NOTES FROM THE UNIFICATION UNDERGROUND:

A COMPILATION OF PAPERS ON UNIFICATION-BASED GRAMMAR FORMALISMS

Technical Note 327

June 1984

By: Stuart M. Shieber,
Lauri Karttunen, and
Fernando C. N. Pereira

Artificial Intelligence Center
SRI International
and
Center for the Study of Language and Information
Stanford University

This research has been made possible in part by a gift from the Systems Development Foundation, and was also supported by the Defense Advanced Research Projects Agency under Contracts N00039-80-C-0575 and N00039-84-K-0078 with the Naval Electronic Systems Command.

333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 326-6200 • TWX: 910-373-2046 • Telex: 334-486

Notes from the Unification Underground:

A Compilation of Papers on Unification-Based Grammar Formalisms

Stuart M. Shieber, Lauri Karttunen, and Fernando C. N. Pereira
Artificial Intelligence Center
SRI International
and
Center for the Study of Language and Information
Stanford University

Contents

Preface

This report is a compilation of papers by the PATR group in the Artificial Intelligence Center at SRI International reporting on ongoing research on both practical and theoretical issues concerning grammar formalisms. The current formalism being simultaneously designed, implemented and used by the group, PATR-II, is based on unification of directed-graph structures. Unification is thus a theme both of our research, and of the papers reproduced in this volume. These papers provide an overview of the design of PATR-II (Chapter 1), a discussion of the use of disjunction and negation in unification-based feature systems (Chapter 2), and a theoretical framework for unification-based grammar formalisms which is founded on the domain-theoretic techniques of Dana Scott (Chapter 3). All three chapters are versions of papers presented at the Tenth International Conference on Computational Linguistics at Stanford University, Stanford, California during July 2 through 7, 1984.

The research was initially part of the KLAUS (Knowledge Learning And Using System) project at SRI, set up with the intention of experimenting with mathematically well-founded alternatives to the DIALOGIC natural-language processing system. The more theoretical research was made possible in part by a gift from the Systems Development Foundation and was conducted as part of a coordinated research effort with the Computer Languages subprogram at the Center for the Study of Language and Information, Stanford University.

The PATR group at SRI is a rather liquid group of researchers which has included, at various times, John Bear, Lauri Kart-

tunen, Fernando Pereira, Jane Robinson, Stan Rosenschein, Stuart Shieber, Susan Stucky, Mabry Tyson, and Hans Uszkoreit. The research reported here is a direct result of their aid and interaction. However, they should not be blamed for any errors in the present work, nor should the opinions expressed herein be construed as indicative of their personal predilections.

Chapter 1.

The Design of a Computer Language for Linguistic Information

This chapter was written by Stuart M. Shieber of the Artificial Intelligence Center, SRI International and the Center for the Study of Language and Information, Stanford University.

Abstract[1]

A considerable body of accumulated knowledge about the design of languages for communicating information to computers has been derived from the subfields of programming language design and semantics. It has been the goal of the PATR group at SRI to

[1]This research has been made possible in part by a gift from the Systems Development Foundation, and was also supported by the Defense Advanced Research Projects Agency under Contracts N00039-80-C-0575 and N00039-84-K-0078 with the Naval Electronic Systems Command. The views and conclusions contained in this document are those of the author and should not be interpreted as representative of the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency, or the United States government.

The author is indebted to Fernando Pereira, Barbara Grosz, and Ray Perrault for their comments on earlier drafts.

utilize a relevant portion of this knowledge in implementing tools to facilitate communication of linguistic information to computers. The PATR-II formalism is our current computer language for encoding linguistic information. This paper, a brief overview of that formalism, attempts to explicate our design decisions in terms of a set of properties that effective computer languages should incorporate.

1.1. Introduction

The goal of natural-language processing research can be stated quite simply: to endow computers with human language capability. The pursuit of this objective, however, has been a difficult task for at least two reasons: first, this capability is far from being a well-understood phenomenon; second, the tools for teaching computers what we do know about human language are still very primitive. The solution of these problems lies within the respective domains of linguistics and computer science.

Similar problems have arisen previously in computer science. Whenever a new computer application area emerges, there follow new modes of communication with computers that are geared towards such areas. Computer languages are a direct result of this need for effective communication with computers. A considerable body of accumulated knowledge about the design of languages for communicating information to computers has been derived from the subfields of programming language design and semantics. It has been the goal of the PATR group at SRI[2] to utilize a relevant portion of this knowledge in implementing tools to facilitate communication of linguistic information to computers.

The PATR-II formalism is our current computer language for encoding linguistic information. This paper, a brief overview

[2]This rather liquid group has included at various times: John Bear, Lauri Karttunen, Fernando Pereira, Jane Robinson, Stan Rosenschein, Susan Stucky, Mabry Tyson, Hans Uszkoreit, and the author.

of that formalism, attempts to explicate our design decisions in terms of a set of properties that effective computer languages should incorporate, namely: simplicity, power, mathematical well-foundedness, flexibility, implementability, modularity, and declarativeness. More extensive discussions of various aspects of the PATR-II formalism and systems can be found in papers by Shieber *et al.*, [83], Pereira and Shieber [84] and Karttunen [84].

The notion of designing specialized computer languages and systems to encode linguistic information is not new; PROGRAMMAR [Winograd, 72], ATNs [Woods, 70], and DIALOGIC [Grosz, *et al.*, 82] are but a few of the better-known examples. Furthermore, a trend has arisen recently in linguistics towards declarativeness in grammar formalisms—for instance, lexical-functional grammar (LFG) [Bresnan, 83], generalized phrase-structure grammar (GPSG) [Gazdar and Pullum, 82] and functional unification grammar (UG) [Kay, 83]. Finally, in computer science there has been a great deal of interest in declarative languages (e.g., logic programming and specification languages), and their supporting denotational semantics. But to our knowledge, no attempt has yet been made to combine the three approaches so as to yield a declarative computer language with clear semantics designed specifically for encoding linguistic information. Such a language, of which PATR-II is an example, would reflect a felicitous convergence of ideas from linguistics, artificial intelligence, and computer science.

1.2. The Critical Properties of the Language

It is not the purpose of this paper to provide a comprehensive description of the PATR-II project, or even of the formalism itself. Rather, we will discuss briefly the critical properties of PATR-II to give a flavor for our approach to the design of the language. References to papers with more complete descriptions of particular aspects of the project are provided when appropriate.

1.2.1. Simplicity: An Introduction to the PATR-II Formalism

Building on a convergence of ideas from the linguistics and AI communities, PATR-II takes as its primitive operation an extended pattern-matching technique, *unification*, first used in logic and theorem-proving research and lately finding its way into research in linguistics [Kay, 79; Gazdar and Pullum, 82] and knowledge representation [Reynolds, 70; Ait-Kaci, 83]. Instead of unifying logic terms, however, PATR unification operates on directed acyclic graphs (DAG).[3]

DAGs can be atomic symbols or sets of label/value pairs, where the values are themselves DAGs (either atomic or complex). Two labels can have the same value—thus the use of the term *graph* rather than *tree*. DAGs are notated either by drawing the graph structure itself, with the labels marking the arcs, or, as in this paper, by notating the sets of label/value pairs in square brackets, with the labels separated from their values by a colon; e.g., a DAG associated with the verb "knight" (as in "Uther wants to knight Arthur") would appear (in at least one of our grammars) as

```
[cat: v
 head: [aux: false
        form: nonfinite
        voice: active
        trans: [pred: knight
                arg1: <f1134>
                      []
                arg2: <f1138>
                      []]]
 syncat: [first: [cat: np
```

[3]Technically, these are rooted, directed, acyclic graphs with labeled arcs. Formal definition of these and other technical notions can be found in Appendix A of Shieber *et al.* [83]. Note that some implementations have been extended to handle cyclic graph structures as well as graph structures with disjunction and negation [Karttunen, 84].

```
               head: [trans: <f1134>]]
      rest: [first: [cat: np
                       head: [trans: <f1138>]]
              rest: <f1140>
                     lambda]
      tail: <f1140>]]
```

Reentrant structure is notated by labeling the DAG with an arbitrary label (in angle brackets), then using that label for future references to the DAG.

Associated with each entry in the lexicon is a set of DAGs.[4] The root of each DAG will have an arc labeled *cat* whose value will be the category of the associated lexical entry. Other arcs may encode information about the syntactic features, translation, or syntactic subcategorization of the entry. But only the label *cat* has any special significance; it provides the link between context-free phrase structure rules and the DAGs, as explicated below.

PATR-II grammars consist of rules with a context-free phrase structure portion and a set of unifications on the DAGs associated with the constituents that participate in the application of the rule. The grammar rules describe how constituents can be built up to form new constituents with associated DAGs. The right side of the rule lists the *cat* values of the DAGs associated with the filial constituents; the left side, the *cat* of the parent. The associated unifications specify equivalences that must exist among the various DAGs and sub-DAGs of the parent and children. Thus, the formalism uses only one representation—DAGs—for lexical, syntactic, and semantic information, and one operation—unification—on this representation.

By way of example, we present a trivial grammar for a fragment of English with a lexicon associating words with DAGs.

$S \rightarrow NP\ VP$

[4]In our implementation, this association is not directly encoded—since this would yield a grossly inefficient characterization of the lexicon—but is mediated by a morphological analyzer. See Section 1.2.6 for further details.

$$<VP\ agr> \ = \ <NP\ agr>$$

$$VP \rightarrow V\ NP$$

$$<VP\ agr> \ = \ <V\ agr>$$

Uther:

$$\begin{aligned} <cat> &= np \\ <agr\ number> &= singular \\ <agr\ person> &= third \end{aligned}$$

Arthur:

$$\begin{aligned} <cat> &= np \\ <agr\ number> &= singular \\ <agr\ person> &= third \end{aligned}$$

knights:

$$\begin{aligned} <cat> &= v \\ <agr\ number> &= singular \\ <agr\ person> &= third \end{aligned}$$

This grammar (plus lexicon) admits the two sentences "Uther knights Arthur" and "Arthur knights Uther." The phrase structure associated with the first of these is:

[$_S$ [$_{NP}$ Uther] [$_{VP}$ [$_V$ knights] [$_{NP}$ Arthur]]]

The VP rule requires that the *agr* feature of the DAG associated with the VP be the same as (unified with) the *agr* of the V. Thus, the VP's *agr* feature will have as its value the *same* node as the V's *agr*, and hence the same values for the *person* and *number* features. Similarly, by virtue of the unification associated with the S rule, the NP will have the same *agr* value as the VP and, consequently, the V. We have thus encoded a form of subject-verb agreement.

Note that the process of unification is *order-independent*. For instance, we would get the same effect regardless of whether the unifications at the top of the parse tree were effected before or after those at the bottom. In either case, the DAG associated with, e.g., the VP node would be

```
[cat: vp
 agr: [person: third
       number: singular]]
```

These trivial examples of grammars and lexicons offer but a glimpse of the techniques used in writing PATR-II grammars, and do not begin to employ the power of unification as a general information-passing mechanism. Examples of the use of PATR-II for encoding much more complex linguistic phenomena can be found in Shieber *et al.* [83].

1.2.2. Power: Two Variants

Augmented phrase-structure grammars such as PATR-II can in fact be quite powerful. The ability to encode unbounded amounts of information in the augmentations (which PATR-II obviously allows) gives this formalism the power of a Turing machine. As a linguistic theory, this much power might be considered disadvantageous; as a computer language, however, such power is clearly desirable, since the intent of the language is to enable the modeling of many kinds of linguistic analyses from a range of theories. As such, PATR-II is a tool, not a result.

Nevertheless, a good case could be made for maintaining at least the decidability of determining whether a string is admitted by a PATR-II grammar. This property can be ensured by requiring the context-free skeleton to have the property of ***off-line parsability*** [Pereira, 83], which was used originally in the definition of LFG to maintain the decidability of that formalism [Kaplan and Bresnan, 83]. Off-line parsability requires that the context-free "skeleton" of the grammar allows no trivial cyclic derivations of the form $A \overset{*}{\Rightarrow} A$.

1.2.3. Mathematical Well-Foundedness: A Denotational Semantics

One reason for maintaining the simplicity of the bare PATR-

II formalism is to permit a clean semantics for the language. We have provided a denotational semantics for PATR-II [Pereira and Shieber, 84] based on the information systems domain theory of Dana Scott [Scott, 82]. Insofar as more complex formalisms, such as GPSG and LFG, can be modeled as appropriate notations for PATR-II grammars, PATR-II's denotational semantics constitutes a framework in which the semantics of these formalisms can also be defined, discussed, and compared. As it appears that not all the power of domain theory is needed for the semantics of PATR-II, we are currently pursuing the possibility of building a semantics based on a less powerful model.[5]

1.2.4. Flexibility: Modeling Linguistic Constructs

Clearly, the bare PATR-II formalism, as it was presented in Section 1.2.1, is sorely inadequate for any major attempt at building natural-language grammars because of its verbosity and redundancy. Efficiency of encoding was temporarily sacrificed in an attempt to keep the underlying formalism simple, general, and semantically well-founded. However, given a simple underlying formalism, we can build more efficient, specialized languages on top of it, much as MACLISP might be built on top of pure LISP. And just as MACLISP need not be implemented (and is not implemented) directly in pure LISP, specialized formalisms built conceptually on top of pure PATR-II need not be so implemented (although currently we do implement them directly through pure PATR-II). The effectiveness of this approach can be seen in the fact that at least a sizable portion of English syntax has been encoded in various experimental PATR-II grammars constructed to date. The syntactic constructs encoded include subcategorization of various complement types (*NP*s, $\overline{S}$s, etc.), active, passive, "there" insertion, extraposition, raising, and equi-NP constructions, and unbounded

[5]But see Pereira and Shieber [84] for arguments in favor of using domain theory even if all the available power is not utilized.

dependencies (such as Wh-movement and relative clauses). Other theory-dependent devices that have been modeled with PATR-II include head-feature percolation [Gazdar and Pullum, 82], and LFG-like semantic forms [Kaplan and Bresnan, 83]. Note that none of these constructs and techniques required expansion of the underlying formalism; indeed, the constructions all make use of the techniques described in this section. See Shieber *et al.* [83] for a detailed discussion of the modeling of some of these phenomena.

The devices now available for molding PATR-II to conform to a particular intended usage or linguistic theory are in their nascent stage. However, because of their great importance in making the PATR-II system a usable one, we will discuss them briefly. It is important to keep in mind that these methods should not be considered a part of the underlying formalism, but merely "syntactic sugar" to increase the system's utility and allow it to conform to a user's intentions.

1.2.4.1. Templates

Because so much of the information in the PATR-II grammars under actual development tends to be encoded in the lexicon, most of our research has been devoted to methods for removing redundancy in the lexicon by allowing the users themselves to define primitive constructs and operations on lexical items. Primitive constructs, such as the transitive, dyadic, or equi-NP properties of a verb, can be defined by means of ***templates***, that is, DAGs that encode some linguistically isolable portion of the DAG of a lexical item. These template DAGs can then be combined to build the lexical item out of the user-defined primitives.

As a simple example, we could define (with the following syntax) the template *Verb* as

Let *Verb* be

$$<cat> \;=\; V$$

and the template *ThirdSing* as

Let *ThirdSing* be

$$\begin{aligned} <\textit{agr number}> &= \textit{singular} \\ <\textit{agr person}> &= \textit{third} \end{aligned}$$

The lexical entry for "knights" would then be

knights:

Verb ThirdSing

Templates can themselves refer to other templates, enabling definition of abstract linguistic concepts hierarchically. For instance, a ***modal verb*** template may use an ***auxiliary verb*** template, which in term may be defined using the ***verb*** template above. In fact, templates are currently employed for abstracting notions of subcategorization, verb form, semantic type, and a host of other concepts.

1.2.4.2. Lexical Rules

More complex relationships among lexical items can be encoded by means of lexical rules. These rules, such as passive and "there" insertion, are user-definable operations on the lexical items, enabling one variant of a word to be built from the specification of another variant. A lexical rule is specified as a set of selective unifications relating an input DAG and an output DAG. Thus, unification is the primitive used in this device as well.

Lexical rules are used to encode the relationships among various lexical entries that would typically be thought of as transformations or relation-changing rules (depending on one's ideological outlook). Because lexical rules perform these operations, the lexicon need include only a prototype entry for each verb. The variant forms can be derived through lexical rules applied in accordance with the morphology actually found on the verb. (The morphological analysis in the implementations of PATR-II is performed by a program based on the system of Koskenniemi [83] and was written by Lauri Karttunen [83].)

For instance, given a PATR-II grammar in which the DAGs are used to emulate the f-structures of LFG, we might write a *passive* lexical rule as follows (following Bresnan [83]):[6]

Define *Passive* as

$$\begin{aligned} <out\ cat> &= <in\ cat> \\ <out\ form> &= passprt \\ <out\ subj> &= <in\ obj> \\ <out\ obj> &= <in\ subj> \end{aligned}$$

The rule states in effect that the output DAG (the one associated with the passive verb form) marks the lexical item as being a passive verb whose object is the input DAG's subject and whose subject is the input's object. Such lexical rules have been used for encoding the active/passive dichotomy, "there" insertion, extraposition, and other so-called relation-changing rules.

1.2.5. Modularity and Declarativeness

The PATR-II formalism is a completely declarative formalism, as evidenced by its denotational semantics and the order-independence of its definition. Modularity is achieved through the ability to define primitive templates and lexical rules that are shared among lexical items, as well as by the declarative nature of the grammar formalism itself, removing problems of interaction of rules. Rules are guaranteed to always mean the same thing, regardless of the environment of other rules in which they are placed.

1.2.6. Implementability

Implementability is an empirical matter, given credence by the fact that we now have three implementations of the formalism. One desirable aspect of the simplicity and declarative nature of the formalism is that even though the three implementations differ substantially from one another, using different parsing algo-

[6]The example is merely meant to be indicative of the syntax for and operation of lexical rules. We do not present this as a valid definition of *Passive* for any grammar we have written in PATR-II.

rithms (with both top down and bottom up properties), different implementations of unification, different methods of compiling the rules, all are able to run on exactly the same grammars yielding the identical results.

The three implementations of the PATR-II system currently in operation at SRI are as follows:

- An INTERLISP version for the DEC-2060 using a variant of the Cocke-Kasami-Younger parsing algorithm and the KIMMO morphological analyzer [Karttunen, 83], and a limited programming environment.
- A ZETALISP version for the Symbolics 3600 using a left-corner parsing algorithm and the KIMMO morphological analyzer, with an extensive programming environment (due primarily to Mabry Tyson) that includes incremental compilation, multiple window debugging facilities, tracing, and an integrated editor.
- A Prolog version (DEC-10 Prolog) running on the DEC-2060 by Fernando Pereira, designed primarily as a testbed for experimentation with efficient structure-sharing DAG unification algorithms, and incorporating an Earley-style parsing algorithm.

In addition, Lauri Karttunen and his students at the University of Texas have implemented a system based on PATR-II but with several interesting extensions, including disjunction and negation in the graph structures [Karttunen, 84]. These extensions will undoubtedly be integrated into the SRI systems and formal semantics for them are being pursued.

1.3. Conclusion

The PATR-II formalism was designed as a computer language for encoding linguistic information. The design was influenced by current theory and practice in computer science, and especially in the areas of programming language design and semantics. The formalism is *simple* (consisting of just one primitive operation, unification), *powerful* (although it can be constrained to be decidable), *mathematically well-founded* (with a complete denotational semantics), *flexible* (as demonstrated by its ability to model analyses in GPSG, LFG, DCG and other formalisms), *modular* (because of its higher-level notational devices such as templates and lexical rules), *declarative* (yielding order-independence of operations), and *implementable* (as demonstrated by three quite dissimilar implemented systems and one highly developed programming environment).

As we have emphasized herein, PATR-II seems to represent a convergence of techniques from several domains—computer science, programming language design, natural language processing and linguistics. Its positioning at the center of these trends arises, however, not from the admixture of many discrete techniques, but rather from the application of a single simple yet powerful concept to the encoding of linguistic information.

Chapter 2.

Features and Values

This chapter was written by Lauri Karttunen of the University of Texas at Austin, the Artificial Intelligence Center, SRI International and the Center for the Study of Language and Information, Stanford University.

Abstract

The paper discusses the linguistic aspects of a new general purpose facility for computing with features. The program was developed in connection with the course I taught at the University of Texas in the fall of 1983. It is a generalized and expanded version of a system that Stuart Shieber originally designed for the PATR-II project at SRI in the spring of 1983 with later modifications by Fernando Pereira and me. Like its predecessors, the new Texas version of the "DG (directed graph)" package is primarily intended for representing morphological and syntactic information but it may turn out to be very useful for semantic representations too.

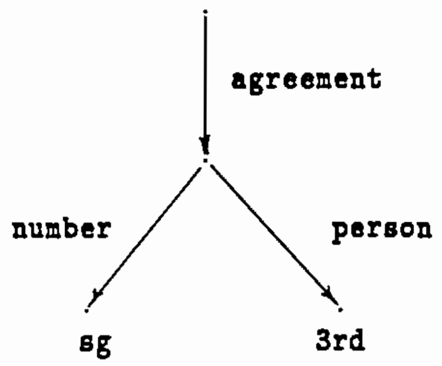
2.1. Introduction

Most schools of linguistics use some type of feature notation in their phonological, morphological, syntactic, and semantic descriptions. Although the objects that appear in rules and conditions may have atomic names, such as "k," "NP," "Subject," and the like, such high-level terms typically stand for collections of features. Features, in this sense of the word, are usually thought of as attribute-value pairs: [person: 1st], [number: sg], although singleton features are also admitted in some theories. The values of phonological and morphological features are traditionally atomic; e.g. 1st, 2nd, 3rd; they are often binary: +, -. Most current theories also allow features that have complex values. A complex value is a collection of features. For example, for some languages it may be useful to postulate a feature called "agreement" whose value is a feature bundle that specifies values for "number" and "person."

$$\begin{bmatrix}\text{agreement:} & \begin{bmatrix}\text{person: 3rd}\\ \text{number: sg}\end{bmatrix}\end{bmatrix}$$

Lexical Functional Grammar (LFG) [Kaplan and Bresnan, 83], Unification Grammar (UG) [Kay, 79], Generalized Phrase Structure Grammar (GPSG) [Gazdar and Pullum, 82], among others, use complex features.

As shown above, features are typically represented as matrices. Another way to represent features is to think of them as directed graphs where values correspond to nodes and attributes to vectors:



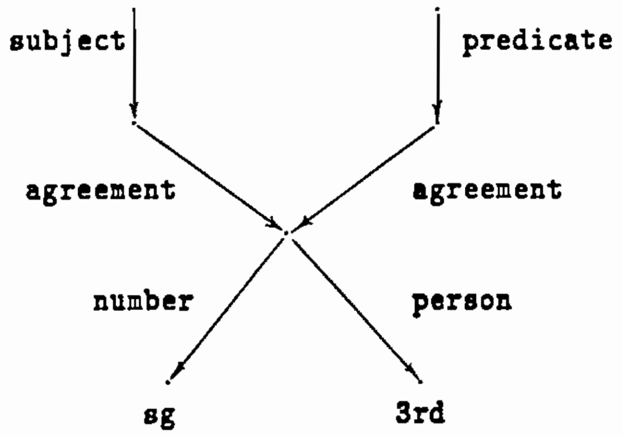
In graphs of this sort, values are reached by traversing paths of attribute names. We use angle brackets to mark expressions that designate paths. With that convention, the above graph can also be represented as a set of equations:

```
<agreement number> = sg
<agreement person> = 3rd
```

Such equations also provide a convenient way to express conditions on features. This idea lies at the heart of UG, LFG, and the PATR-II grammar for English [Shieber, *et al.*, 83] constructed at SRI. For example, the equation

```
<subject agreement> = <predicate agreement>
```

states that subject and predicate have the same value for agreement. In graph terms, this corresponds to a lattice where two vectors point to the same node:



In a case like this, the values of the two paths have been "unified." To indicate unification in feature matrices one needs some convention such as the connecting lines in LFG-style functional structures (f-structures) [Kaplan and Bresnan, 83] to show that the values of two attributes are the very same entity and not identical twins.

A third way to view collections of features is to think of them as partial functions that assign values to attributes [Sag *et.al.*, 84].

2.2. Unification and Generalization

Several related grammar formalisms (UG, LFG, PATR-II, and GPSG) now exist that are based on a very similar conception of features and use unification as their basic operation. In this section we survey briefly some properties of unification.

Because feature matrices (lattice nodes) are sets of attribute-value pairs, unification is closely related to the operation of forming a union of two sets. But there are two important differences:

- unification can fail
- unification changes structure

While set union is is an operation that always yields something– at least the null set, unification may fail to produce a value. When it fails the operands remain unchanged; when it succeeds, the operands are permanently altered in the process. They become the same object. Like set union, unification is associative and commutative. The result of unifying three or more graphs in pairs with one another does not depend on the order in which the operations are performed. They all become the same graph at the end. Another important characteristic is that unification can be applied to values that are still indeterminate. When a value is eventually assigned to one of two unified paths, the other path acquires it at the same time.

In trying to unify two structures, A and B, one proceeds as follows. If A and B contain the same attribute but have incompatible values for it, they cannot be unified. When A and B are merged, every attribute that appears only in one of the structures is included in (Unify A B) with its original value. If some attribute appears both in A and B, then the value of that attribute in (Unify A B) is the unification of the two values. For example, if A and B are as shown below

$$\mathbf{A} = \begin{bmatrix} \texttt{agreement:} & \begin{bmatrix} \texttt{number: pl} \end{bmatrix} \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \texttt{agreement:} & \begin{bmatrix} \texttt{person: 3rd} \end{bmatrix} \\ \texttt{case: nominative} & \end{bmatrix}$$

the effect of (Unify A B) is to make them identical to

$$\texttt{(Unify A B)} = \begin{bmatrix} \texttt{agreement:} & \begin{bmatrix} \texttt{person: 3rd} \\ \texttt{number: pl} \end{bmatrix} \\ \texttt{case: nominative} & \end{bmatrix}$$

Simple cases of grammatical concord, such as number, case and gender agreement between determiners and nouns in many languages, can be expressed straight-forwardly by stating that the values of these features must unify.

Besides unification, there are other possible operations on features that may have linguistic relevance. Generalization is perhaps the the most likely candidate for that status. It is closely related to set intersection. The generalization of two simple matrices A and B consists of the attribute-value pairs that A and B have in common. If A and B have the same attribute with different values, the value of that attribute in (Generalize A B) is the generalization of the original values. Unlike unification, generalization cannot fail.

For example,

$$\mathbf{A} = \begin{bmatrix} \texttt{agreement:} & \begin{bmatrix} \texttt{person: 2nd} \\ \texttt{number: sg} \end{bmatrix} \\ \texttt{case: nominative} & \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} \texttt{agreement:} & \begin{bmatrix} \texttt{gender: masc} \\ \texttt{person: 3rd} \\ \texttt{number: sg} \end{bmatrix} \\ \texttt{case: genitive} & \end{bmatrix}$$

$$\textbf{(Generalize A B)} = \begin{bmatrix} \texttt{agreement:} & \begin{bmatrix} \texttt{number: sg} \end{bmatrix} \end{bmatrix}$$

Generalization may to be a useful notion for expressing how number and gender agreement works in coordinate noun phrases (see [Gazdar and Pullum, 82]). We discuss that possibility in Section 2.5 below.

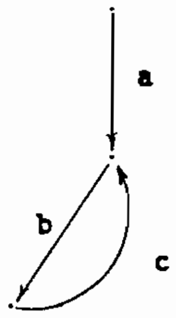
While there is substantial agreement on what "unification" means, the same is not true of generalization. For example, it is not clear what the result of generalization should be in a case where A and B have the same attribute with conflicting values. Is it indeterminate or a disjunction of the two incompatible values? If generalization lives up to its promise and turns out to have interesting linguistic applications, technical issues of this sort can perhaps be resolved on linguistic grounds.

2.3. Limitations of Some Current Formalisms

Most current grammar formalisms for features have certain built-in limitations. Three are relevant here:

- no cyclic structures
- no negation
- no disjunction.

The prohibition against cyclicity rules out structures that contain circular paths, as in the following example.



Here the path <a b c> folds back onto itself, that is, <a> = <a b c>. It is not clear whether such descriptions should be ruled out on theoretical grounds. Whatever the case might be, current implementations of LFG, UG, or GPSG with which I am familiar do not support them.

The prohibition against negation makes it impossible to characterize a feature by saying that it does NOT have such and such a value. Except for LFG, none the above theories allows specifications such as the following. We use the symbol "-" to mean 'not.'

$$\left[\texttt{case: -dat}\right]$$

$$\left[\texttt{agreement:}\ \left[-\begin{bmatrix}\texttt{number: sg}\\ \texttt{person: 3rd}\end{bmatrix}\right]\right]$$

The first statement says that case is "not dative," the second says that the value of agreement is "anything but 3rd person singular." In case of LFG, the formalism allows negation in functional descriptions (f-descriptions) but not in f-structures that are derived from them.

Not allowing disjunctive specifications rules out matrices of the following sort. We indicate disjunction by enclosing the alternative values in {}.

$$\left[\begin{array}{ll} \text{agreement:} & \left\{\begin{array}{l} \left[\begin{array}{l}\text{number: sg}\\ \text{gender: fem}\end{array}\right] \\ \left[\text{number: pl}\right] \end{array}\right\} \\ \text{case: } \{\text{nom acc}\} & \end{array}\right]$$

The first line describes the value of case as being "either nominative or accusative." The value for agreement is given as "either feminine singular or plural." Among the theories mentioned above, only Kay's UG allows disjunctive feature specifications in its formalism. In LFG, disjunctions are allowed in f-descriptions but there are no disjunctive values in f-structures.

Of the three limitations, the first one may be theoretically justified since it has not been shown that there are phenomena in natural languages that involve circular structures (cf. [Kaplan and Bresnan, 83], p. 281). PATR-II at SRI and its expanded version at the University of Texas allow such structures for practical reasons because they tend to arise, mostly inadvertently, in the course of grammar construction and testing. An implementation that does not handle unification correctly in such cases is too fragile to use.

The other two restrictions are linguistically unmotivated. There are many cases, especially in morphology, in which the most natural feature specifications are negative or disjunctive. In fact, the examples given above all represent such cases.

The first example, [case: -dat], arises in the plural paradigm of words like "Kind" *child* in German. Such words have two forms in the plural: "Kinder" and "Kindern." The latter is used only in the plural dative, the former in the other three cases (nominative, genitive, accusative). If we accept the view that there should be just one rather than three entries for the plural suffix "-er", we have the choice between

$$\textbf{-er} = \begin{bmatrix} \texttt{number: pl} \\ \texttt{case: \{nom gen acc\}} \end{bmatrix}$$

$$\textbf{-er} = \begin{bmatrix} \texttt{number: pl} \\ \texttt{case: -dat} \end{bmatrix}$$

The second alternative seems preferrable given the fact that there is, in this particular declension, a clear two-way contrast. The marked dative is in opposition with an unmarked form representing all the other cases.

The second example is from English. Although the features "number" and "person" are both clearly needed in English verb morphology, most verbs are very incompletely specified for them. In fact, the present tense paradigm of all regular verbs just has two forms of which one represents the 3rd person singular ("walks") and the other ("walk") is used for all other persons. Thus the most natural characterization for "walk" is that it is not 3rd person singular. The alternative is to say, in effect, that "walk" in the present tense has five different interpretations.

The system of articles in German provides many examples that call for disjunctive feature specifications. The article "die," for example, is used in the nominative and accusative cases of singular feminine nouns and all plural nouns. The entry given above succinctly encodes exactly this fact.

There are many cases where disjunctive specifications seem necessary for reasons other than just descriptive elegance. An item which is in underspecified with respect to some attribute can typically appear in constructions that require conflicting values for that attribute. For example, in German [Eisenberg, 73] noun phrases like:

> des Dozenten (gen sg) ***the docent's***
> der Dozenten (gen pl) ***the docents'.***

can blend as in

der Antrag des oder der Dozenten
the petition of the docent or docents.

This is not possible when the noun is overtly marked for number, as in the case of "des Professors" (gen sg) and "der Professoren" (gen pl):

*der Antrag des oder der Professors
*der Antrag des oder der Professoren
the petition of the professor or professors

In the light of such cases, it seems reasonable to assume that there is a single form, "Dozenten," which has a disjunctive feature specification, instead of postulating several fully specified, homonymous lexical entries. It is obvious that the grammaticality of the example crucially depends on the the fact that "Dozenten" is not definitely singular or definitely plural but can be either.

Similar examples are easy to find in many other languages, arguably, even in English:

? this and those sheep

In French the clitic "me" is either accusative or dative. Because of that, it can simultaniously function both as direct and indirect object of two conjoined verb phrases, as in

Jean m'a frappe et donne des coups de pied
John hit and kicked me ("hit and gave me kicks").

Although these examples provide convincing evidence for disjunctive feature values, they are at the same time extremely problematic. Consider the German case. Like "this" and "those" in English, the two German articles are in conflict with one another. The features of the noun are compatible with either one of the two sets of article features but unification is not possible. The operation would fail because it in part involves unifying the two incompatible sets. The only obvious alternative is to say that the merge of "Dozenten" with "des" and "der" is done using two identical copies of the noun features. It works technically but it raises many unsettling general questions about unification.

2.4. Unification with Disjunctive and Negative Feature Specifications

I sketch here briefly how the basic unification procedure can be modified to deal with negation and to admit disjunctive values. These ideas have been implemented in the new Texas version of the PATR-II system for features. (I am much indebted to Fernando Pereira for his advice on this topic.)

There are no negative values *per se*, only negative constraints. A negative constraint attached to some structure A limits the class of structures that can be unified with A. It is not part of the content of A itself. Negative constraints are created by the following operation. If A and B are distinct, i.e. contain a different value for some feature, then (Negate A B) does nothing to them. If A and B are not distinct, A is marked with -B and B with -A. These constraints prevent the two nodes from ever becoming alike. When A is unified with C, unification succeeds only if the result is distinct from B. The result of (Unify A C) has to satisfy all the negative constraints of both A and C. It inherits all negative constraints of A and C that could fail in some later unification.

Disjunction is more complicated. Suppose A, B and C are all simple atomic values. In this situation C unifies with {A B} just in case it is identical to one or the other of the disjuncts. The result is C. Now suppose that A, B, and C are all complex. Furthermore, let us suppose that A and B are distinct but C is compatible with both of them as in the following:

$$A = \begin{bmatrix} \text{gender: fem} \\ \text{number: sg} \end{bmatrix}$$

$$B = \begin{bmatrix} \text{number: pl} \end{bmatrix}$$

$$C = \begin{bmatrix} \text{case: acc} \end{bmatrix}$$

What should be the result of (Unify {A B} C)? Because A and

B are incompatible, we cannot actually unify C with both of them. That operation would fail. Because there is no basis for choosing one, both alternatives have to be left open. Nevertheless, we need to take note of the fact that either A or B is to be unified with C. This is done by making the result a complex disjunction.

```
C' =  {(A C)  (B C)}  .
```

The new value of C, C', is a disjunction of tuples which can be, but have not yet been unified. Tuples (A C) and (B C) are sets of compatible structures. At least one of the tuples in the complex disjunction must always remain consistent regardless of what happens to A, B, and C later. After the first unification we can still unify A with any structure that it is compatible with, such as:

```
D =  [case: nom]       .
```

If this happens, then the tuple (A C) is no longer consistent. A side effect of A becoming

```
A' =    [gender: fem]
        |number: sg |
        [case: nom  ]
```

is that C' simultaniously reduces to {(B C)}. Since there is now only one viable alternative left, B and C can at this point be unified. The original result from (Unify {A B} C) now reduces to the same as (Unify B C).

```
C'' = {(B C)} =   [number: pl]        .
                  [case: acc ]
```

As the example shows, once C is unified with {A B}, A and B acquire a "positive constraint." All later unifications involving them must keep at least one of the two pairs (A C), (B C) unifieable. If at some later point one of the two tuples becomes inconsistent, the members of the sole remaining tuple finally can and should be unified. When that has happened, the positive constraint on A and B can also be discarded. A more elaborate example of this sort is given in the Appendix.

Essentially the same procedure also works for more complicated cases. For example, unification of {A B} with {C D} yields {(A C) (A D) (B C) (B D)} assuming that the two values in each tuple are compatible. Any pairs that could not be unified are left out. The complex disjunction is added as a positive constraint to all of the values that appear in it. The result of unifying {(A C) (B C)} with {(D F) (E F)} is {(A C D F) (A C E F) (B C D F) (B C E F)}, again assuming that no alternative can initially be ruled out.

As for generalization, things are considerably simpler. The result of (Generalize A B) inherits both negative and positive constraints of A and B. This follows from the fact that the generalization of A and B is the maximal subgraph of A and B that will unify with either one them. Consequently, it is subject to any constraint that affects A or B. This is analogous to the fact that, in set theory,

$$(A - C) \cap (B - D) = (A \cap B) - (C \cup D) \quad .$$

In our current implementation, negative constraints are dropped as soon as they become redundant as far as unification is concerned. For example, when [case: acc] is unified with with .[case: -dat], the resulting matrix is simply [case: acc]. The negative constraint is eliminated since there is no possibility that it could ever be violated later. This may be a wrong policy. It has to be modified to make generalization work as proposed in Section 2.5 for structures with negative constraints. If generalization is defined as we have suggested above, negative constraints must always be kept because they never become redundant for generalization.

When negative or positive constraints are involved, unification obviously takes more time. Nevertheless, the basic algorithm remains pretty much the same. Allowing for constraints does not significantly reduce the speed at which values that do not have any get unified in the Texas implementation.

In the course of working on the project, I gained one insight that perhaps should have been obvious from the very beginning: the problems that arise in this connection are very similar to those that come up in logic programming. One can actually use the feature system for certain kind of inferencing. For example, let Mary, Jane, and John have the following values:

```
Mary = [hair: blond]

Jane = [hair: dark]

John = [sister: {Jane Mary}]
```

If we now unify John with

```
[sister: [eyes: blue]] ,
```

both Jane and Mary get marked with the positive constraint that at least one of them has blue eyes. Suppose that we now learn that Mary has green eyes. This immediately gives us more information about John and Jane as well. Now we know that Jane's eyes are blue and that she definitely is John's sister. The role of positive constraints is to keep track of partial information in such a way that no inconsistencies are allowed and proper updating is done when more things become known.

2.5. Future prospects: Agreement in Coordinate Structures

One problem of long standing for which the present system may provide a simple solution is person agreement in coordinate noun phrases. The conjunction of a 1st person pronoun with either 2nd or 3rd person pronoun invariably yields 1st person agreement. "I and you" is equivalent to "we," as far as agreement is concerned. When a second person pronoun is conjoined with a third person NP, the resulting conjunction has the agreement properties of a second person pronoun. Schematically:

> 1st + 2nd = 1st (*I and you* talked about *ourselves.*)
> 3rd + 1st = 1st (*She and I* talked about *ourselves.*)
> 2nd + 3rd = 2nd. (*You and he* talked about *your-selves.*)

Sag, Gazdar, Wasow, and Weisler [84] propose a solution which is based on the idea of deriving the person feature for a coordinate noun phrase by generalization (intersection) from the person features of its heads. It is obvious that the desired effect can be obtained in any feature system that uses the fewest features to mark 1st person, some additional feature for 2nd person, and yet another for 3rd person. Because generalization of 1st and 2nd, for example, yields only the features that two have in common, the one with fewest features wins.

Any such solution can probably be implemented easily in the framework outlined above. However, this proposal has one very counterintuitive aspect: markedness hierarchy is the reverse of what traditionally has been assumed. Designating something as 3rd person requires the greatest number of feature specifications. In the Sag *et al.* system, 3rd person is the most highly marked member and 1st person the least marked member of the trio. Traditionally, 3rd person has been regarded as the unmarked case.

In our system, there is a rather simple solution under which the value of person feature in coordinate NPs is derived by generalization, just as Sag *et al.* propose, which nevertheless preserves the traditional view of markedness. The desired result can be obtained by using negative constraints rather than additional features for establishing a markedness hierarchy. In addition to regular features we allow lexical entries to contain negative constraints. The following feature specifications for personal pronouns have the effect we are are seeking. First the regular (positive) features:

$$\textbf{1st} = \left[\begin{array}{l}\texttt{conversant: +}\\ \texttt{speaker: +}\end{array}\right]$$

$$\textbf{2nd} = \left[\begin{array}{l}\texttt{conversant: +}\\ \texttt{speaker: -}\end{array}\right]$$

$$\textbf{3rd} = \left[\begin{array}{l}\texttt{conversant: -}\\ \texttt{speaker: -}\end{array}\right]$$

The associated negative constraints are:

$$\textbf{1st} = \left[- \left[\begin{array}{l}\texttt{conversant: -}\\ \texttt{speaker: -}\end{array}\right]\right]$$

$$\textbf{2nd} = \left[- \left[\texttt{conversant: -}\right]\right]$$

$$\textbf{3rd} = \textbf{(no constraints)}$$

Assuming that generalization with negative constraints works as indicated above, i.e. negative constraints are always inherited, it immediately follows that the generalization of 1st person with any other person is compatible with only 1st person and that 2nd person wins over 3rd when they are combined. The results are as follows.

$$\textbf{1st + 2nd} = \begin{bmatrix} \texttt{conversant: -} \\ - \begin{bmatrix} \texttt{conversant: -} \\ \texttt{speaker: -} \end{bmatrix} \end{bmatrix}$$

$$\textbf{1st + 3rd} = \begin{bmatrix} - \begin{bmatrix} \texttt{conversant: -} \\ \texttt{speaker: -} \end{bmatrix} \end{bmatrix}$$

$$\textbf{2nd + 3rd} = \begin{bmatrix} \texttt{speaker: -} \\ - \begin{bmatrix} \texttt{conversant: -} \end{bmatrix} \end{bmatrix}$$

Note that the proper part of 1st+2nd excludes 3rd person. It is compatible with both 1st and 2nd person but the negative constraint rules out the latter one. In the case of 1st+3rd, the negative constraint is compatible with 1st person but incompatible with 2nd and 3rd. In the last case, the specification [speaker: -] rules out 1st person and the negative constraint -[conversant: -] eliminates 3rd person.

When negative constraints are counted in, 1st person is the most and 3rd person the least marked member of the three. In that respect, the proposed analysis is in line with traditional views on markedness. Another relevant observation is that the negative constraints on which the result crucially depends are themselves not too unnatural. In effect, they say of 1st person that it is "neither 2nd nor 3rd" and that 2nd person is "not 3rd."

It will be interesting to see whether other cases of markedness can be analyzed in the same way.

2.6. Acknowledgements

This research was made possible in part by a grant from the Alfred P. Sloan Foundation to the Center for Cognitive Science at the University of Texas at Austin.

I am indebted to Martin Kay for introducing me to unifica-

tion and to Fernando Pereira, Stuart Shieber, Remo Pareschi, and Annie Zaenen for many insightful suggestions on the project.

Appendix A: Some Examples of Unification

(These examples were produced using the Texas version of the DG package.)

$$\textbf{die} = \left[\begin{array}{ll} \texttt{infl:} & \left[\begin{array}{ll} \texttt{case:} & \{\texttt{nom acc}\} \\ \texttt{agr:} & \left\{\begin{array}{l} \left[\begin{array}{ll}\texttt{gender:} & \texttt{fem}\\ \texttt{number:} & \texttt{sg}\end{array}\right] \\ \left[\texttt{number: pl}\right]\end{array}\right\}\end{array}\right]\end{array}\right]$$

$$\textbf{Kinder} = \left[\begin{array}{ll} \texttt{infl:} & \left[\begin{array}{ll} \texttt{case:} & [\texttt{-dat}] \\ \texttt{agr:} & \left[\begin{array}{ll}\texttt{gender:} & \texttt{neut}\\ \texttt{number:} & \texttt{pl}\end{array}\right]\end{array}\right]\end{array}\right]$$

$$\textbf{die Kinder} = \left[\begin{array}{ll} \texttt{infl:} & \left[\begin{array}{ll} \texttt{case:} & \{\texttt{nom acc}\} \\ \texttt{agr:} & \left[\begin{array}{ll}\texttt{gender:} & \texttt{neut}\\ \texttt{number:} & \texttt{pl}\end{array}\right]\end{array}\right]\end{array}\right]$$

$$\textbf{den} = \left[\begin{array}{ll} \texttt{infl:} & \left\{\begin{array}{l} \left[\begin{array}{ll} \texttt{case:} & \texttt{acc} \\ \texttt{agr:} & \left[\begin{array}{ll}\texttt{gender:} & \texttt{masc}\\ \texttt{number:} & \texttt{sg}\end{array}\right]\end{array}\right] \\ \left[\begin{array}{ll} \texttt{case:} & \texttt{dat} \\ \texttt{agr:} & [\texttt{number: pl}]\end{array}\right]\end{array}\right\}\end{array}\right]$$

den Kinder = *FAILS*

$$\textbf{den Kindern} = \left[\begin{array}{ll} \texttt{infl:} & \left[\begin{array}{ll} \texttt{case:} & \texttt{dat} \\ \texttt{agr:} & \left[\begin{array}{ll}\texttt{gender:} & \texttt{neut}\\ \texttt{number:} & \texttt{pl}\end{array}\right]\end{array}\right]\end{array}\right]$$

$$\mathbf{I} = \begin{bmatrix} \text{infl:} & \begin{bmatrix} \text{case: nom} & \\ \text{agr:} & \begin{bmatrix} \text{number: sg} \\ \text{person: 1st} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\mathbf{he} = \begin{bmatrix} \text{infl:} & \begin{bmatrix} \text{case: nom} & \\ \text{agr:} & \begin{bmatrix} \text{gender: masc} \\ \text{number: sg} \\ \text{person: 3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\mathbf{do} = \begin{bmatrix} \text{infl:} & \begin{bmatrix} \text{tense: present} & \\ \text{agr:} & \begin{bmatrix} - \begin{bmatrix} \text{number: sg} \\ \text{person: 3rd} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$\mathbf{I\ do} = \begin{bmatrix} \text{infl:} & \begin{bmatrix} \text{tense: present} & \\ \text{case: nom} & \\ \text{agr:} & \begin{bmatrix} \text{number: sg} \\ \text{person: 1st} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

he do = *FAILS*

$$\mathbf{x} = \left\{ \begin{bmatrix} \text{a: } - \\ \text{b: } + \end{bmatrix} \begin{bmatrix} \text{b: } - \\ \text{c: } + \end{bmatrix} \begin{bmatrix} \text{a: } + \\ \text{c: } - \end{bmatrix} \right\} \qquad \mathbf{y} = \left\{ \begin{bmatrix} \text{a: } + \\ \text{b: } - \end{bmatrix} \begin{bmatrix} \text{b: } + \\ \text{c: } + \end{bmatrix} \right\} \qquad \mathbf{z} = \begin{bmatrix} \text{a: } + \\ \text{c: } + \end{bmatrix}$$

(Unify x y)

$$= \left\{ \begin{array}{l} \left(\begin{array}{l} \begin{bmatrix} \text{b:} & - \\ \text{c:} & + \end{bmatrix} \\ {}^{1}\begin{bmatrix} \text{a:} & + \\ \text{b:} & - \end{bmatrix} \end{array} \right) \\ \left(\begin{array}{l} \begin{bmatrix} \text{a:} & + \\ \text{c:} & - \end{bmatrix} \\ \langle 1 \rangle \end{array} \right) \\ \left(\begin{array}{l} \begin{bmatrix} \text{a:} & - \\ \text{b:} & + \end{bmatrix} \\ \begin{bmatrix} \text{b:} & + \\ \text{c:} & + \end{bmatrix} \end{array} \right) \end{array} \right\}$$

(Unify (Unify x y) z)

$$= \begin{bmatrix} \text{a:} & + \\ \text{b:} & - \\ \text{c:} & + \end{bmatrix}$$

Chapter 3.

The Semantics of Grammar Formalisms Seen as Computer Languages

This chapter was written by Fernando C. N. Pereira and Stuart M. Shieber of the Artificial Intelligence Center, SRI International and the Center for the Study of Language and Information, Stanford University.

Abstract[1]

The design, implementation, and use of grammar formalisms for natural language have constituted a major branch of computational linguistics throughout its development. By viewing grammar formalisms as just a special case of computer languages, we can take advantage of the machinery of denotational semantics to provide a precise specification of their meaning. Using Dana Scott's domain theory, we elucidate the nature of the feature systems used in augmented phrase-structure grammar formalisms, in particular those of recent versions of generalized phrase structure

[1]The research reported in this paper has been made possible by a gift from the Systems Development Foundation.

grammar, lexical functional grammar and PATR-II, and provide a denotational semantics for a simple grammar formalism. We find that the mathematical structures developed for this purpose contain an operation of feature generalization, not available in those grammar formalisms, that can be used to give a partial account of the effect of coordination on syntactic features.

3.1. Introduction

The design, implementation, and use of grammar formalisms for natural language have constituted a major branch of computational linguistics throughout its development. However, notwithstanding the obvious superficial similarity between designing a grammar formalism and designing a programming language, the design techniques used for grammar formalisms have almost always fallen short with respect to those now available for programming language design.

Formal and computational linguists most often explain the effect of a grammar formalism construct either by example or through its actual operation in a particular implementation. Such practices are frowned upon by most programming-language designers; they become even more dubious if one considers that most grammar formalisms in use are based either on a context-free skeleton with augmentations or on some closely related device (such as ATNs), consequently making them obvious candidates for a declarative *semantics*[2] extended in the natural way from the declarative semantics of context-free grammars.

The last point deserves amplification. Context-free grammars possess an obvious declarative semantics in which nonterminals represent sets of strings and rules represent n-ary relations over strings. This is brought out by the reinterpretation familiar from

[2]This use of the term "semantics" should not be confused with the more common usage denoting that portion of a grammar concerned with the meaning of object sentences. Here we are concerned with the meaning of the metalanguage.

formal language theory of context-free grammars as polynomials over concatenation and set union. The grammar formalisms developed from the definite-clause subset of first order logic are the only others used in natural-language analysis that have been accorded a rigorous declarative semantics—in this case derived from the declarative semantics of logic programs [Colmerauer, 78; Pereira and Warren, 80; Pereira, 81].

Much confusion, wasted effort, and dissension have resulted from this state of affairs. In the absence of a rigorous semantics for a given grammar formalism, the user, critic, or implementer of the formalism risks misunderstanding the intended interpretation of a construct, and is in a poor position to compare it to alternatives. Likewise, the inventor of a new formalism can never be sure of how it compares with existing ones. As an example of these difficulties, two simple changes in the implementation of the ATN formalism, the addition of a well-formed substring table and the use of a bottom-up parsing strategy, required a rather subtle and unanticipated reinterpretation of the register-testing and -setting actions, thereby imparting a different meaning to grammars that had been developed for initial top-down backtrack implementation [Woods, *et al.*, 76].

Rigorous definitions of grammar formalisms can and should be made available. Looking at grammar formalisms as just a special case of computer languages, we can take advantage of the machinery of *denotational semantics* [Stoy, 77] to provide a precise specification of their meaning. This approach can elucidate the structure of the data objects manipulated by a formalism and the mathematical relationships among various formalisms, suggest new possibilities for linguistic analysis (the subject matter of the formalisms), and establish connections between grammar formalisms and such other fields of research as programming-language design and theories of abstract data types. This last point is particularly interesting because it opens up several possibilities—among them that of imposing a *type discipline* on the use of a formalism, with all the attendant advantages of compile-time error checking, modularity, and optimized compilation techniques for grammar rules, and that of relating grammar formalisms to other knowledge representation languages [Ait-Kaci, 83].

As a specific contribution of this study, we elucidate the nature of the feature systems used in augmented phrase-structure grammar formalisms, in particular those of recent versions of generalized phrase structure grammar (GPSG) [Gazdar and Pullum, 82; Sag *et al.*, 84], lexical-functional grammar (LFG) [Kaplan and Bresnan, 83] and PATR-II [Shieber, *et al.*, 83; Shieber, 84]; we find that the mathematical structures developed for this purpose contain an operation of *feature generalization*, not available in those grammar formalisms, that can be used to give a partial account of the effect of coordination on syntactic features.

Just as studies in the semantics of programming languages start by giving semantics for simple languages, so we will start with simple grammar formalisms that capture the essence of the method without an excess of obscuring detail. The present enterprise should be contrasted with studies of the generative capacity of formalisms using the techniques of formal language theory. First, a precise definition of the semantics of a formalism is a prerequisite for such generative-capacity studies, and this is precisely what we are trying to provide. Second, generative capacity is a very coarse gauge: in particular, it does not distinguish among different formalisms with the same generative capacity that may, however, have very different semantic accounts. Finally, the tools of formal language theory are inadequate to describe at a sufficiently abstract level formalisms that are based on the simultaneous solution of sets of constraints [Kay, 79; Marcus, *et al.*, 82]. An abstract analysis of those formalisms requires a notion of partial information that is precisely captured by the constructs of denotational semantics.

3.2. Denotational Semantics

In broad terms, denotational semantics is the study of the connection between programs and mathematical entities that represent their input-output relations. For such an account to be useful, it must be *compositional*, in the sense that the meaning of a program is developed from the meanings of its parts by a fixed set of mathematical operations that correspond directly to the ways in which the parts participate in the whole.

For the purposes of the present work, denotational semantics will mean the semantic domain theory initiated by Scott and Strachey [Stoy, 77]. In accordance with this approach, the meanings of programming language constructs are certain partial mappings between objects that represent partially specified data objects or partially defined states of computation. The essential idea is that the meaning of a construct describes what information it adds to a partial description of a data object or of a state of computation. Partial descriptions are used because computations in general may not terminate and may therefore never produce a fully defined output, although each individual step may be adding more and more information to a partial description of the undeliverable output.

Domain theory is a mathematical theory of considerable complexity. Potential nontermination and the use of functions as "first-class citizens" in computer languages account for a substantial fraction of that complexity. If, as is the case in the present work, neither of those two aspects comes into play, one may be justified in asking why such a complex apparatus is used. Indeed, both the semantics of context-free grammars mentioned earlier and the semantics of logic grammars in general can be formulated using elementary set theory [Harrison, 78; van Emden and Kowalski, 76].

However, using the more complex machinery may be beneficial for the following reasons:

- *Inherent partiality*: many grammar formalisms operate in terms of constraints between elements that do not fully specify all the possible features of an element.
- *Technical economy*: results that require laborious construc-

tions without utilizing domain theory can be reached trivially by using standard results of the theory.

- *Suggestiveness*: domain theory brings with it a rich mathematical structure that suggests useful operations one might add to a grammar formalism.
- *Extensibility*: unlike a domain-theoretic account, a specialized semantic account, say in terms of sets, may not be easily extended as new constructs are added to the formalism.

3.3. The Domain of Feature Structures

We will start with an abstract denotational description of a simple feature system which bears a close resemblance to the feature systems of GPSG, LFG and PATR-II, although this similarity, because of its abstractness, may not be apparent at first glance. Such feature systems tend to use data structures or mathematical objects that are more or less isomorphic to directed graphs of one sort or another, or, as they are sometimes described, partial functions. Just what the relation is between these two ways of viewing things will be explained later. In general, these graph structures are used to encode linguistic information in the form of attribute-value pairs. Most importantly, partial information is critical to the use of such systems—for instance, in the variables of definite clause grammars [Pereira and Warren, 80] and in the GPSG analysis of coordination [Sag *et.al.*, 84]. That is, the elements of the feature systems, called *feature structures* (alternatively, feature bundles, f-structures [Kaplan and Bresnan, 83], or terms) can be partial in some sense. The partial descriptions, being in a domain of attributes and complex values, tend to be equational in nature: some feature's value is equated with some other value. Partial descriptions can be understood in one of two ways: either the descriptions represent sets of fully specified elements of an underlying domain or they are regarded as participating in a relationship of partiality with respect to each other. We will hold to the latter view here.

What are feature structures from this perspective? They are repositories of information about linguistic entities. In domain-theoretic terms, the underlying domain of feature structures F is a recursive domain of partial functions from a set of *labels* L (features, attribute names, attributes) to complex values or primitive atomic values taken from a set C of *constants*. Expressed formally, we have the domain equation

$$F = [L \rightarrow F] + C \quad .$$

The solution of this domain equation can be understood as a set of trees (finite or infinite) with branches labeled by elements of L, and with other trees or constants as nodes. The branches $l_1, \ldots, l_m$ from a node n point to the values $n(l_1), \ldots, n(l_m)$ for which the node, as a partial function, is defined.

3.4. The Domain of Descriptions

What the grammar formalism does is to talk *about* F, not *in* F. That is, the grammar formalism uses a domain of descriptions of elements of F. From an intuitive standpoint, this is because, for any given phrase, we may know facts about it that cannot be encoded in the partial function associated with it.

A partial description of an element n of F will be a set of equations that constrain the values of n on certain labels. In general, to describe an element $x \in F$ we have equations of the following forms:

$$\begin{array}{rcl} (\cdots(x(l_{i_1}))\cdots)(l_{i_m}) & = & (\cdots(x(l_{j_1}))\cdots)(l_{j_n}) \\ (\cdots(x(l_{i_1}))\cdots)(l_{i_m}) & = & c_k \quad , \end{array}$$

which we prefer to write as

$$\begin{array}{rcl} \langle l_{i_1} \cdots l_{i_m} \rangle & = & \langle l_{j_1} \cdots l_{j_m} \rangle \\ \langle l_{i_1} \cdots l_{i_m} \rangle & = & c_k \end{array}$$

with x implicit. The terms of such equations are constants $c \in C$ or *paths* $\langle l_{i_1} \cdots l_{i_m} \rangle$, which we identify in what follows with strings in L^*. Taken together, constants and paths comprise the *descriptors*.

Using Scott's *information systems* approach to domain construction [Scott, 82], we can now build directly a characterization of feature structures in terms of information-bearing elements, equations, that engender a system complete with notions of compatibility and partiality of information.

The information system $\mathbf{D}$ describing the elements of F is defined, following Scott, as the tuple

$$\mathbf{D} = \langle \mathcal{D}, \Delta, \mathrm{Con}, \vdash \rangle \quad ,$$

where $\mathcal{D}$ is a set of *propositions*, Con is a set of finite subsets of $\mathcal{D}$, the *consistent* subsets, $\vdash$ is an *entailment* relation between elements of Con and elements of $\mathcal{D}$ and Δ is a special *least informative element* that gives no information at all. We say that a subset S of $\mathcal{D}$ is *deductively closed* if every proposition entailed by a consistent subset of S is in S. The *deductive closure* $\overline{S}$ of $S \subseteq \mathcal{D}$ is the smallest deductively closed subset of $\mathcal{D}$ that contains S.

The descriptor equations discussed earlier are the propositions of the information system for feature structure descriptions. Equations express constraints among feature values in a feature structure and the entailment relation encodes the reflexivity, symmetry, transitivity and substitutivity of equality. More precisely, we say that a finite set of equations E entails an equation e if

- *Membership*: $e \in E$
- *Reflexivity*: e is Δ or $d = d$ for some descriptor d
- *Symmetry*: e is $d_1 = d_2$ and $d_2 = d_1$ is in E
- *Transitivity*: e is $d_1 = d_2$ and there is a descriptor d such that $d_1 = d$ and $d = d_2$ are in E
- *Substitutivity*: e is $d_1 = p_1 \cdot d_2$ and both $p_1 = p_2$ and $d_1 = p_2 \cdot d_2$ are in E

- *Iteration*: there is $E' \subset E$ such that $E' \vdash e$ and for all $e' \in E'$ $E \vdash e'$

With this notion of entailment, the most natural definition of the set Con is that a finite subset E of $\mathcal{D}$ is consistent if and only if it does not entail an ***inconsistent equation***, which has the form $c_1 = c_2$, with c_1 and c_2 as distinct constants.

An arbitrary subset of $\mathcal{D}$ is consistent if and only if all its finite subsets are consistent in the way defined above. The consistent and deductively closed subsets of $\mathcal{D}$ ordered by inclusion form a complete partial order or ***domain*** D, our domain of descriptions of feature structures.

Deductive closure is used to define the elements of D so that elements defined by equivalent sets of equations are the same. In the rest of this paper, we will specify elements of D by convenient sets of equations, leaving the equations in the closure implicit.

The inclusion order $\sqsubseteq$ in D provides the notion of a description being more or less specific than another. The least-upper-bound operation $\sqcup$ combines two descriptions into the least instantiated description that satisfies the equations in both descriptions, their *unification*. The greatest-lower-bound operation $\sqcap$ gives the most instantiated description containing all the equations common to two descriptions, their *generalization*.

The foregoing definition of consistency may seem very natural, but it has the technical disadvantage that, in general, the union of two consistent sets is not itself a consistent set; therefore, the corresponding operation of unification may not be defined on certain pairs of inputs. Although this does not cause problems at this stage, it fails to deal with the fact that failure to unify is not the same as lack of definition and causes technical difficulties when providing rule denotations. We therefore need a slightly less natural definition.

First we add another statement to the specification of the entailment relation:

- *Falsity*: if e is inconsistent, $\{e\}$ entails every element of $\mathcal{D}$.

That is, falsity entails anything. Next we define Con to be simply the set of all finite subsets of $\mathcal{D}$. The set Con no longer corresponds to sets of equations that are consistent in the usual equational sense.

With the new definitions of Con and $\vdash$, the deductive closure of a set containing an inconsistent equation is the whole of $\mathcal{D}$. The partial order D is now a lattice with top element $\top = \mathcal{D}$, and the unification operation $\sqcup$ is always defined and returns $\top$ on unification failure.

We can now define the *description mapping* $\delta : D \rightarrow F$ that relates descriptions to the described feature structures. The idea is that, in proceeding from a description $d \in D$ to a feature structure $f \in F$, we keep only definite information about values and discard information that only states value constraints, but does not specify the values themselves. More precisely, seeing d as a set of equations, we consider only the subset $\lfloor d \rfloor$ of d with elements of the form

$$\langle l_1 \cdots l_m \rangle = c_k \quad .$$

Each $e \in \lfloor d \rfloor$ defines an element $f(e)$ of F by the equations

$$\begin{aligned} f(e)(l_1) &= f_1 \\ &\cdots \\ f_{i-1}(l_i) &= f_i \\ &\cdots \\ f_{m-1}(l_m) &= c_k \quad , \end{aligned}$$

with each of the f_i undefined for all other labels. Then, we can define $\delta(d)$ as

$$\delta(d) = \bigsqcup_{e \in \lfloor d \rfloor} f(e) \quad .$$

This description mapping can be shown to be continuous in the sense of domain theory, that is, it has the properties that increasing information in a description leads to nondecreasing information in the described structures (*monotonicity*) and that if a sequence of descriptions approximates another description, the same condition holds for the described structures.

Note that δ may map several elements of D on to one element of F. For example, the elements given by the two sets of equations

$$\left\{ \begin{array}{lcl} \langle f\ h \rangle & = & \langle g\ i \rangle \\ \langle f\ h \rangle & = & c \end{array} \right\} \quad \left\{ \begin{array}{lcl} \langle f\ h \rangle & = & c \\ \langle g\ i \rangle & = & c \end{array} \right\}$$

describe the same structure, because the description mapping ignores the link between $\langle f\ h \rangle$ and $\langle g\ i \rangle$ in the first description. Such links are useful only when unifying with further descriptive elements, not in the completed feature structure, which merely provides feature-value assignments.

Informally, we can think of elements of D as directed rooted graphs and of elements of F as their unfoldings as trees, the unfolding being given by the mapping δ. It is worth noting that if a description is *cyclic*—that is, if it has cycles when viewed as a directed graph—then the resulting feature tree will be infinite.[3]

Stated more precisely, an element f of a domain is *finite*, if for any ascending sequence $\{d_i\}$ such that $f \sqsubseteq \bigsqcup_i d_i$, there is an i such that $f \sqsubseteq d_i$. Then the cyclic elements of D are those finite elements that are mapped by δ into nonfinite elements of F.

3.5. Providing a Denotation for a Grammar

We now move on to the question of how the domain D is used to provide a denotational semantics for a grammar formalism.

We take a simple grammar formalism with rules consisting of a context-free part over a nonterminal vocabulary $\mathcal{N} = \{N_1, \ldots, N_k\}$ and a set of equations over paths in $([0..\infty] \cdot L^*) \cup C$. A sample

[3]More precisely a *rational* tree, that is, a tree with a finite number of distinct subtrees.

rule might be

$$\begin{array}{l} S \rightarrow NP\ VP \\ \quad \langle \mathit{0\ subj} \rangle = \langle 1 \rangle \\ \quad \langle \mathit{0\ predicate} \rangle = \langle 2 \rangle \\ \quad \langle \mathit{1\ agr} \rangle = \langle \mathit{2\ agr} \rangle \end{array} .$$

This is a simplification of the rule format used in the PATR-II formalism [Shieber, *et al.*, 83; Shieber, 84]. The rule can be read as "an S is an NP followed by a VP, where the ***subj***ect of the S is the NP, its ***predicate*** the VP, and the ***agr***eement of the NP the same as the ***agr***eement of the VP".

More formally, a grammar is a quintuple $G = \langle \mathcal{N}, S, L, C, R \rangle$, where

- $\mathcal{N}$ is a finite, nonempty set of nonterminals $N_1, \ldots, N_k$
- S is the set of strings over some alphabet (a flat domain with an ancillary continuous function concatenation, notated with the symbol $\cdot$).
- R is a set of pairs $r = \langle N_{r_0} \rightarrow N_{r_1} \ldots N_{r_m}, E_r \rangle$, where E_r is a set of equations between elements of $([0..m] \cdot L^*) \bigcup C$.

As with context-free grammars, local ambiguity of a grammar means that in general there are several ways of assembling the same subphrases into phrases. Thus, the semantics of context-free grammars is given in terms of sets of strings. The situation is somewhat more complicated in our sample formalism. The objects specified by the grammar are pairs of a string and a partial description. Because of partiality, the appropriate construction cannot be given in terms of sets of string-description pairs, but rather in terms of the related domain construction of ***powerdomains*** [Plotkin, 76; Smyth, 78; Scott, 82]. We will use the ***Hoare powerdomain*** $P = P_M(S \times D)$ of the domain $S \times D$ of string-description pairs. Each element of P is an approximation of a ***transduction relation***, which is an association between strings and their possible descriptions.

We can get a feeling for what the domain P is doing by examining our notion of ***lexicon***. A lexicon will be an element of the domain P^k, associating with each of the k nonterminals N_i, $1 \leq i \leq k$

a transduction relation from the corresponding coordinate of P^k. Thus, for each nonterminal, the lexicon tells us what phrases are under that nonterminal and what possible descriptions each such phrase has. Here is a sample lexicon:

$$NP : \left\{ \begin{array}{l} \langle\text{“Uther”}, \\ \qquad \{\langle agr\ num\rangle = sg, \langle agr\ per\rangle = 3\}\rangle \\ \langle\text{“many knights”}, \\ \qquad \{\langle agr\ num\rangle = pl, \langle agr\ per\rangle = 3\}\rangle \end{array} \right\}$$

$$VP : \left\{ \begin{array}{l} \langle\text{“storms Cornwall”}, \\ \qquad \{\langle agr\ num\rangle = sg\}\rangle \\ \langle\text{“sit at the Round Table”}, \\ \qquad \{\langle agr\ num\rangle = pl\}\rangle \end{array} \right\}$$

$$S : \quad \{\}$$

By decomposing the effect of a rule into appropriate steps, we can associate with each rule r a denotation

$$[r] : P^k \longrightarrow P^k$$

that combines string-description pairs by concatenation and unification to build new string-description pairs for the nonterminal on the left-hand side of the rule, leaving all other nonterminals untouched. By taking the union of the denotations of the rules in a grammar, (which is a well-defined and continuous powerdomain operation,) we get a mapping

$$T_G(\ell) \stackrel{\mathrm{def}}{=} \bigcup_{r \in R} [\![r]\!](\ell)$$

from P^k to P^k that represents a one-step application of all the rules of G “in parallel.”

We can now provide a denotation for the entire grammar as a mapping that completes a lexicon with all the derived phrases and their descriptions. The denotation of a grammar is the function that maps each lexicon ℓ into the smallest fixed point of T_G containing ℓ. The fixed point is defined by

$$[\![G]\!](\ell) = \bigsqcup_{i=0}^{\infty} T_G^i(\ell) \quad ,$$

as T_G is continuous.

It remains to describe the decomposition of a rule's effect into elementary steps. The main technicality to keep in mind is that rules state constraints among several descriptions (associated with the parent and each child), whereas a set of equations in $\mathcal{D}$ constrains but a single description. This mismatch is solved by embedding the tuple $\langle d_0, \ldots, d_m\rangle$ of descriptions in a single larger description, as expressed by

$$\langle i\rangle = d_i, \qquad 0 \leq i \leq m$$

and only then applying the rule constraints—now viewed as constraining parts of a single description. This is done by the *indexing* and *combination* steps described below. The rest of the work of applying a rule, extracting the result, is done by the *projection* and *deindexing* steps.

The four steps for applying a rule

$$r = \langle N_{r_0} \rightarrow N_{r_1} \ldots N_{r_m}, E_r\rangle$$

to string-description pairs $\langle s_1, d_1\rangle, \ldots, \langle s_k, d_k\rangle$ are as follows. First, we index each d_{r_j} into $d_{r_j}^j$ by replacing every path p in any of its equations with the path $j \cdot p$. We then combine these indexed descriptions with the rule by unifying the deductive closure of E_r with all the indexed descriptions:

$$d = \overline{E_r} \sqcup \bigsqcup_{j=1}^{m} d_{i_j}^j \quad .$$

We can now project d by removing from it all equations with paths that do not start with 0. It is clearly evident that the result d^0 is still deductively closed. Finally, d^0 is deindexed into d_{i_0} by removing 0 from the front of all paths $0 \cdot p$ in its equations. The pair associated with N_{r_0} is then $\langle s_{r_1} \cdots s_{r_m}, d_{r_0} \rangle$.

It is not difficult to show that the above operations can be lifted into operations over elements of P^k that leave untouched the coordinates not mentioned in the rule and that the lifted operations are continuous mappings. With a slight abuse of notation, we can summarize the foregoing discussion with the equation

$$[\![r]\!] = \mathit{deindex} \circ \mathit{project} \circ \mathit{combine}_r \circ \mathit{index}_r$$

In the case of the sample lexicon and one rule grammar presented earlier, $[\![G]\!](\ell)$ would be

$$\begin{array}{ll} NP: & \{\cdots \text{as before} \cdots\} \\ VP: & \{\cdots \text{as before} \cdots\} \\ S: & \left\{ \begin{array}{l} \langle \text{“Uther storms Cornwall”}, \\ \qquad \{\langle \mathit{subj\ agr\ num} \rangle = \mathit{sg}, \ldots\}\rangle \\ \langle \text{“many knights sit at the Round Table”}, \\ \qquad \{\langle \mathit{subj\ agr\ num} \rangle = \mathit{pl}, \ldots\}\rangle \\ \langle \text{“many knights storms Cornwall”}, \top \rangle \\ \ldots \end{array} \right\} \end{array}$$

3.6. Applications

We have used the techniques discussed here to analyze the feature systems of GPSG [Sag *et.al.*, 84], LFG [Kaplan and Bresnan, 83] and PATR-II [Shieber, 84]. All of them turn out to be specializations of our domain D of descriptions. Figure 1 provides a

	DCG-II[a]	PATR-II	LFG	GPSG[b]
FEATURE SYSTEM	full	finite	finite	nonrec.
CF SKELETON	full	full	off-line	full

[a]DCGs based on Prolog-II which allows cyclic terms.

[b]HPSG, the current Hewlett-Packard implementation derived from GPSG, would come more accurately under the PATR-II classification.

Figure 3.1: Summary of Grammar System Properties

summary of two of the most critical formal properties of context-free-based grammar formalisms, the domains of their feature systems (full F, finite elements of F, or elements of F based on nonrecursive domain equations) and whether the context-free skeletons of grammars are constrained to be *off-line parseable* [Pereira, 83] thereby guaranteeing decidability.

Though notational differences and some grammatical devices are glossed over here, the comparison is useful as a *first step* in unifying the various formalisms under one semantic umbrella. Furthermore, this analysis elicits the need to distinguish carefully between the domain of feature structures F and that of descriptions. This distinction is not clear in the published accounts of GPSG and LFG, which imprecision is responsible for a number of uncertainties in the interpretation of operators and conventions in those formalisms.

In addition to formal insights, linguistic insights have also been gleaned from this work. First of all, we note that while the systems make crucial use of unification, generalization is also a well-defined notion therein and might indeed be quite useful. In fact, it was this availability of the generalization operation that suggested a simplified account of coordination facts in English now being used in GPSG [Sag *et.al.*, 84] and in an extension of PATR-II [Karttunen, 84]. Though the issues of coordination and agreement are discussed in greater detail in these two works, we present here a simplified view of the use of generalization in a GPSG coordination analysis.

Circa 1982 GPSG [Gazdar, *et al.*, 82] analyzed coordination by using a special principle, the conjunct realization principle (CRP), to achieve partial instantiation of head features (including agreement) on the parent category. This principle, together with the head feature convention (HFC) and control agreement principle (CAP), guaranteed agreement between the head noun of a subject and the head verb of a predicate in English sentences. The HFC, in particular, can be stated in our notation as $\langle \mathit{0\ head} \rangle = \langle \mathit{n\ head} \rangle$ for n the head of 0.

A more recent analysis [Farkas, *et al.*, 83; Sag, *et al.*, 84] replaced the conjunct realization principle with a modified head feature convention that required a head to be more instantiated than the parent, that is: $\langle \mathit{0\ head} \rangle \sqsubseteq \langle \mathit{n\ head} \rangle$ for all constituents n which are heads of 0. Making coordinates heads of their parent achieved the effect of the CRP. Unfortunately, since the HFC no longer forced identity of agreement, a new principle—the nominal completeness principle (NCP), which required that NP's be fully instantiated—was required to guarantee that the appropriate agreements were maintained.

Making use of the order structure of the domains we have just built, we can achieve straightforwardly the effect of the CRP and the old HFC without any notion of the NCP. Our final version of the HFC merely requires that the parent's head features be the *generalization* of the head features of the head children. Formally, we have:

$$\langle \mathit{0\ head} \rangle = \bigsqcap_{i \in \text{heads of } 0} \langle \mathit{i\ head} \rangle \quad .$$

In the case of parents with one head child, this final HFC reduces to the old HFC requiring identity; it reduces to the newer one, however, in cases (like coordinate structures) where there are several head constituents.

Furthermore, by utilizing an order structure on the domain of constants C, it may be possible to model that troublesome coordination phenomenon, number agreement in coordinated noun phrases [Karttunen, 84; Sag, *et al.*, 84].

3.7. Conclusion

We have approached the problem of analyzing the meaning of grammar formalisms by applying the techniques of denotational semantics taken from work on the semantics of computer languages. This has enabled us to

- account rigorously for intrinsically partial descriptions,
- derive directly notions of unification, instantiation and generalization,
- relate feature systems in linguistics with type systems in computer science,
- show that feature systems in GPSG, LFG and PATR-II are special cases of a single construction,
- give semantics to a variety of mechanisms in grammar formalisms, and
- introduce operations for modeling linguistic phenomena that have not previously been considered.

We plan to develop the approach further to give accounts of negative and disjunctive constraints [Karttunen, 84], besides the simple equational constraints discussed here.

On the basis of these insights alone, it should be clear that the view of grammar formalisms as programming languages offers considerable potential for investigation. But, even more importantly, the ***linguistic discipline*** enforced by a rigorous approach to the design and analysis of grammar formalisms may make possible a hitherto unachievable standard of research in this area.

References

Ait-Kaci, H., 1983: "A new Model of Computation Based on a Calculus of Type Subsumption," Doctoral Dissertation Proposal, Dept. of Computer and Information Science, University of Pennsylvania (November).

Bresnan, J., 1983: *The mental representation of grammatical relations* (ed.), Cambridge: MIT Press.

Colmerauer, A., 1978: "Metamorphosis Grammars." In L. Bolc, Ed., *Natural Language Communication with Computers*, Springer-Verlag, Berlin. First appeared as "Les Grammaires de M'etamorphose," Groupe d'Int'elligence Artificielle, Universit'e de Marseille II (November 1975).

Eisenberg, P., 1973: "A Note on Identity of Constituents," *Linguistic Inquiry 4:3,* 417-20.

Farkas, D., D.P. Flickinger, G. Gazdar, W.A. Ladusaw, A. Ojeda, J. Pinkham, G.K. Pullum, and P. Sells, 1983: "Some Revisions to the Theory of Features and Feature Instantiation." Unpublished manuscript (August).

Gazdar, G., E. Klein, G.K. Pullum, and I.A. Sag, 1982: "Coordinate Structure and Unbounded Dependencies." In M. Barlow, D. P. Flickinger and I. A. Sag, eds., *Developments in Generalized Phrase Structure Grammar.* Indiana University Linguistics Club, Bloomington, Indiana.

Gazdar, G. and G.K. Pullum, 1982: "Generalized Phrase Structure Grammar: A Theoretical Synopsis," Indiana University Linguistics Club, Bloomington, Indiana.

Grosz, B., N. Haas, G. Hendrix, J. Hobbs, P. Martin, R. Moore, J. Robinson and S. Rosenschein, 1982: "DIALOGIC: a core natural-language processing system," *Proceedings of the Ninth International Conference on Computational Linguistics*, Prague, Czechoslavakia (July), pp. 95-100.

Harrison, M., 1978: *Introduction to Formal Language Theory.* Addison-Wesley, Reading, Massachusetts.

Kaplan, R. and J. Bresnan, 1983: "Lexical-Functional Grammar: A Formal System for Grammatical Representation," in J. Bresnan (ed.), *The mental representation of grammatical relations* (ed.), Cambridge: MIT Press.

Karttunen, L., 1984: "Features and Values," *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford California (4-7 July).

Karttunen, L., 1983: "KIMMO: a general morphological processor," *Texas Linguistic Forum,* Volume 22 (December), pp. 161-185.

Kay, M., 1979: "Functional Grammar," in *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society,* Berkeley Linguistics Society,B erkeley, California (17-19 February).

Kay, M., 1979: "Functional Grammar." *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistic Society,* Berkeley Linguistic Society, Berkeley, California (February 17-19, 1979), pp. 142-158.

Kay, M., 1983: "Unification Grammar," unpublished memo, Xerox Palo Alto Research Center.

Koskenniemi, K., 1983: "A Two level Model for Morphological Analysis and Synthesis," forthcoming Ph.D. dissertation, University of Helsinki, Helsinki, Finland.

Marcus, M., D. Hindle and M. Fleck. "D-Theory: Talking about Talking about Trees." *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics,* Boston, Massachusetts (15-17 June).

Pereira, F. and D.H.D. Warren, 1983: "Parsing as Deduction," in *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics* (15-17 June), pp. 137–144.

Pereira, F., 1981: "Extraposition Grammars." *American Journal of Computational Linguistics 7*, 4 (October-December), 243-256.

Pereira, F. and S. Shieber, 1984: "The Semantics of Grammar Formalisms Seen as Computer Languages," *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford California (4-7 July).

Pereira, F. and D. H. D. Warren, 1980: "Definite Clause Grammars for Language Analysis—a Survey of the Formalism and a Comparison with Augmented Transition Networks." *Artificial Intelligence 13* (1980), 231-278.

Pereira, F. C. N., and D. H. D. Warren, 1983: "Parsing as Deduction." *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, Boston, Massachusetts, (15-17 June), pp. 137-144.

Plotkin, G., 1976: "A Powerdomain Construction." *SIAM Journal of Computing 5*, 452-487.

Reynolds, J., 1970: "Transformational Systems and the Algebraic Structure of Atomic Formulas," in D. Michie (ed.), *Machine Intelligence*, Vol. 5, Chapter 7, Edinburgh, Scotland: Edinburgh University Press, pp. 135–151.

Sag, I.A., G. Gazdar, T. Wasow, and S. Weisler, 1984: "Coordination and How to Distinguish Categories." CLSI Report No. 3. Center for the Study of Language and Information, Stanford, Ca., (March).

Scott, D., 1982: "Domains for Denotational Semantics," ICALP '82, Aarhus, Denmark (July).

Shieber, S.M., 1984: "The Design of a Computer Language for Linguistic Information." *Proceedings of the Tenth International Conference on Computational Linguistics* (4-7 July).

Shieber, S.M., H. Uszkoreit, F. Pereira, J. Robinson, and M. Tyson, 1983: "The Formalism and Implementation of PATR-II," in B. Grosz and M. Stickel, *Research on Interactive Acquisition and Use of Knowledge*, SRI Final Report 1894, SRI International, Menlo Park, California (November).

Smyth, M., 1978: "Power Domains." *Journal of Computer and System Sciences 16*, 23-36.

Stoy, J., 1977: *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* MIT Press, Cambridge, Massachusetts.

van Emden, M. and R. A. Kowalski, 1976: "The Semantics of Predicate Logic as a Programming Language." *Journal of the ACM 23*, 4 (October 1976), 733-742.

Winograd, T., 1972: *Understanding Natural Language,* New York, New York: Academic Press.

Woods, W., 1970: "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM,* Vol. 13, No. 10 (October).

Woods, W., *et al.*, 1976: "Speech Understanding Systems: Final Report." BBN Report 3438, Bolt Beranek and Newman, Cambridge, Massachusetts.