are determined by a stack suffix above the mark. Otherwise, the action cannot be chosen and an *induced* conflict occurs; a new mark is pushed, and parsing of the remaining input can resume.

For most grammars, the stack configurations in which marks resolve the LR conflict, and how to resolve it, are computed at construction. For other grammars, the conflict is resolved at parsing time by comparing the stack content with the candidate derivations (which are given by the marks), but the stack configurations in which marks trigger these comparisons are computed at construction.

One can argue that conflict marks are an implementation of multiple stacks. But the stack is not explored below a conflict mark, and Extended DRLR needs a single stack and a single parser. Thus, it is not comparable to GLR. In particular, Extended DRLR parsers for which conflicts resolution can be computed at construction are linear in time and size.

2 Principles of (Extended) DRLR construction

This presentation will rely on a DRLR(0) parser. In a DRLR(0) item $[i, A \to \alpha {\bullet} \beta]$, i is a parsing action (shift if i =3D 0, reduce according tth $i$production if $i > 0$); the core $A \to \alpha {\bullet} \beta$ means that a stack suffix σ has been scanned, and $\beta \overset{*}{\underset{rm}{\Rightarrow}} \sigma x$. The states of the DRLR automaton are the states of the *dfa* which recognizes these σ's by reading them from right to left. If $[i, A \to \alpha X {\bullet} \beta]$ belongs to a state q and if no $[i', A' \to \alpha' X {\bullet} \beta']$, $i' \neq$3D i, belongs to q, the suffix $X\sigma$ determines action i; else more stack must be read, and a transition by X to q', which will contain $[i, A \to \alpha {\bullet} X\beta]$, is done; $[i, A \to {\bullet} \alpha]$ infers (closure computation) $[i, B \to \alpha' {\bullet} A\beta']$, that is a step back in a rightmost derivation chain.

A LR(0) conflict occurs when distinct rigthmost derivations produce a same prefix ϕ. These derivations (or *proto*-derivations when ϕ belongs to a language $\alpha\beta^*\gamma$) can be computed from the states of the DRLR(0) automaton. Each step of these derivations defines *mark positions.* For example, let

$$S \overset{*}{\underset{rm}{\Rightarrow}} \pi Ax \underset{rm}{\Rightarrow} \pi\alpha B\beta x \overset{*}{\underset{rm}{\Rightarrow}} \pi\alpha Byx \underset{rm}{\Rightarrow} \pi\alpha\gamma C\delta yx \overset{*}{\underset{rm}{\Rightarrow}} \pi\alpha\gamma Czyx \underset{rm}{\Rightarrow} \phi zyx$$
$$S \overset{*}{\underset{rm}{\Rightarrow}} \pi'A'x' \underset{rm}{\Rightarrow} \pi'\alpha'B'\beta'x' \overset{*}{\underset{rm}{\Rightarrow}} \pi'\alpha'B'y'x' \underset{rm}{\Rightarrow} \pi'\alpha'\gamma'C'\delta'y'x' \overset{*}{\underset{rm}{\Rightarrow}} \pi'\alpha'\gamma'C'z'y'x' \underset{rm}{\Rightarrow} \phi z'y'x'$$

be two derivations in conflict: a mark is pushed onto stack after ϕ, and it will be found instead of C or C', but only the C and C' of $B \xrightarrow{i} \gamma C\delta$ and $B' \xrightarrow{i'} \gamma'C'\delta'$. The cores $B \to \gamma C{\bullet}\delta$ and $B' \to \gamma'C'{\bullet}\delta'$ will be the mark positions.

If the languages of δ and δ' are disjoint and not prefix, z and z' can be reduced respectively to δ and δ' without scanning the stack until the mark. As, in this case, δ ≠3D, $\delta[i, B \to \gamma C{\bullet}\delta]$ and $[i', B' \to \gamma'C'{\bullet}\delta']$ cannot be in the same state, and the mark can determine action i or action i'. Choosing i or i' means the LR(0) conflict must be resolved by reducing respectively to C or to C'.

But, if for example δ =3D, $\{i, B \to \gamma C{\bullet}\delta]$ and $[i', B' \to \gamma'C'{\bullet}\delta']$ will be in a same state. The mark cannot decide between i and i'. The new mark for this induced conflict can be found in stack instead of B or B', but only in positions $A \to \alpha B{\bullet}\beta$ for reduction to C and $A' \to \alpha'B'{\bullet}\beta'$ for reduction to C'. The LR conflict can be resolved if the languages of β and β' are discriminant.

Conflicts cannot be resolved at construction if a mark has the positions $A \to \alpha B{\bullet}\beta$ and $A \to \alpha'B'{\bullet}\beta$, which mean that $\pi\alpha B \overset{*}{\Rightarrow} w$, $\pi'\alpha'B' \overset{*}{\Rightarrow} w$. This does not mean that the grammar is ambiguous, as shown by the example below.

3 A working example

The example will rely on the grammar $S \xrightarrow{2} aAb$, $S \xrightarrow{3} aBbb$, $A \xrightarrow{4} aAb$, $A \xrightarrow{5} c$, $B \xrightarrow{6} aBbb$, $B \xrightarrow{7} c$, which produces the non deterministic language $a^ncb^n \cup a^ncb^{2n}, n > 0$. DRLR(0) augments the initial grammar with $S' \xrightarrow{1} \vdash S \dashv$.

Reductions 5 and 7 are in conflict. A mark m_0 with positions $S \to aA{\bullet}b$ and $A \to aA{\bullet}b$ for 5, with positions $S \to aB{\bullet}bb$ and $B \to aB{\bullet}bb$ for 7, is pushed onto the stack. It decides a shift for any of its position in q_0, but it produces an induced conflict between $[2, S \to aA{\bullet}b]$, $[4, A \to aA{\bullet}b]$, $[0, S \to aB{\bullet}bb]$ and $[0, B \to aB{\bullet}bb]$ in some q_i for the stack suffix m_0b.

The new mark m_1 has positions $S' \to \vdash S{\bullet} \dashv$ for action 2, $S \to aA{\bullet}b$ and $A \to aA{\bullet}b$ for 4, $S \to aBb{\bullet}b$ and $B \to aBb{\bullet}b$ for 0. It resolves the LR conflict in favor of 5 with the stack suffix $m_1 \dashv$, and it produces another induced conflict between $[2, S \to aA{\bullet}b]$, $[4, A \to aA{\bullet}b]$, $[3, S \to aBb{\bullet}b]$ and $[6, B \to aBb{\bullet}b]$.

The mark m_2 has positions $S' \to \vdash S{\bullet} \dashv$ for actions 2 and 3, $S \to aA{\bullet}b$ and $A \to aA{\bullet}b$ for 4, $S \to aB{\bullet}bb$ and $B \to aB{\bullet}bb$ for 6. When the stack suffix is $m_2 \dashv$, the conflict cannot be resolved at construction. Otherwise (suffix =3D$_2$b), m_2 has the same positions as m_0, and pushes m_1.

The mark m_2 means that an even number of b has been read, while m_1 means that an odd number of b has been read. The suffix $m_2 \dashv$ decides a conflict resolution at parsing time, while $m_1 \dashv$ can decide at construction to resolve the conflict in favor of 5. But if we have $S \xrightarrow{2} aAba$ instead of $S \xrightarrow{2} aAb$, the conflict resolution can be decided at construction in any case : the position $S' \to \vdash S{\bullet} \dashv$ of m_2 is no more compatible with the reduction in conflict 5, and it can decide the reduction 7.

References

[1] Fortes-Gálvez, J. 1996. A practical small LR parser with action decision through minimal stack suffix scanning. in *Developments in Language Theory II*, World Scientific.

Direct Parsing of Schema–TAGs*

K. Harbusch & J. Woch
University of Koblenz–Landau, Computer Science Department
E–mail: {harbusch|woch}@uni-koblenz.de

Schema–Tree Adjoining Grammars (S–TAGs) are an extension of *Tree Adjoining Grammars (TAGs)* (for an introduction see, e.g., [5]), initially introduced by [7]. In a *schematic elementary tree*, a *regular expression (RX)*[1], is annotated at each inner node of an elementary tree. This means, that the elementary schemata enumerate a possibly infinite set of elementary trees. For instance, see the tree t_1 of Fig. 1, which performs NP–coordination as, e.g., the zero–coordination *Peter* (|2|) or *Bob, Bill, Mary, Sue, and the dog* ($|2|^4.|3|.|1|.|2|$). Obviously, the schemata of S–TAGs provide a condensed

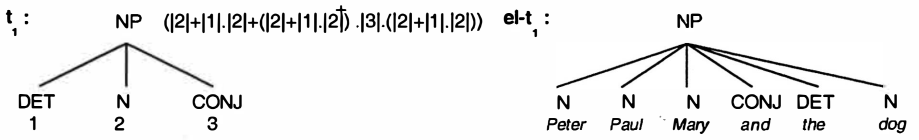


Figure 1: NP-Coordination

grammar representation. Concerning the structures, TAGs and S–TAGs are identical and hence all advantages of TAGs are conserved such as the extended domain of locality and the mild context–sensitivity. Thus S–TAGs are well suited for linguistic specifications.

In order to analyse S–TAGs, an ordinary TAG parser, e.g., the Earley–based parser by Yves Schabes ([6]), can only be applied if the length of the input string is considered in order to compute a finite subset of applicable elementary trees. This is disadvantegeous because the computation must be performed at runtime and the property of condensed representations is lost. Therefore we introduce a *direct parser of S–TAGs* which works directly on the schemata. Accordingly, common prefixes (i.e. substructures) of the same scheme are represented only once instead of enumerating all explicit orderings in the individual elementary trees. This leads to a better average runtime. However, the worst case time complexity remains the same as for TAGs ($O(n^6)$). The direct parser adapts the basic idea of an Earley parser [2] to the exploration of regular expressions. This means, that a new dot position ($\odot$) is defined which indicates whether a prefix of an alternative in a regular expression is analysed. In order to move the $\odot$, two operations SHIFT and NEXT are defined. $SHIFT(\phi)$ moves $\odot$ by one token to the right, i.e. $SHIFT(\alpha \odot \beta) = \{\gamma | \gamma = \alpha NEXT(\beta) \odot (\beta \setminus NEXT(\beta))\}$ where $NEXT(\phi)$ returns a set of all alternatives for the next symbol (Gorn address). $NEXT(\alpha) = \{\odot\alpha\}$ iff α is a Gorn address; $NEXT(\alpha) = \{\odot\beta_1, ..., \odot\beta_n\}$, iff $\alpha = (\beta_1 + ... + \beta_n)$, $n \geq 2$ and $NEXT$ is applied recursively as long as β_i starts with an opening bracket, finally a Gorn address is reached[2]; $\{\odot\beta^+.\gamma,$

*This work is partially funded by the DFG — German Research Foundation — under grant HA 2716/1-1 and 1-2.

[1]RXs are inductively defined. Let α, β and $\beta_1, ..., \beta_n$ ($n \geq 2$) be Gorn addresses uniquely referring to daughters or arbirarily complex RXs, then $\alpha.\beta$ (concatenation of branches), $(\beta_1 + ... + \beta_n)$ (enumeration of alternatives), α^* (Kleene Star) and α^+ (α^* without the empty repetition) are RXs. Finally, "–" allows to cut off a subtree at the end of a path. As an abbreviation we write $\alpha^{(n|m)}$ which enumerates $\sum_{i=0}^{m} \alpha^{n+i}$ ($n, m \geq 0$). Notice, "." binds stronger than "+".

[2]Notice sums are always bracketed even if no binding conflict occurs as on the top level (cf. footnote 1).

$\beta^*.\odot\gamma\}$, iff $\alpha = \beta^*.\gamma$; $\{\odot\beta.\beta^*.\gamma\}$, iff $\alpha = \beta^+.\gamma$. Here also $NEXT$ is recursively computed as long as β (and γ for $\beta^*.\odot\gamma$) starts with an opening bracket in order to reach the Gorn addresses to be finally returned. Traversing the regular expressions replaces the analysis of branches in the six procedures of the TAG parser by Schabes in a straight–forward manner. For more details of the extension of the individual procedures of the Schabes parser see [3] and [8].

With the close relation to ordinary TAGs and the claim that direct parsing is advantageous, the question arises how can existing TAGs made available to the direct parser? Obviously, an arbitrary TAG G can trivially be transformed into an S–TAG G' by annotating the concatenation of all daughters from left to right at each inner node of each elementary tree. This transformation involves no compression and accordingly, no benefit of a direct S–TAG parser arises. Therefore, we have developed a transformation component which produces a S–TAG where each label at the root node occurs only once in the set of initial and auxiliary trees. The following steps lead to this goal: Firstly, in all elementary trees all subtrees which do not contain the foot node are rewritten by substitution in order to find shared structures. Some reformulations become necessary to prevent the grammar from overgeneration because existing substitution trees with the same label would become applicable unintentionally. Accordingly, the corresponding trees are renamed. Now, all alternatives with the same root node are collected and described by a regular expression denoting the set of all these trees. Finally, the resulting RXs are condensed by applying the distributivity law ($\alpha\gamma^{(l|k)}\gamma\beta = \alpha\gamma^{(l|k+1)}\beta$, $\alpha(\gamma\delta_1)\beta + ... + \alpha(\gamma\delta_m)\beta = \alpha\gamma(\delta_1 + ... + \delta_m)\beta$ and $\alpha\gamma\beta + \alpha\beta = \alpha\gamma^{(0|1)}\beta$ where $\alpha, \beta, \gamma, \delta_1, ..., \delta_m$ are arbitrary complex RXs). This procedure has been applied to the XTAG system [1] and the transformed grammar serves a syntax grammar for analysis and generation as we run our direct parser as a *reversible uniform generation modul* (cf. [3] and [4]). Currently we transform existing semantic and pragmatic knowledge sources into the S–TAG formalism.

References

[1] C. Doran, D. Egedi, B. Hockey, B. Srinivas & M. Zaidel. XTAG System — A Wide Coverage Grammar for English. In M. Nagao, ed., *Proceedings of the 15th International Conference on Computational Linguistics*, vol. 2, pp. 922–928, Kyoto, Japan, 1994.

[2] J. Earley. An Efficient Context–Free Parsing Algorithm. *Communications of the Association for Computing Machinery*, 13(2):94–102, 1970.

[3] K. Harbusch, F. Widmann & J. Woch. Towards a Workbench for Schema-TAGs. In A. Abeillé, T. Becker, O. Rambow, G. Satta & K. Vijay-Shanker, eds., *Procs. of the 4th International TAG+ Workshop*, pp. 56 – 61, Philadelphia, PA, USA, 1998. IRCS-Report 98–12.

[4] K. Harbusch, F. Widmann & J. Woch. Ein reversibles Analyse–/Generierungsmodul für Schema-Tree Adjoining Grammars. In C. Habel & T. Pechmann, eds., *Sprachproduktion.* Westdeutscher Verlag, Wiesbaden, Germany, 2000.

[5] A. K. Joshi & Y. Schabes. Tree Adjoinung Grammars. In G. Rozenberg, A. Salomaa, eds., Handbook of Formal Languages, Vol. 3, pp. 69–214, 1997.

[6] Y. Schabes. *Mathematical and Computational Aspects of Lexicalized Grammars.* Ph.D. thesis, University of Pennsylvania, Philadelphia, PA, USA, 1990.

[7] D. Weir. Characterising Mildly Context–Sensitive Grammar Formalisms. PhD. Proposal, University of Pennsylvania, Philadelphia, USA, 1987.

[8] J. Woch & F. Widmann. Implementation of a Schema-TAG-Parser. Fachberichte Informatik 8–99, Universität Koblenz, Koblenz, Germany, 1999.

ANALYSIS OF EQUATION STRUCTURE USING LEAST COST PARSING

R. Nigel Horspool **John Aycock**

Department of Computer Science, University of Victoria
P.O. Box 3055, Victoria, BC, Canada V8W 3P6

{nigelh,aycock}@csr.uvic.ca

Abstract

Mathematical equations in LaTeX are composed with tags that express formatting as opposed to structure. For conversion from LaTeX to other word-processing systems, the structure of each equation must be inferred. We show how a form of least cost parsing used with a very general and ambiguous grammar may be used to select an appropriate structure for a LaTeX equation. MathML provides another application for the same technology; it has two alternative tagging schemes – presentation tags to specify formatting and content tags to specify structure. While conversion from content tagging to presentation tagging is straightforward, the converse is not. Our implementation of least cost parsing is based on Earley's algorithm.

1 Introduction

LaTeX [1] is a widely used system for typesetting where the user embeds markup tags in a text document to control the formatting. Some tags are *structure tags* because they denote the logical structure of some part of the document (e.g. \section or \footnote are structure tags). Others are *presentation tags* because they specify formatting without necessarily imparting any information about the document structure. For example, \it and \tiny are presentation tags. HTML is another example of a markup language where structure tags (e.g. <ul> and <h2>) can be mixed with presentation tags (e.g. <font> and <center>) in the same document.

For reformatting a document according to new style guidelines or for importing into some word processing systems, it is necessary to know the full structure of the document. Conversion of a LaTeX document into an Adobe FrameMaker document is an example where full structural information is needed. We have developed such a conversion tool, a FrameMaker import filter named *laimport*. However, conversion of equations in LaTeX documents has been problematic and motivated the research reported here. The research also has applications to MathML [2] where both presentation and content tagging schemes are provided.

2 A 'Cost-Based' Grammar for LaTeX Equation Notation

The grammar must be able to accept any sequence of mathematical symbols, even nonsensical ones. But when a conventional structure exists, it should be inferred. We accomplish this goal by attaching cost expressions to grammar rules, where high costs are associated with rules when parentheses do not balance ,operator symbols lack operands, and so on. The grammar also takes account of hints in the form of spacing – operators surrounded by white spaceare assumed to have lower precedence than operators with less surrounding white space. A grammar for a subset of LaTeX equations is shown in Figure 1. In this grammar, '~' represents an inserted space. Each non-terminal has a synthesized cost attribute, computed by the cost expression attached to the grammar rule. These costs increase if parentheses do not balance or if '*' is used with lower precedence than '+'.

With the example grammar, the LaTeX equation $a+b~*~c$ has parses equivalent to ${a+{b~*~c}}$ with cost 14780 and ${{a+b}~*~c}$ with cost 9873.5. The whitespace has overridden the usual operator precedences, as is desirable when handling operators whose meanings are unknown to the translator. Similarly, the grammar provides a parse for $((a$ while rejecting a parse of $(a+b)$ as equivalent to ${(a}+{b)}$.

S	→	E	%cost{$1}
E	→	E OP E	%cost{$2*($1 + 1.5*$3)}
E	→	(E)	%cost{$2}
E	→	(E	%cost{$2*2.0}
E	→	E)	%cost{$1*2.0}
E	→	symbol	%cost{1.0}
E	→	OP	%cost(200.0)
OP	→	~ OP	%cost{0.7*$2}
OP	→	OP ~	%cost{0.7*$1}
OP	→	+	%cost{80.0}
OP	→	*	%cost{100.0}

Figure 1: A Partial Grammar with Cost Expressions

3 Extending Earley's Parsing Algorithm

The problem of finding a least cost parse is similar to that of finding the most probable parse. Our implementation is based upon Stolcke's extension of Earley's algorithm to probabilistic parsing [3]. It is necessary to restrict our cost expressions to be non-negative monotonically non-decreasing functions of their inputs; this permits a pruning strategy that removes items from the parser's states.

The essence of the parsing method is to extend each item in an Earley parser state with two extra components.An item, therefore, has this structure:

$$[\, A \rightarrow \alpha \bullet \beta \,;\, @n \,;\, \overline{C} \,;\, @r \,]$$

The *n* component refers back to the ancestor state where the parser started matching the rule $A \rightarrow \alpha\beta$. The $\overline{C}$ component is a vector of costs, one per symbol in α. The *r* component is called the *least cost predecessor*. It records the identity of a predecessor item *in the same state* which generated this particular item.

The completer step of the parser is responsible for computing the cost of the completed rule based on the input values in $\overline{C}$. It must then consider whether to add a new item to the current state where the marker has been advanced past the non-terminal A. If a similar item (one with the same rule, same dot position and same ancestor state) already exists in the current state, the monotonicity property of the ancestor rule's cost expression can frequently be used to choose only one of the two similar items to retain. The least cost predecessor in the new item is linked to the item containing the completed rule, and may be used to construct the least cost parse after all the input has been processed.

The worst-case time complexity of the parsing method is exponential. We note, however, that if the grammar can be transformed so that each righthand side contains no more than two non-terminals, the pruning strategy would be totally effective and the parsing efficiency will be the same as Earley's method. Such a transformation would come at the cost of reducing the expressivity of the cost formulae however.

Acknowledgements

Financial support was received from the Natural Sciences and Engineering Research Council of Canada.

References

[1] L. Lamport. *LATEX: A Document Preparation System*. 2nd Edition. Addison-Wesley, 1994.

[2] Mathematical Markup Language (MathML™) 1.01 Specification, 7 July 1999. URL: `http://www.w3.org/TR/REC-MathML`.

[3] A. Stolcke. An Efficient Probabilistic Context-Free Parsing Algorithm that Computes Prefix Probabilities. *Computational Linguistics* 21, 2 (1995), pp. 165-201.

EXPLOITING PARALLELISM IN UNIFICATION-BASED PARSING

M. P. van Lohuizen
Dept. of Information Technology and Systems - PDS
Delft University of Technology
Delft, The Netherlands
mpvl@acm.org

Abstract
Because of the nature of the parsing problem, unification-based parsers are hard to parallelize. We present a parallelization technique designed to cope with these difficulties.

Parallel parsing of natural language has been researched extensively. In [6] we can find an overview of parallel chart parsing. Most attempts, however, were not very successful [8, 3]. Only recently two NLP applications were successfully parallelized [7, 5]. However, the former focussed on Prolog and the latter exploits coarse-grained parallelism of the kind that proved unusable for our Deltra system[1] or other systems [3]. We present a more widely applicable approach, not limited to Prolog.

Most unification-based parsers have characteristics that make them particularly hard to parallelize. Typically, unifications account for the bulk of the processing time in unification-based parsing. However, parallelizing this operation is difficult [1] and does not speed up the CF part. Therefore, most research has focussed on exploiting parallelism at the CF level, where the unification operations are atomic and distributed amongst processors. This approach has several problems as well. First, each item has a different impact on the derivation of new items. In addition, the computational cost of unifying the items in one parse can vary wildly. These irregularities make it hard to find a good distribution of work (*load balancing*). Another characteristic is the lack of *locality*, causing excessive communication and ineffective cache utilization. Finally, because all intermediate results need to be recorded in a chart, there is a lot of *synchronization* between processors. This aspect is aggravated by the need of dynamic load balancing, which requires additional synchronization.

The implementation is aimed at *shared-memory* architectures, mainly driven by the difficulties discussed before. The increasing availability of these systems further justifies this choice. Let us first consider the problem of load balancing. Experiments in [10] indicate that only dynamic load balancing of single unification operations can yield a scalable solution to parallel parsing. However, putting each individual unification in a central queue will result in too much overhead. We solve this dilemma by taking a *work stealing* approach:[2] each processor—or thread[3]—has its own work queue. However, whenever a processor runs out of work, it may steal work from other processors. This considerably decreases the amount of synchronization because now synchronization is only required when a processor is stealing work. We further limit the number of steals by allowing a thread to steal an amount of work proportional to the amount of work available at the victim.

[1]The Deltra system comprises a medium sized grammar and dictionary for Dutch.
[2]The work-stealing approach is similar to the Cilk-5 system [2], but differs in its optimization for chart parsers.
[3]There is one thread for each processor. Threads are automatically distributed amongst processors by the OS.

As mentioned before, synchronization is also required to store results in the chart. Letting threads wait unconditionally for exclusive access incurs too much contention. We solve this problem by splitting all tasks in distributed and sequential parts. Instead of contending for chart access, a thread simply queues the sequential work[4] and proceeds with other work. The scheduler [9] ensures that at most one processor is processing sequential work, automatically providing synchronization.

A final optimization improves on the locality. Our parser allows synchronization of the chart per grammar rule. By associating each grammar rule—and its data—with a single processor, we can improve caching behavior. This does not compromise on the ability to balance load, because assigned work may still be stolen. In general, the scheduler has been designed to move most overhead to the stealing side, leaving minimal overhead for normal operation.

Experiments confirmed that work stealing indeed yields a well-balanced distribution of work. On a 4-thread setup, 6% of the total time was consumed by idling and 6% could be attributed to scheduling overhead.[5] Work stealing proved to be necessary, as omitting it yielded 20% idling for a 2-thread setup (rapidly increasing for more processors). The solution to reduce synchronization proved effective as well. Only 1/2000 to 1/5000 synchronizations per unification were required on a 2-processor run. The ratio of running time of the sequential version and the 1-processor parallel version $T_1//_sT$was 1.03, yielding minimal overhead. Running a 2-processor version on a dual Pentium-II yielded a speedup of 1.7 (1.4 for a Pentium-Pro). Experiments verified that the gap in speedup between this result and the load balancing simulation was caused by communication of the (hardware) cache coherency protocol. This indicates that the bottleneck is caused by ineffective cache usage. Currently our research is focusing on improving cache utilization to mitigate the usage of memory bandwidth.

In conclusion, the speedup that can be obtained depends on the specific grammar being used. Nevertheless, the presented scheduling method allows for the finest possible grain of parallelism, without resorting to parallel unification. In addition, although originally designed for the parallelization of chart parsers, the presented technique can be applied to any tabulation programming technique.

References

[1] C. Dwork et. al.. On the sequential nature of unification. *J. of Logic Programming*, 1(1):35–50, 1984.

[2] M. Frigo, C.E. Leiserson, and K.H. Randall. The implementation of the Cilk-5 multithreaded language. *ACM SIGPLAN Notices*, 33(5):212–223, May 1998.

[3] G. Görz, et. al. Research on architectures for integrated speech/language systems in Verbmobil. In *The 16th COLING*, vol. 1, pp. 484–489, Copenhagen, DK, Aug 5–9 1996.

[4] H.J. Honig. A new double dotted parsing algorithm. TR 94-108, Delft University of Technology, 1994.

[5] A.G. Manousopoulou et. al. Automatic generation of portable parallel natural language parsers. In *Proc. of the 9th Conf. on Tools with Artificial Intelligence (ICTAI '97)*, pp. 174–177. IEEE CS Press, 1997.

[6] A. Nijholt. Overview of parallel parsing strategies. In M. Tomita, ed. *Current Issues in Parsing Technology.* Kluwer Academic Publishers, Norwell, MA, 1991.

[7] E. Pontelli, G. Gupta, J. Wiebe, and D. Farwell. Natural language multiprocessing: A case study. In *Proc. of the 15th Nat. Conf. on Artificial Intelligence (AAAI '98)*, July 1998.

[8] H.S. Thompson. Parallel parsers for context-free grammars–two actual implementations compared. In G. Adriaens et. al., ed. *Parallel Natural Language Processing.* Ablex Publ. Corp., Norwood, NJ, 1994.

[9] M.P. van Lohuizen. Parallel processing of natural language parsers. In *PARCO '99*, 1999.

[10] M.P. van Lohuizen. Simulating communication of parallel unification-based parsers. PDS Report Series PDS-1998-008, Delft University of Technology, 1998.

[4]We use a queuing mechanism that does not requires locking.

[5]This result was obtained from a multi-threaded run on 1 processor to abstract from cache coherency effects.

PARTIAL PARSING WITH GRAMMATICAL FEATURES

A. G. Manousopoulou, G. Papakonstantinou and P. Tsanakas
Computer Systems Laboratory, National Technical University of Athens
Iroon Polytechneioy 9, 15773, Athens, Greece
e-mail: natasha@cslab.ece.ntua.gr

Abstract

This paper describes a rule based method for partial parsing, particularly for noun phrase recognition, which has been used in the development of a noun phrase recognizer for Modern Greek. This technique is based on a cascade of finite state machines, adding to them a characteristic very crucial in the parsing of words with free word order: the simultaneous examination of part of speech and grammatical feature information, which are deemed equally important during the parsing procedure, in contrast with other methodologies.

1 Introduction

Partial parsing is an area of natural language processing that has been widely explored in the past years with rule based and statistical methodologies. The rule based techniques usually employ pattern matching techniques [1], special grammars such as Constraint Grammars [2] or finite state cascades [3, 4]. The statistical techniques, on the other hand, mainly use Hidden Markov Models, such as the one introduced by Church [5], which is an extension of his well known statistical part of speech tagging methodology. There are also hybrid techniques that combine rules and statistics [6].

In this paper we describe a rule based method for partial parsing, particularly for noun phrase recognition. This technique is based on a cascade of finite state machines, adding to them a characteristic very crucial in the parsing of words with free word order: the simultaneous examination of part of speech and grammatical feature information, which are deemed equally important during the parsing procedure, in contrast with previously employed methodologies.

2 Grammatical Features in Free Word Order Languages

The noun phrase in Modern Greek consists of various parts of speech; it can be elementary or composite, containing recursively other noun phrases or secondary clauses. In this paper we will concentrate on simple, non recursive noun phrases. Modern Greek is a language with free order of the sentence constituents; this also holds partially for the words inside the noun phrase. The possible part of speech combinations are numerous, although there are some general rules that apply to any noun phrase [7].

Due to the freedom in the order of words inside a noun phrase the patterns that describe it are very flexible. An important parameter in the correct determination of noun phrase boundaries are thus the grammatical feature values. A change in feature values from one word to its next usually denotes a phrase boundary, which cannot be generally detected in another manner merely from the POS information. A noun recognizer - or a partial parser in general - for such a language should examine part of speech information together with grammatical features and not just use the features for later verification.

3 Finite State Transducers with Constraints

A very common way of constructing multi-level rule based partial parsers is that of combining finite state transducers (FSTs), which generally operate on the POS tag of the word and cannot take grammatical features into account in a straightforward manner. To solve this problem, we can use parameterization of FSTs, creating finite state machines that manage a limited amount of internal data. These data represent grammatical features and will be used to verify constraints during the transitions of the machine.

Features are added to an FST as a finite list of feature sets $A = (a_1, a_2, \ldots, a_m)$ attached to symbols of the input and output alphabet of the trabsducer. Each set a_i contains the possible values of the i-th feature. The input of an FST with constraints is a string of the form:

$$x_1 \left\{T_1^1, T_1^2, \ldots, T_1^{k_1}\right\} x_2 \left\{T_2^1, T_2^2, \ldots, T_2^{k_2}\right\} \ldots\ x_n \left\{T_n^1, T_n^2, \ldots, T_n^{k_n}\right\}$$

Each tuple T_i^j is a member of the cartesian product $a_1 \times a_2 \times \ldots \times a_m$ and represents a possible value combination for the features of the symbol x_i.

Transitions in FSTs with constraints accumulate intersections of tuple sets $\left\{T_i^1, T_i^2, \ldots, T_i^{k_i}\right.$. If the accumulated intersection is empty, the transition fails. In this way, we can ensure feature agreement in noun phrases.

4 The Partial Parser for Greek

The model of FSTs with constraints has been employed in the creation of a partial parser for Modern Greek as a three level FST cascade. The first level locates and groups consecutive adjectives, as well as other auxiliary word groups, in order to make the task of subsequent levels easier. The second level performs the major part of the noun phrase recognition task. It resolves all types of noun phrases, without incorporating clitic forms of pronouns. This is due to the curious role of clitics in the noun phrase of Modern Greek. This level incorporates "stray" clitics (i.e., ones that are after a noun phrase and do not constitute a noun phrase by themselves) into the preceding noun phrases.

In the input text stream there are two types of ambiguities to be resolved: at the tag level and at the feature level. Resolution of a tag ambiguity takes place when a pattern matches only one of the tags assigned to the word. Ambiguities at the feature level happen when a symbol is accompanied by more than one feature tuples. These ambiguities are resolved during the transitions, since the FST actually calculates the intersection of tuple sets for consecutive symbols inside the sentence. The result of the intersection contains the disambiguated values of the features for both the word group recognized by the FST and the words themselves.

Each transducer of the parser has been implemented as a lex [8] program, and translated with flex, the implementation of the tool available for many platforms. The partial parser has been incorporated and used in a machine learning application for the identification of named entities in Greek business texts [9].

References

[1] Bourigault, D. 1992. Surface grammatical analysis for the extraction of terminological noun phrases. In *Proc. of COLING-'92*, pages 977-981.

[2] Voutilainen, A. 1993. NPTool, a detector of English noun phrases. In *Proc. of the Workshop on Very Large Corpora*, pages 48-57.

[3] Abney, S. 1996. Part-of-speech tagging and partial parsing. In Ken Church, Steve Young and Gerrit Bloothooft, eds. *Corpus-based methods in Language and Speech*, Kluwer Academic Publishers, Dordrecht.

[4] Abney, S. 1996. Partial parsing via finite-state cascades. In *Proc. of the ESSLLI '96 Robust Parsing Workshop.*

[5] Church, K. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proc. of the 2nd Conference on Applied Natural Language Processing*, Austin, Texas.

[6] Chen, Kuang-hua and Hsin-His Chen. 1994. Extracting noun phrases from large-scale texts: A hybrid approach and its automatic evaluation. In *Proc. of the ACL94.*

[7] Mackridge, P.. 1990. The Modern Greek Language (Greek translation). Patakis editions.

[8] Lesk, M. E. 1975. Lex - A Lexical Analyzer Generator. *CompSci. Tech.Rep. No.39*, Bell Laboratories, New Jersey.

[9] Karkaletsis, V., et al. 1999. Named-Entity Recognition from Greek and English Texts. *Journal of Intelligent and Robotic Systems*, 23(5):123-135.

UNIQUELY PARSABLE ACCEPTING GRAMMAR SYSTEMS

Carlos Martín-Vide
Research Group in Mathematical Linguistics and Language Engineering
Rovira i Virgili University
Pça. Imperial Tàrraco 1, 43005 Tarragona, Spain
cmv@astor.urv.es

Victor Mitrana
University of Bucharest, Faculty of Mathematics
Str. Academiei 14, 70109, Bucharest, Romania
mitrana@funinf.math.unibuc.ro

Extended Abstract

Cooperating distributed grammar systems (CDGS, for short) have been introduced independently in [3] as a grammatical approach of the so-called "blackboard model" in the problem solving theory [8] and [2], with motivations coming from regulated rewriting area. Most of the results known in this field until the middle of 1992 can be found in [4], while more recent results are surveyed in [5]. Constructing parsers in the grammar systems set-up is a topic, which is not only of theoretical interest, but it will make grammar systems more appealing to researchers in applied computer science. This will clearly bring to the user all the advantages of having a model which can cope with such phenomena as cooperation and distribution of the work carried out by several processors.

In [6], Mihalache and Mitrana study the effect on CD grammar systems of some syntactical constraints similar to those considered for strict deterministic context-free grammars. They obtained a promising result in this respect, namely the *unambiguity* of the derivations holds for some classes of grammar systems as well.

We belive that a more involved study of the derivations in a CD grammar system would be very useful to a possible parser constructor for the languages generated by grammar systems. This is the aim of the present paper: a new class of accepting devices called uniquely parsable accepting grammars systems (UPAGS, for short) is introduced. These mechanisms have a restricted type of accepting rules such that parsing can be done without backtracking. Each component of a UPAGS is a so-called RC-uniquely parsable grammar [7] viewed as an accepting grammar. In [7] one provided a hierarchy of uniquely parsable grammars that gave a simple grammatical characterization of the deterministic counterpart of the classical Chomsky hierarchy.

When extending the restrictions for unique parsability to accepting grammar systems, two variants should be taken into consideration, depending on the level, *local/global*, to which the restrictions address. In the global level case, where the restrictions apply to all rules of the system consiered altogether as a single set, the accepting capacity of the systems equals the accepting power of deterministic pushdown automata. In other words, the systems collapse to RC-uniquely parsable grammars. When local level is considered, where conditions apply to all rules of each component independently, for some more restrictive classes, one gets more computational power keeping the parsability without backtracking. We give a simple recognition algorithm for these systems.

Our algorithm has two distinct phases: one in which we find the unique component which is to become active and the other one in which a t-leftmost reduction is performed by this component. The former phase is based on the algorithm proposed in [1] for solving the multiple pattern matching problem. The only difference is that the searching process is stopped as soon as a matching pattern has been found in the text. The text is the sentential form while the dictionary of matching patterns is formed by the words in the lefthand side of all rules of the given uniquely parsable grammar system.

For the latter phase we define a procedure which provides the word obtained by a t-leftmost reduction of the current sentential form in the component found in the first phase. Then, this process is resumed. A criterion to detect infinite loops in the derivation process is also presented. When no reduction is possible anymore, we check the sentential form and decide whether or not the input word is accepted.

The exact complexity of this algorithm is briefly discussed.

Acknowledgements

The work done by the second author has been supported by the Dirección General de Enseñanza Superior e Investigación Cientifica, SB 97–00110508.

References

[1] A. V. Aho and M. J. Corasick. 1975. Efficient string matching: an aid to bibliographic search. *Commun. ACM* 18:333–340.

[2] A. Atanasiu, V. Mitrana. 1989. The modular grammars. *Internat. J. Comp. Math.* 30:17–35.

[3] E. Csuhaj-Varju and J. Dassow. 1990. On cooperating distributed grammar systems. *J. Inform. Process. Cybern., EIK*, 26:49 – 63.

[4] E. Csuhaj-Varju, J. Dassow, J. Kelemen and Gh. Păun. 1993. *Grammar Systems* Gordon and Breach, 1993.

[5] J. Dassow, Gh. Păun and G. Rozenberg. 1997. Grammar Systems. In *The Handbook of Formal Languages*, (G. Rozenberg and A. Salomaa, eds.), Springer-Verlag, 3 volumes.

[6] V. Mihalache and V. Mitrana. 1997. Deterministic cooperating distributed grammar systems. In *New Trends in Formal Languages. Control, Cooperation, Combinatorics. LNCS 1218* (Gh. Păun, A. Salomaa eds.), 137–149.

[7] K. Morita, N. Nishihara, Y. Yamamoto and Z. Zhang. 1997. A hierarchy of uniquely parsable grammars and deterministic acceptors. *Acta Informatica* 34:389–410.

[8] P. H. Nii. 1989. Blackboard systems. In *The Handbook of AI*, vol. 4 (A. Barr, P. R. Cohen, E. A. Feigenbaum, eds.), Addison-Wesley, 1989.

CHART PARSING AS CONSTRAINT PROPAGATION

Frank Morawietz
Universität Tübingen,
Wilhelmstr. 113,
72074 Tübingen
frank@sfs.nphil.uni-tuebingen.de

The parsing-as-deduction approach proposed in Pereira and Warren [6] and extended in Shieber et al. [7] and the parsing schemata defined as special deduction systems in Sikkel [8] are well established parsing paradigms. They allow for a uniform presentation and comparison of a variety of parsing algorithms. Furthermore, they are extensible to more complex formalisms, e.g., augmented phrase structure rules or the ID/LP format. Constraint Programming has been used in computational linguistics in several areas, for example in (typed) feature-based systems based on Oz [9], or conditional constraints [5], or advanced compilation techniques [3] or specialized constraint solvers [4]. But to my knowledge, none of these approaches uses constraint programming techniques to implement standard chart parsing algorithms directly in a constraint system.

The core idea of the proposal is that the items of a conventional chart parser are constraints on labeled links between the words and positions of an input string. The inference rules allow for the deduction of new constraints, again labeled and spanning parts of the input string, via constraint propagation. The resulting constraint store represents the chart which can be accessed to determine whether the parse was successful or to reconstruct a parse tree. While this may seem a trivial observation it allows for a rapid and flexible method of implementation. The goal is not necessarily to build the fastest parser, but rather to build – for an arbitrary algorithm – a parser fast and perspicuously. The advantage of our approach compared to the one proposed in Shieber et al. [7] is that we do not have to design a special deduction engine, we do not have to handle chart and agenda explicitly. and that it can be seamlessly integrated into existing applications (e.g., within the Constraint Handling Rules (CHR) framework [2]).

Assuming familiarity with parsing-as-deduction, I will presuppose the definitions as given in Shieber et al. [7]. Their algorithms are implemented by specifying the inference rules as constraint propagation rules, the axioms and the goal items as constraints. As an example, Earley's algorithm is presented in Tab. 1. We cannot go into the details of the definitions, but it is easily observable that the given constraint propagation rules are nothing but literal realizations of the inference rules. And the only other ingredient we need for the specification of a parser is a predicate which traverses the input and posts the corresponding initial edge constraints. This predicate is universal to all algorithms, but can be varied with respect to the order the constraints are posted, thereby achieving for example a left-to-right or right-to-left traversal of the string.

So, the similarity between parsing-as-deduction and constraint propagation is used to propose a flexible and simple system which is easy to implement and offers itself as a testbed for different

Table 1: Parsing systems as CHR programs: Earley's algorithm

Items	Axiom	Goal
`edge(A,Alpha,Beta,I,J)`	`edge(sprime,[],[s],0,0)`	`edge(sprime,[s],[],0,Len)`

```
Scan      edge(A,Alpha,[Cat|Beta],I,J), word(J,Cat-_Word) ==>
                    J1 is J+1,
                    edge(A,[Cat|Alpha],Beta,I,J1).

Predict   edge(_A,_Alpha,[B|_Beta],_I,J) ==>
                    findall(Gamma, rule(Gamma,B), List) |
                    post_edges(List,B,J).

Complete  edge(A,Alpha,[B|Beta],I,K), edge(B,Gamma,[],K,J) ==>
                    edge(A,[B|Alpha],Beta,I,J).
```

parsing strategies with varying traversals of the input and even for different types of grammars (such as for example minimalist grammars). Maybe the most interesting extension is that one can use each created edge to post other constraints, for example about the well-formedness of associated typed feature structures. The posted constraints automatically become available for other constraint solvers. In particular, systems implementing HPSG seem to suffer from the problem how to drive the constraint resolution process efficiently. Some systems, as for example ALE [1], use a phrase structure backbone to drive the process. The proposal here would allow to use the ID/LP schemata directly as constraints, but nevertheless as the driving force behind the other constraint satisfaction techniques on the well-formedness of feature structures.

References

[1] B. Carpenter and G. Penn. ALE: The attribute logic engine. Carnegie Mellon University, 1998.

[2] T. Frühwirth. Theory and practice of constraint handling rules. *JLP*, 37(1–3):95–138, 1998.

[3] T. Götz and D. Meurers. Interleaving universal principles and relational constraints over typed feature logic. In *ACL/EACL Conference '97*, Madrid, Spain, 1997.

[4] S. Manandhar. An attributive logic of set descriptions and set operations. In *ACL Conference '94*, 1994.

[5] J. Matiasek. *Principle-Based Processing of Natural Language Using CLP Techniques.* PhD thesis, TU Wien, 1994.

[6] F. C. N. Pereira and D. H. D. Warren. Parsing as deduction. In *ACL Conference '83*, 1983.

[7] S. M. Shieber, Y. Schabes, and F. C. N. Pereira. Principles and implementation of deductive parsing. *JLP*, 24(1–2):3–36, 1995.

[8] K. Sikkel. *Parsing Schemata.* ETACS series, Springer, 1997.

[9] G. Smolka. The Oz programming model. In J. van Leeuwen, editor, *Computer Science Today*, LNCS 1000, pages 324–343. Springer, 1995.

TREE-STRUCTURED CHART PARSING

Paul W. Placeway
Language Technologies Institute, School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 USA
pwp@cs.cmu.edu

Abstract
We investigate a method of improving the memory efficiency of a chart parser. Specifically, we propose a technique to reduce the number of active arcs created in the process of parsing. We sketch the differences in the chart algorithm, and provide empirical results that demonstrate the effectiveness of this technique.

One basic shortcoming of a classic chart parser [6, 1, 10] is that it does not make efficient use of its grammar. In grammars used to parse natural languages, there is quite often a substantial amount of redundancy in the prefixes of the rule right-hand-sides. A naïve implementation of a chart parser will not take advantage of this redundancy. In contrast, a shift-reduce parser [4, 9, 2, 10] will often use a grammar that has been optimized to eliminate this redundancy [4]. Since chart parsing and shift-reduce parsing are substantially similar [10], many techniques used in shift-reduce parsing can be applied to a chart parser, including this particular optimization.

Tree-Structured Grammar

Consider a context-free grammar represented as follows: we will refer to a sequence of children (the "right-hand-side" of a rule) as a sequence of *shifts*, and the parent (or "left-hand-side") as the *reduce* operation. We write the rules with the children on the left leading to the parent reduction on the right. Finally, a child symbol can have multiple shifts *and* multiple reductions to its right.

standard representation	*tree representation*
$S \Leftarrow NP \quad VP$	$NP \quad VP \Rightarrow S$
$NP \Leftarrow NP \quad PP$	$\searrow PP \Rightarrow NP$

The tree grammar is then constructed in the straight-forward way, compressing the left prefixes of the right-hand-sides as much as possible.

Using the Tree-Structured Grammar

Parsing with the tree grammar is quite straightforward. The principle difference between this algorithm and the classic chart algorithm [1] is that in the classic implementation, extending an active arc results in *one* new arc, whereas when using the tree-grammar, extending an arc may result in *several* new arcs. Finally, since one active arc could spawn multiple arcs, if we must keep track of of children used to create an arc (e.g. to resolve unifications), we must do so using an up-tree [3]. The resulting inner loop remains quite straight-forward:

```
while the agenda is not empty, do:
  let e = next entry from agenda
  add e to chart.
  foreach arc in continued by e, do:
    foreach tnode in arc.tnode.shiftlist, do:
      new-arc = make-arc (e.end, tnode, traceback = (cons(e, arc.traceback))
      arc-add (new-arc)
    foreach rule in arc.tnode.reducelist
      let new-children = reverse (cons(e, arc.traceback))
      new = make-entry (first(new-children).start,
                        e.end, rule.LHS, children = list (new-children))
      agenda-add (new)
```

This technique was evaluated in a chart parser with unification, left-corner and look-ahead constraints, among other features. We used a large-scale English grammar for machine-translation of heavy equipment manuals [7, 5], and a test-set of 2524 sentences (22,558 words). Without any special restrictions, when compared to the naïve implementation, the tree-structured grammar reduced the number of active arcs created by 23%, and when employing full left-corner and look-ahead constraints [11, 8, 4] on the parser, the tree-grammar gave a 40% reduction.

Acknowledgements

This work was supported by the CMU Language Technologies Institute. The author wishes to thank Dr. Eric Nyberg, Dr. Alon Lavie, Dr. Carolyn Rosé, and the anonymous reviewers for comments that improved this paper. A full version of this paper is available as technical report from:
`http://www.cs.cmu.edu/~pwp/papers/tree_parse_tr.ps`

References

[1] James Allen. *Natural Language Understanding.* Benjamin/Cummings, Redwood City, CA, second edition, 1995.

[2] John Andrew Carroll. *Practical Unification-based Parsing of Natural Language.* PhD thesis, University of Cambridge, Computer Laboratory, September 1993.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L Rivest. *Introduction to Algorithms.* McGraw-Hill and MIT Press, Cambridge, MA, 1990.

[4] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Series in Computer Science. Addison-Wesley, Reading, MA, 1979.

[5] Kamprath, Adolphson, Mitamura, and Nyberg. Controlled Language for Multilingual Document Production: Experience with Caterpillar Technical English. In *Proc. Second Int. Workshop on Controlled Language Applications (CLAW '98)*, 1998.

[6] Martin Kay. Algorithm schemata and data structures in syntactic processing. In *Readings in Natural Language Processing.* Morgan Kaufmann, San Mateo, CA, 1986 (1980).

[7] T. Mitamura, E. Nyberg, and J. Carbonell. An efficient interlingua translation system for multi-lingual document production. In *Proc. 3rd Machine Translation Summit*, 1991.

[8] Carolyn P. Rosé and Alon Lavie. LCFLEX: An efficient robust left-corner parser, 1999.

[9] Masaru Tomita. *Efficient Parsing for Natural Language.* Kluwer, Boston, 1986.

[10] G. van Noord, M-J. Nederhof, R. Koeling, and G. Bouma. Conventional Natural Language Processing in the NWO Priority Programme on Language and Speech Technology. Technical report, Rijksuniversiteit Groningen, Vakgroep Alfa-informatica & BCN, March 1996.

[11] Gertjan van Noord. An efficient implementation of the head-corner parser. *Computational Linguistics*, 23(3):425–456, March 1997.

A PARSING METHODOLOGY FOR ERROR DETECTION

Davide Turcato[§] **Devlan Nicholson**[§] **Trude Heift**[†]
Janine Toole[§] **Stavroula Tsiplakou**[†]
Simon Fraser University, Burnaby Mountain, BC, Canada V5A 1S6
[§]Natural Language Lab, School of Computing Science
{turk,devlan,toole}@cs.sfu.ca
[†] Department of Linguistics
{heift,stavroula_tsiplakou}@sfu.ca

1 Introduction

We discuss the design of grammars for syntactic error detection. The topic is equally relevant for grammar checkers and Intelligent Language Tutoring Systems (henceforth ILTS). The proposed methodology addresses three interrelated issues recurrent in the error detection literature:

Efficiency. In error detection the search space is larger than in ordinary parsing. Therefore there is a need to keep the search space manageable and to introduce some control mechanism over parsing.

Modularity. Different knowledge sources need to be used, some of which are domain-dependent. In order to combine efficiency with modularity, different knowledge sources should be amenable of being stored separately but accessed in parallel at parsing time. A related issue is also the ability to reuse the same grammar for both error detection and ordinary parsing.

Expressive power. There is a tendency to use unification-based frameworks of the same sort used for ordinary parsing. However, a more powerful machinery is needed for error detection than for ordinary parsing.

Although the different requirements of error detection have been separately addressed in various ways, their contemporaneous satisfaction can be problematic, due to the drawbacks that solutions to one requirements can imply for other requirements. In the following, we show how an extended unification-based formalism can be used to satisfactorily address all the different requirements listed above at the same time. The described ideas have been implemented in a grammar for error detection, embedded in an ILTS for Modern Greek.

2 Methodology description

The task of our ILTS grammar module is to analyze an input sentence and return a list of violations, along with a list of 'non-violations', i.e. correct instances of grammatical phenomena occurring in a sentence, used to update an individual student model.

Expressive power. We take an approach to error detection based on a unification grammar extended via procedural attachments. We defined a typed feature structure formalism with definite clause logic programming attachments, akin to existing formalisms like ALE [1]. In such formalisms, not only unification, but also any other operation that can be defined in terms of the underlying logic programming language can be performed on linguistic descriptions. The advantage of this approach is twofold: (i) procedures of any complexity can be encoded, while keeping a single control structure driven by a declarative grammar; (ii) any number of additional knowledge sources can be separately encoded, while being all accessible at parsing time.

Modularity. One of our design guideline was to develop a *multi-functional* grammar: we wanted to be able to use the same grammar for both ordinary parsing and error detection. This can be achieved by defining attachments in a consistent way. Whenever a feature is relevant to error detection, any check on its value is done in some suitable attachment to the relevant rule, instead of the rule body. Moreover, each predicate is systematically defined as a disjunction: one clause performs whatever checks and actions are needed for error detection, the other one performs whatever standard unification is needed for ordinary parsing. The two clauses of each predicate are mutually exclusive: the former is only used when error detection is needed, the latter when only ordinary parsing is. To this end we define environment variables, again in the form of definite clause, which we use as system parameters.

All violation-related attachments are uniformly handled in three stages: (i) A *test* on the relevant features, where violations are detected; (ii) A *selection* of the action to be taken, where detected violation are interpreted, in terms of some grammatical configuration; (iii) The *execution* of the selected action, where a grammatical configuration is mapped onto an appropriate label to be added to the list of violation or non-violation, respectively. This processing scheme allows one to decouple constraint relaxation from error (or non-error) flagging, which is based on a number of other factors, independently handled. If the system were ported to a different domain, the use of a different inventory of violations in the error flagging phase would not necessarily involve changes in the error detection phase. Different error inventories can be alternatively used by setting an appropriate system parameter.

Efficiency. We implemented a mechanism that allows user-defined control strategies on parsing. A control strategy is defined via a definite clause, which takes as arguments a *linguistic description* and a *priority value*. Given a linguistic description, the clause assigns a numeric value, representing the description's priority in the parsing agenda. Edges in the parsing chart are retrieved from the agenda and asserted according to the priority value associated to them by the priority clause. In this way, parses are obtained in the order defined by a user, according to some relevant criterion. Different parsing strategies can be defined and the system can be switched between them, again by means of system parameters. For example, we use different parsing strategies depending on whether error detection or ordinary parsing is performed.

In the proposed architecture, a single control structure represented by a declarative unification grammar can be integrated with any number of further knowledge sources and procedures of any complexity, and provided with means to define a parsing strategy based on any information available. The result is a modular integrated system whose behavior can be changed between different modes by simply setting some general system parameters.

References

[1] Bob Carpenter and Gerald Penn. ALE. the Attribute Logic Engine user's guide. version 3.2 beta. Technical report, Bell Laboratories and Universität Tübingen, Murray Hill, NJ, USA and Tübingen, Germany, May 1999.

Dependency Model Using Posterior Context

Kiyotaka Uchimoto[†] **Masaki Murata**[†]
Satoshi Sekine[‡] **Hitoshi Isahara**[†]

[†]Communications Research Laboratory
588-2, Iwaoka, Iwaoka-cho, Nishi-ku
Kobe, Hyogo, 651-2492, Japan
{uchimoto,murata,isahara}@crl.go.jp

[‡]New York University
715 Broadway, 7th floor
New York, NY 10003, USA
sekine@cs.nyu.edu

Abstract

We describe a new model for dependency structure analysis. This model learns the relationship between two phrasal units called bunsetsus as three categories; 'between', 'dependent', and 'beyond', and estimates the dependency likelihood by considering not only the relationship between two bunsetsus but also the relationship between the left bunsetsu and all of the bunsetsus to its right. We implemented this model based on the maximum entropy model. When using the Kyoto University corpus, the dependency accuracy of our model was 88%, which is about 1% higher than that of the conventional model using exactly the same features.

1 Introduction

Dependency structure analysis is one of the basic techniques used in Japanese sentence analysis. The Japanese dependency structure is usually represented by the relationships between phrasal units called 'bunsetsu.' The analysis is done in two steps. In the first step, a dependency matrix is prepared where each element represents the likelihood of one bunsetsu being dependent on another in a sentence. In the second step, an optimal set of dependencies for the entire sentence is found. In this paper we only discuss the first step, a model for estimating the likelihood of dependency.

In our approach, the value for each element in the dependency matrix is estimated as a probability. We previously developed a statistical model that considers only the relationship between two bunsetsus when estimating the dependency likelihood[1]. We call this model the old model in this work. Here we describe a model that considers not only the relationship between two bunsetsus, but also the relationship between the left bunsetsu and all of the bunsetsus to its right in a sentence. The probability of whole sentence dependencies is calculated as the product of all the dependency probabilities in a sentence. By searching for the dependencies that maximize the probability, we can identify the optimal dependencies in a sentence. The dependencies in a sentence are identified by analyzing it from right to left[2].

2 Dependency Model Using Posterior Context

Given a tokenization of a test corpus, the problem of dependency structure analysis in Japanese can be reduced to the problem of assigning one of two tags to each relationship between two bunsetsus. A relationship can be tagged with a '1' or a '0' to indicate whether or not there is a dependency between the bunsetsus, respectively. Assigning these tags is the usual way to describe a dependency relationship [3, 4, 1]. However, there are two other possibilities when there is not a dependency between two bunsetsus. One is the case where an anterior bunsetsu depends on one between it and the posterior bunsetsu. The other case is where an anterior bunsetsu depends on a bunsetsu beyond the posterior one. We believe there is a big difference between the two cases.

We developed a dependency model to identify this difference. A dependency relationship between two bunsetsus is tagged with a '0,' '1,' or '2' to indicate the three cases, respectively. The anterior bunsetsu can depend on (1) a bunsetsu between the two, (2) the posterior bunsetsu, or (3) a bunsetsu beyond the posterior one. Our new model uses these three categories while the old model uses only two. The dependency probability of two bunsetsus is estimated by using the product of the probabilities of the relationship between the left bunsetsu and those to its right in a sentence.

We show how to estimate dependency probability with this model using the example in Fig. 1. Figure. 1 shows a simulated calculation of the dependency probabilities of a bunsetsu that has five bunsetsus to its right and is represented by the left most circle. The probabilities of the relationship between this bunsetsu and each modifiee candidate are shown in the table in Fig. 1. The dependencies of the five bunsetsus on the right are assumed to have been identified. Each dependency is represented by an arrow with a dotted line. In this example, bunsetsu 3 and 4 cannot be modified by the current bunsetsu because we assume that "dependencies do not cross." For example, the dependency probability between the current bunsetsu and bunsetsu 5 is calculated as shown in the bottom example in Fig. 1. It has a normalized probability of 52.2%.

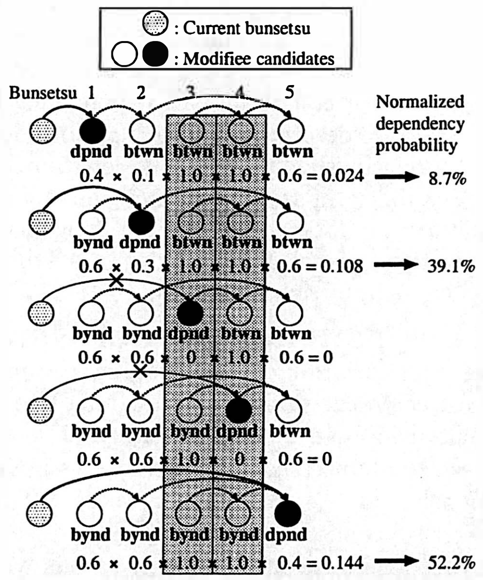
3 Experimental Result

We implemented our model based on the maximum entropy model. We used in our experiments the same features as in Ref. [1]. Those features were basically some attributes of a bunsetsu itself or those between bunsetsus. We used the Kyoto University text corpus (Version 2) [5], a tagged corpus of the Mainichi newspaper. For training we used 7,958 sentences from newspaper articles appearing from January 1st to January 8th in 1995, and for testing we used 1,246 sentences from articles appearing on January 9th. We assumed that the input sentences were morphologically analyzed and their bunsetsus were identified correctly.

The results of our experiment are shown in Table 1. The dependency accuracy means the percentage of correct dependencies out of the total analyzed. The sentence accuracy means the percentage of sentences in which all the dependencies were analyzed correctly. The first and the second lines in Table 1 compare the accuracy of our new model and the old model. The bottom line in Table 1 shows the accuracy when we assumed that every bunsetsu depended on the next one. The dependency accuracy of the new model was about 1% better than that of the old model and there was a 3% improvement in sentence accuracy. Our Investigation of the relationships between sentence length (number of bunsetsus) and dependency accuracy, and between the amount of training data (number of sentences) and the accuracy of the model found that the accuracy of the new model was almost always better than that of the old one for any sentence length. And we found that the accuracy of the new model was about 1% higher than that of the old model for any size of training data used.

Table 1: Results of dependency analysis.

Model	Dependency accuracy	Sentence accuracy
New	87.93% (9,904/11,263)	43.58%(540/1,239)
Old	87.02% (9,801/11,263)	40.68%(504/1,239)
Baseline	64.09% (7,219/11,263)	6.38%(79/1,239)



Candidate	Beyond(bynd)	Dependent(dpnd)	Between (btwn)
1	0.6	0.4	0
2	0.6	0.3	0.1
3	0.3	0.5	0.2
4	0.1	0.5	0.4
5	0	0.4	0.6

Figure 1: Simulated calculation of dependency probability.

References

[1] Kiyotaka Uchimoto, Satoshi Sekine, and Hitoshi Isahara. Japanese dependency structure analysis based on maximum entropy models. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, pages 196–203, 1999.

[2] Satoshi Sekine, Kiyotaka Uchimoto, and Hitoshi Isahara. Statistical dependency analysis using backward beam search. *Journal of Natural Language Processing*, 6(3):59–73, 1999. (in Japanese).

[3] Masahiko Haruno, Satoshi Shirai, and Yoshifumi Ooyama. Using decision trees to construct a practical parser. *Proceedings of the COLING-ACL '98*, 1998.

[4] Masakazu Fujio and Yuji Matsumoto. Japanese dependency structure analysis based on lexicalized statistics. *Proceedings of Third Conference on Empirical Methods in Natural Language Processing*, pages 87–96, 1998.

[5] Sadao Kurohashi and Makoto Nagao. Kyoto university text corpus project. In *3rd Annual Meeting of the Association for Natural Language Processing*, pages 115–118, 1997. (in Japanese).

THE EDITING DISTANCE IN SHARED FOREST

M. Vilares D. Cabrero F.J. Ribadas
Computer Science Department
University of A Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
{vilares, cabrero, ribadas}@dc.fi.udc.es

Abstract
In an information system indexing can be accomplished by creating a citation based on context-free parses, and matching becomes a natural mechanism to extract patterns. However, the language intended to represent the document can often only be approximately defined, and indices can become shared forests. Queries could also vary from indices and an approximate matching strategy becomes also necessary. We present a proposal intended to prove the applicability of tabulation techniques in this context.

1 A dynamic frame for parsing

We use ICE [2] as parsing frame. The formalism is an extended,LALR(1), push-down transducer (PDT). It proceeds by building *items*, compact representations of the PDT stacks, which are produced by applying transitions to existing ones, until no new application is possible. These items provide an optimal sharing of computations for non-deterministic inputs. A merit ordering guarantees fairness and completeness, and redundant items are ignored by using a simple subsumption relation.

We represent a parse as the chain of the context-free rules used in a leftmost reduction of the input sentence, whose non-terminals are items. The output grammar is then represented in finite shared form by an AND-OR graph that in our case is precisely the shared-forest. The time complexity (resp. space complexity) for this bottom-up parser is $\mathcal{O}(n^3)$ (resp. $\mathcal{O}(n^2)$), for inputs of length n. This complexity is lineal for deterministic inputs, which favourices the performance in practice.

2 A dynamic frame for approximate pattern matching

Given P, a target tree, and D, a data tree, we define an *edit operation* as a pair $a \rightarrow b$ with an associated cost, $\gamma(a \rightarrow b)$, that we extend to a sequence S of edit operations s_1, s_2, $\ldots$, s_n in the form $\gamma(S) = \sum_{i=1}^{|S|}(\gamma(s_i))$. The distance between P and D is defined by the metric δ, in the form $\delta(P, D) = \min\{\gamma(S),\ S$ editing sequence taking P *to* $D\}$. Given an inverse postorder traversal to name each node i of a tree T by $T[i]$, we focus on mappings [1], a particular kind of sequences. Given $\mathcal{D}$ and $\mathcal{I}$, respectively, the nodes in P and D not touched by edit operations; the cost of a mapping M is computed by $\gamma(M) = \sum_{(i,j)\in M} \gamma(P[i] \rightarrow D[j]) + \sum_{i\in\mathcal{D}} \gamma(P[i] \rightarrow \varepsilon) + \sum_{j\in\mathcal{I}} \gamma(\varepsilon \rightarrow D[j])$. It can be proved that $\delta(P, D) = \min\{\gamma(M),\ M$ mapping from P to $D\}$.

In relation to Zhang *et al.* in [3], we can deal with data shared forest. The original approach would require a less efficient top-down parsing to avoid redundant computations, due to postorder tree traversal applied and computing the distance by left-recursion on this search. The time complexity is $\mathcal{O}(\mid P \mid\mid D \mid \min(\text{depth}(P), \text{leaves}(P))\min(\text{depth}(D), \text{leaves}(D)))$, where $\mid P \mid$ (resp. $\mid D \mid$) is the number of nodes in P (resp. in the tree of D with a maximum number of nodes), leaves(P) (resp. leaves(D)) is the number of leaves in P (resp. in the tree of D with a maximum number of leaves), and depth(P) (resp. depth(D)) is the depth of P (resp. in the tree of D with maximal depth).

3 Experimental results

We consider the language, $\mathcal{L}$, of arithmetic expressions to compare our proposal with Tai [1], and Zhang *et al.* [3], using two deterministic grammars, $\mathcal{G}_L$ and $\mathcal{G}_R$, for the left and right associative versions of $\mathcal{L}$; and one non-deterministic $\mathcal{G}_N$. Parses are built using ICE, and tests have been applied on data inputs $a_1 + a_2 + \ldots + a_i + a_{i+1}$, with i even. As pattern, parse trees from inputs $a_1 + b_1 + a_3 + b_3 + \ldots + b_{i-1} + a_{i-1} + b_{i+1} + a_{i+1}$, where $b_j \neq a_{j-1}$, for all $j \in \{1, 3, \ldots i-1, i+1\}$.

Patterns are built from the left-associative (resp. right-associative) interpretation for $\mathcal{G}_L$ (resp. $\mathcal{G}_R$), as in the non-deterministic case, to evaluate the impact of traversal orientation. So, Fig. 1 proves the adaptation of our proposal (resp. Zhang *et al.*) to left-recursive (resp. right-recursive) derivations, and the gain in efficiency due to sharing of computations in a dynamic frame for $\mathcal{G}_N$.

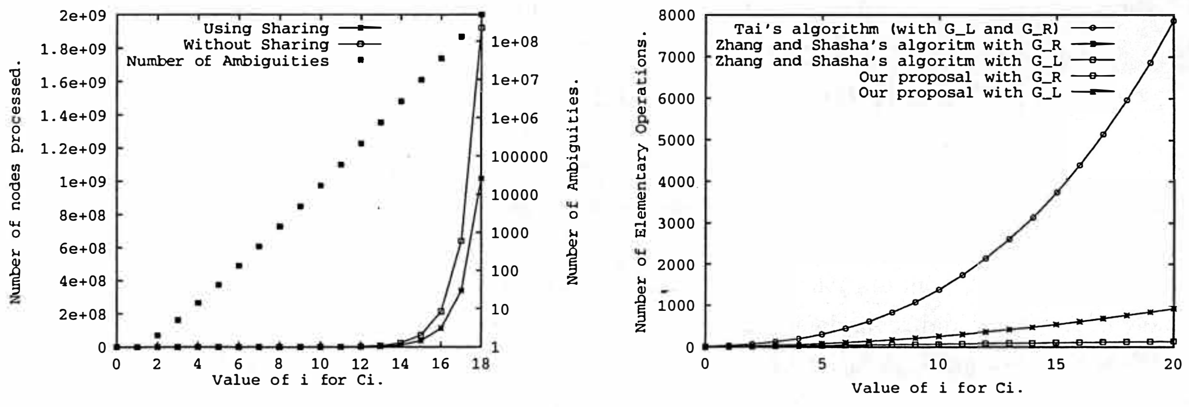


Figure 1: Results on approximate tree matching

Acknowledgements

Work partially supported by projects XUGA 20402B97 of the Autonomous Government of Galicia, project PGIDT99XI10502B of Spanish Government and project 1FD97-0047-C04-02 by the EU.

References

[1] Kuo-Chung Tai. 1978. Syntactic error correction in programming languages. IEEE Transactions on Software Engineering, SE-4(5):414-425.

[2] M. Vilares and B. A. Dion. 1994. Efficient incremental parsing for context-free languages. Proc. of the 5th IEEE Int. Conf. on Computer Languages, pages 241-252, Toulouse, France.

[3] K. Zhang and D. Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. SIAM Journal on Computing, 18:1245-1262.