An Experimental Parser for Systemic Grammars

Robert T. KASPER
USC/Information Sciences Institute
4676 Admiralty Way, Suite 1001
Marina del Rey, CA 90292 U.S.A.

Abstract

We describe a general parsing method for systemic grammars. Systemic grammars contain a paradigmatic analysis of language in addition to structural information, so a parser must assign a set of grammatical features and functions to each constituent in addition to producing a constituent structure. Our method constructs a parser by compiling systemic grammars into the notation of Functional Unification Grammar. The existing methods for parsing with unification grammars have been extended to handle a fuller range of paradigmatic descriptions. In particular, the PATR-II system has been extended by using disjunctive and conditional information in functional descriptions that are attached to phrase structure rules. The method has been tested with a large grammar of English which was originally developed for text generation. This testing is the basis for some observations about the bidirectional use of a grammar.

1 Introduction

Many computational linguists have found systemic grammar (SG) to be quite useful, because it provides an explicit representation of features that determine how a sentence functions in the context of communication. SG has been used directly as the basis for several computer text generation programs [Mann 82], but it has only been used indirectly for computational parsers. Winograd used principles from SG in designing SHRDLU [Winograd 72], a successful natural language understanding program, but his program did not contain an explicit representation of the grammar's system network. Instead, he used a special purpose programming language to encode grammatical knowledge in a procedural form tailored specifically to the language understanding task. Another procedural implementation of SG was developed by McCord [McCord 77]. Both of these methods would require a significant programming step before a parser could be produced for a different grammar. Our goal has been to develop a general parsing method using a declarative representation of SG, and to determine to what extent a grammar that is adequate for text generation can also be used for text analysis. Our parser has been developed and tested using Nigel [Mann 83], a large grammar of English that has previously been used as part of a text generation system.

Systemic linguistics builds on the foundation of Halliday's concept of the *system network* [Halliday 76]. A systemic grammar is organized around choices between grammatical features that reflect the structure and content of a constituent. Each choice between features is called a *system*. Thus, a systemic grammar has two major components:

1. a system network of feature choices, and
2. structural realization statements corresponding to each feature.

The feature choices define the options available to be expressed in a language, and may be regarded as "hooks" into a semantic component. The realization statements determine the constituent structure. There are realization statements to declare the presence of constituents, conflate constituents, specify feature constraints on constituents, and specify ordering constraints among constituents.

Consider, for example, the fragment of a grammar of English clauses shown in Figure 1. There are two systems, labeled by Mood-type and Indicative-type. Each system has an input condition to its left, specifying when its options are applicable. The input condition for Indicative-type is the single feature, Indicative, but input conditions may also be expressed by boolean combinations of features. In each system, exactly one of the features to the right of the vertical bar must be chosen. For example, in the Indicative-type system, either Declarative or Interrogative must be chosen. Under each feature are realization statements, such as SUBJECT $\wedge$ FINITE under the Declarative feature. This statement specifies that the SUBJECT constituent must precede the FINITE constituent in declarative clauses. Each realization statement is associated with a particular feature, so that structural constraints are distributed throughout the system network. The distributed nature of structural information in SG presents a challenge to the design of a parser, which we will address in Section 3.

In addition to building a constituent structure for a sentence, as do most syntactic approaches to natural language parsing, a parser for SG must also perform the following tasks:

1. determine the set of systemic features for each constituent,
2. assign grammatical functions to each constituent.

Other theories of grammar also make use of features and grammatical functions, however they have a distinct significance in systemic theory. The feature set associated with a constituent plays an important role in specifying its meaning (i.e., the features are not simply discarded after syntactic analysis), so a relatively large number (e.g., over 50) of features may need to be assigned to each constituent. Each constituent may also be assigned to several grammatical functions, because of the *multifunctional* nature of systemic analysis. For example, it is common to describe a single constituent simultaneously as SUBJECT, ACTOR and TOPIC. Therefore, in order to determine that a clause has an ACTOR, it may be necessary to check whether the clause has a SUBJECT and whether the SUBJECT of the clause is conflatable with the ACTOR function.

An example of the type of output produced by the parser is shown in Figure 2. This example shows only the functional structure that the parser assigns to the sentence. In addition, each constituent is assigned a set of grammatical features, such as Indicative and Declarative. These features are also accessible in the data structures produced by the parser, but they are too numerous to display in this short paper.

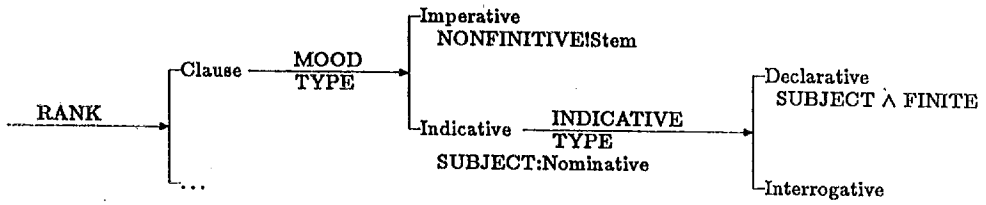

Figure 1: The Mood-type and Indicative-type Systems.

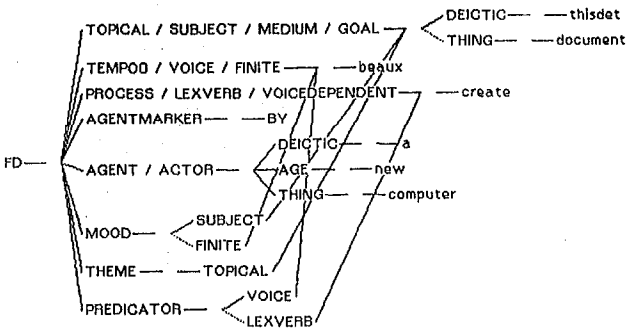

Figure 2: Functional Structure of: "This document was created by a new computer."

2 Compilation into Functional Unification Grammar

The basic method used to construct the parser has been to develop a compiled representation of systemic grammars in the notation of Functional Unification Grammar (FUG). The parsing process itself is then derived by extending methods already developed for parsing with FUG [Kay 85]. In FUG, a grammar can be regarded as a special kind of logical formula [Rounds 87], and the parsing problem is reduced to finding the set of feature structures that satisfy the formula subject to the constraints of the words in a particular sentence. Using the feature description logic (FDL) of Kasper and Rounds [Kasper 86], the types of formula used to define a grammar include:[1]

NIL	denoting *no information*;
a	where $a \in A$, to describe atomic values;
$l : \phi$	where $l \in L$ and $\phi \in$ FDL, to describe structures in which the feature labeled by l has a value described by ϕ;
$\exists l$ or $l : ANY$	where $l \in L$, to describe a structure in which l has a substantive (non-NIL) value;
$< p >$	where $p \in L^*$, to describe a structure that shares a common value with the path p;
$[\phi_1 \ldots \phi_n]$	where $\phi_i \in$ FDL, denoting conjunction;
$\{\phi_1 \ldots \phi_n\}$	where $\phi_i \in$ FDL, denoting disjunction;
$\phi_1 \rightarrow \phi_2$	where $\phi_i \in$ FDL, denoting classical implication.

The last type of formula, denoting implication, is an extension to FUG that enables a more efficient modeling of systemic descriptions than is possible in Kay's version of FUG [Kasper 87d].

The compilation of systems into FUG is relatively straightforward. Each system is represented by a disjunction containing alternatives for each feature that can be chosen in the system. These alternatives also contain attributes that represent constraints on grammatical functions imposed by realization statements. For example, the Mood-type and Indicative-type systems can be represented by the description shown in Figure 3. System input conditions are bidirectional: they are represented by the embedding of descriptions, and also by feature existence conditions.

In the FUG representation there is one *functional description* (FD) corresponding to each major constituent category of the systemic grammar. Major constituent categories for English include *clause, nominal-group*, and *prepositional-phrase*. The method of representing a systemic grammar as a set of FDs in FUG is described in greater detail in [Kasper 87b,Kasper 87d]. A program has been implemented to automatically translate any system network into FDs, verifying the effectiveness and generality of this compilation procedure. This program has been used to compile the entire Nigel grammar, which contains over 500 systems, into FUG many different times as changes to the grammar have been made.

[1] Let A and L be sets of symbols used to denote atomic values and feature labels, respectively.

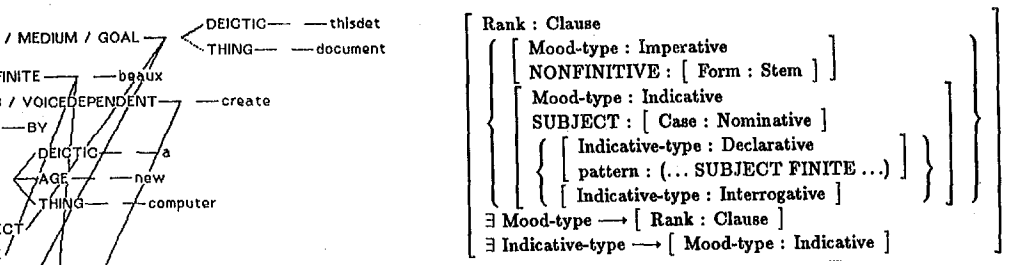

Figure 3: The Mood-type and Indicative-type Systems in extended-FUG notation.

3 Parser Implementation: Extending PATR-II

Our early experiments using the Nigel grammar showed that the existing methods for parsing with FUG had several shortcomings when applied to a large grammar. Kay's method for parsing with FUG [Kay 85] cannot be applied directly to our grammar because it requires:

1. expanding the grammar FD to disjunctive normal form (DNF);
2. creating a disjunct for each possible ordered combination of constituents that is compatible with pattern features. Each of these disjuncts can be regarded as equivalent to an annotated phrase structure rule.

In both cases our grammar contains too many alternatives to carry out the procedure:

1. Our grammar of English clauses contains over 100 systems. Since each system is represented by a disjunction in FUG, the DNF expansion of the clause grammar might require over 2^{100} disjuncts!
2. Our grammar contains many optional grammatical functions. A particularly striking example concerns the large number of optional adjunct types that may be used to modify an English clause.[2] These adjuncts occur most frequently at the end of the clause, although other orders are possible. Assuming that there are at least 10 optional adjunct types, we have 2^{10} different combinations of adjuncts, not counting any additional combinations resulting from order variation.

The first problem has been solved by a new unification algorithm for disjunctive descriptions that does not require expansion to DNF [Kasper 87c]. The second problem has been solved by adding a small phrase structure component to the grammar and using the PATR-II active chart parsing algorithm, which was developed by Shieber et al. at SRI [Shieber 84].

3.1 Skeletal Phrase Structure Component

The role of phrase structure (PS) rules in our parser is similar to their role in Lexical Functional Grammar [Kaplan 83], however they have less theoretical significance in our parser. We use the PS component to recognize possible patterns for each major constituent category, but the unification component builds the functional structure and assigns a feature set to each constituent. The PS component is something like a skeleton that cannot be seen in the final descriptions produced by the parser. Not very many PS rules are required, because they only need to encode broad category distinctions. Fine category distinctions are encoded by the FDs that are attached to rules. Each major constituent category of the grammar has a special rule that is annotated with the FD produced by compilation from the systemic grammar for that category. For example, the category CLAUSE has a rule of the form:

[2] A partial list of adjunct types includes: MANNER, CAUSE, ACCOMPANIMENT, SPACE-LOCATIVE, SPACE-EXTENT, TIME-LOCATIVE, TIME-EXTENT, MATTER, ROLE, ATTITUDE.

```
CLAUSE  →  CLAUSE-PS:
           <CLAUSE> = <CLAUSE-PS fd>
           <CLAUSE> = [ compiled FD for CLAUSE ].
```

CLAUSE-PS is a non-terminal that can derive any valid constituent pattern for clauses. The first unification of this rule identifies any features that are known from the constituents derived by CLAUSE-PS with features of the CLAUSE nonterminal. The second unification provides the functional description that must be satisfied for any clause.

Consider again the problem of optional adjuncts. Instead of producing a distinct disjunct for each combination of adjuncts, it is much more efficient to describe all possible combinations using a single recursive PS rule. This rule is annotated with a disjunctive description that contains a single alternative for each adjunct type:

```
CLAUSE-PS1  →  CLAUSE-PS2 ADJUNCT:
               <CLAUSE-PS-1> = {  [ MANNER : <ADJUNCT>]
                                  [ CAUSE : <ADJUNCT>]
                                  ... other alternatives }.
```

The PS component is the only part of the grammar used by the PATR-II parser that is not produced automatically from a systemic grammar. The parsing grammar for Nigel currently contains about 60 PS rules.

3.2 Extensions to PATR-II

The PATR-II system has been extended in several significant ways to carry out our implementation:

1. handling disjunctive and conditional descriptions [Kasper 87c,Kasper 87d];
2. using tables compiled from the realization statements of SG. These tables include the possible conflations for each grammatical function, and lexical items that are associated with particular features in the grammar.

The compiled tables and skeletal phrase structure component enable the parser to directly deduce structural information about a sentence, despite the distributed nature of structural constraints in SG.

4 Bidirectional Grammar

Bidirectional grammar, i.e. using the same grammatical knowledge for both parsing and generation of a language, has been a real but sometimes elusive goal in computational linguistics. The goal of bidirectional grammar was clearly a motivation for Kay's formulation of FUG [Kay 85]. Kay has shown that if a declarative representation is used to encode the grammatical knowledge of a language, then it should be possible to compile that knowledge into appropriate data structures for parsing or generation. We have followed this method in constructing a parser for systemic grammars by compiling the grammar into a notation like FUG. Our discussion in this section focuses on other issues besides compilation that have been identified in our effort to develop a bidirectional systemic grammar.

Our experience with the Nigel grammar has indicated that it is possible to develop a bidirectional grammar within the systemic-functional framework, although a substantial amount of effort may be required to tune the grammar for both parsing and generation. In other words, the framework of systemic grammars is potentially invertible, but particular grammars may require some modification before they can be used effectively for both parsing and generation. Generally, parsing places greater demands on the realization component of the grammar, while generation places greater demands on the systems of choice. The Nigel grammar was originally developed for use in a text generation program, so our observations deal mostly with problems that can arise when inverting a grammar that is adequate for generation but untested for analysis.

Most modifications that we have made to enable parsing involve eliminating unintended ambiguities or disjunctive alternatives in the grammar that require an inordinate amount of time to resolve. Systemic grammars can exhibit ambiguity between grammatical features, in addition to the well known types of lexical and structural ambiguity.

Unintended ambiguities between grammatical features often arise from underspecified parts of the grammar, i.e., the grammar contains an alternation between two or more features with insufficient realization information to determine which features apply in many cases. Usually the solution to this problem is to add realization information for those features. Sometimes the realization of those features may depend on other features and the modification is somewhat complex. In such cases, it is possible to temporarily disable the underspecified alternatives while parsing until a more complete solution is developed.

Some features may have realizations that are formally adequate and efficient for generation, but quite inefficient for parsing. For example, the Nigel grammar for nominal groups contains the *Pronominal* feature to indicate that the head constituent is a pronoun. There is no explicit realization statement associated with this feature, but the system network contains more specific features for each of type of pronoun. These more specific features have realizations that specify particular lexical items for the head constituent. Since English pronouns are a closed class, there is a finite number of features that need to be examined to determine whether a nominal group is pronominal. However, it is quite inefficient to consider each member of the class individually. Obviously, we can improve the parsing efficiency of the grammar by adding a realization to the Pronominal feature that constrains the head to be a member of the class of pronouns. We have found a significant number of similar cases, where the grammar was adequate for generation, but was missing some useful generalization for analysis.

It seems reasonable to expect that most grammars that are originally developed specifically for generation or parsing tasks will need similar kinds of tuning before they can be used effectively for the inverse task. A bidirectional grammar seems to be a reachable goal, but it will probably have some specifications that are superfluous for either parsing or generation. These specifications can be marked if necessary for efficiency, so that the parser or generator does not have to examine unnecessary information.

5 Conclusions

We have developed a general method for parsing systemic grammars by extending the techniques of FUG and the PATR-II system. The parser is reasonably efficient for grammar testing and use as a linguistic research tool, but further refinement would be necessary for applications demanding real-time performance. Using the full Nigel grammar, it currently requires less than a minute to parse simple single-clause sentences, and several minutes to parse more complex sentences. It should be noted that parsing speed depends heavily on grammar size, and we are using a grammar that is significantly larger than most grammars that have been implemented to this date with unification-based methods.

We have only investigated an exhaustive bottom-up strategy, in which the parser produces all possible parses for a sentence. This strategy is well-suited to grammar testing, but other strategies should be developed for applications demanding more selectivity and efficiency. We have not yet attempted to incorporate extra-grammatical (i.e., semantic and pragmatic) information for ambiguity resolution, but this would also be necessary for most practical applications.

It would be very desirable to discover a way to produce the phrase structure component of the parsing grammar, or some functionally equivalent mechanism, automatically from a systemic description. If accomplished, this would make it possible to fully automate the production of a parsing grammar, but this appears to be a difficult problem. It is currently much easier to produce a small phrase structure component manually from one's knowledge of the grammar.

Acknowledgements

I would like to thank Bill Mann for originally suggesting and encouraging this topic of research. I would also like to thank Christian

Matthiessen, Martin Kay, Lauri Karttunen, John Bateman and Bill Rounds for helpful comments on the design of the parser, and Stuart Shieber for providing help in the use of the PATR-II system.

This research was sponsored in part by the United States Air Force Office of Scientific Research contract F49620-87-C-0005, and in part by the United States Defense Advanced Research Projects Agency under contract MDA903-81-C-0335; the opinions expressed here are solely those of the author.

References

[Kaplan 83] Kaplan, R. and J. Bresnan. Lexical Functional Grammar: A Formal System for Grammatical Representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, Massachusetts, 1983.

[Kasper 87a] Kasper, R. *Feature Structures: A Logical Theory with Application to Language Analysis*. PhD dissertation, University of Michigan, 1987.

[Kasper 87b] Kasper, R. Systemic Grammar and Functional Unification Grammar. In J. Benson and W. Greaves, editors, *Systemic Functional Approaches to Discourse*, Norwood, New Jersey: Ablex (in press). Also available as USC/Information Sciences Institute Reprint RS-87-179.

[Kasper 87c] Kasper, R. A Unification Method for Disjunctive Feature Descriptions. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, CA, July 6-9, 1987. Also available as USC/Information Sciences Institute Reprint RS-87-187.

[Kasper 87d] Kasper, R. *Conditional Descriptions in Functional Unification Grammar*. USC/Information Sciences Institute Research Report RR-87-191, November, 1987.

[Kasper 86] Kasper, R. and W. Rounds. A Logical Semantics for Feature Structures. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, Columbia University, New York, NY, June 10-13, 1986.

[Kay 85] Kay, M. Parsing in Functional Unification Grammar. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Parsing*. Cambridge University Press, Cambridge, England, 1985.

[Halliday 76] G.R. Kress, editor. *Halliday: System and Function in Language*. Oxford University Press, London, England, 1976.

[Mann 82] Mann, W.C. Text Generation. Section of Applied Computational Linguistics in Perspective: Proceedings of the Workshop, In *American Journal of Computational Linguistics*, Vol. 8:2, 1982.

[Mann 83] Mann, W.C. and C. Matthiessen. Nigel: A Systemic Grammar for Text Generation. USC / Information Sciences Institute, RR-83-105. Also appears in R. Benson and J. Greaves, editors, *Systemic Perspectives on Discourse: Selected Papers Papers from the Ninth International Systemics Workshop*, Ablex, London, England, 1985.

[McCord 77] McCord, Michael C. Procedural systemic grammars. In *International Journal of Man-Machine Studies*, Vol. 9, pp. 255-286, 1977.

[Rounds 87] Rounds, W. C. and Manaster-Ramer, A. A Logical Version of Functional Unification Grammar. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, Stanford University, Stanford, CA, July 6-9, 1987.

[Shieber 84] Shieber, S. M. The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics: COLING 84*, Stanford University, Stanford, California, July 2-7, 1984.

[Winograd 72] Winograd, T. *Understanding Natural Language*. New York: Academics Press, 1972.