The semantics of grammar formalisms seen as computer languages

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th>Citation</th>
<th>Fernando C. N. Pereira and Stuart M. Shieber. The semantics of grammar formalisms seen as computer languages. In Proceedings of the Tenth International Conference on Computational Linguistics, pages 123-129, Stanford University, Stanford, California, July 2-6 1984.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Published Version</td>
<td><a href="http://dx.doi.org/10.3115/980491.980518">http://dx.doi.org/10.3115/980491.980518</a></td>
</tr>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:2309658">http://nrs.harvard.edu/urn-3:HUL.InstRepos:2309658</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Abstract

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 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.

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 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 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—this case derived from the declarative semantics of logic programs [3,12,11].

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 [22].

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 [20] 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.

1This 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.

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

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)
[5,15], lexical functional grammar (LFG) [2] and PATR-II
[18,17]; we find that the mathematical structures developed
for this purpose contain an operation of feature generaliza-
tion, not available in those grammar formalisms, that can
be used to give a partial account of the effect of coordina-
tion on syntactic features.

Just as studies in the semantics of programming lan-
guages 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
tools of formal language theory. First, a precise defini-
tion on syntactic features.

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 defini-
tion 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 features.

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
[9,10]. An abstract analysis of those formalisms requires a
notion of partial information that is precisely captured by
the constructs of denotational semantics.

2. Denotational Semantics

In broad terms, denotational semantics is the study of the
connection between programs and mathematical enti-
ties 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 oper-
ations that correspond directly to the ways in which the
parts participate in the whole.

For the purposes of the present work, denotational se-
manics will mean the semantic domain theory initiated by
Scott and Strachey [20]. In accordance with this ap-
proach, 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 gen-
eral 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 consider-
able complexity. Potential nontermination and the use of
functions as "first-class citizens" in computer languages ac-
count 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 seman-
tics of logic grammars in general can be formulated using
elementary set theory [7,21].

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

• Inherent partiality: many grammar formalisms oper-
ate in terms of constraints between elements that do
not fully specify all the possible features of an ele-
ment.

• Technical economy: results that require laborious
constructions without utilizing domain theory can be
reached trivially by using standard results of the the-
ory.

• Suggestiveness: domain theory brings with it a rich
mathematical structure that suggests useful oper-
ations 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. The Domain of Feature Struc-
tures

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, al-
though 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 an-
other, or, as they are sometimes described, partial func-
tions. 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, par-
tial information is critical to the use of such systems—for
instance, in the variables of definite clause grammars [12]
and in the GPSG analysis of coordination [15]. That is, the
elements of the feature systems, called feature structures
(alternatively, feature bundles, f-structures [2], or terms)
can be partial in some sense. The partial descriptions, be-
ing 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
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

\[ F = [L \rightarrow F] + C \]

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.

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:

\[
(\cdots(x(l_1))\cdots)(l_m) = (\cdots(x(l_1))\cdots)(l_n)
\]

which we prefer to write as

\[
\langle l_1, \ldots, l_m \rangle = \langle l_1, \ldots, l_n \rangle
\]

with \( x \) implicit. The terms of such equations are constants \( c \in C \) or paths \( (l_1, \ldots, l_m) \), 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 [16], 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 \( D \) describing the elements of \( F \) is defined, following Scott, as the tuple

\[ D = (\mathcal{D}, \Delta, \text{Con}, \vdash) \]

where \( D \) is a set of propositions, \( \text{Con} \) is a set of finite subsets of \( D \), the consistent subsets, \( \vdash \) is an entailment relation between elements of \( \text{Con} \) and elements of \( D \) and \( \Delta \) is a special least informative element that gives no information at all. We say that a subset \( S \) of \( D \) is deductively closed if every proposition entailed by a consistent subset of \( S \) is in \( S \). The deductive closure \( \text{C} \) of \( S \subseteq D \) is the smallest deductively closed subset of \( D \) 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 \( \Delta \) or \( e = d \) 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 \( d_1 = d_2 \) and both \( p_1 = p_2 \) and \( d_1 = p_1 \cdot d_2 \) are in \( E \)
- Iteration: there is \( E' \subseteq E \) such that \( E' \vdash e \) and for all

\[ e' \in E' \vdash e' \]

With this notion of entailment, the most natural definition of the set \( \text{Con} \) is that a finite subset \( E \) of \( D \) 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 \( D \) is consistent if and only if all its finite subsets are consistent in the way defined above. The consistent and deductively closed subsets of \( D \) 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 \( \subseteq \) in \( D \) provides the notion of a description being more or less specific than another. The least-upper-bound operation \( \cup \) combines two descriptions into the least instantiated description that satisfies the equations in both descriptions, their unification. The greatest-lower-bound operation \( \cap \) 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:
That is, falsity entails anything. Next we define Con to be simply the set of all finite subsets of 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 ⊥, the deductive closure of a set containing an inconsistent equation is the whole of D. The partial order D is now a lattice with top element T = D, and the unification operator ♯ is always defined and returns T on unification failure.

We can now define the description mapping δ : D → F that relates descriptions to the described feature structures. The idea is that, in proceeding from a description d ∈ D to a feature structure f ∈ 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 [d] of d with elements of the form

\[(l_1 \cdots l_m) = c_k \cdots \]

Each e ∈ [d] defines an element f(e) of F by the equations:

\[f(e)(l_1) = f_1 \]
\[\vdots \]
\[f_{m-1}(l_m) = f_m \]
\[f_m(l_m) = c_k \]

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

\[δ(d) = \bigsqcup_{e \in [d]} f(e) \]

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 onto one element of F. For example, the elements given by the two sets of equations:

\[\{ (f \ h) = (c \ i) \} \quad \{ (f \ h) = (g \ i) = c \} \]

describe the same structure, because the description mapping ignores the link between (f h) and (g i) 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 ∈ \bigsqcup d_i, there is an i such that f ⊈ d_i. Then the cyclic elements of D are those finite elements that are mapped by δ into nonfinite elements of F.

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 \(N = \{N_1, \ldots, N_k\}\) and a set of equations over paths in \([0..\infty] \cdot L^*\). A sample rule might be

\[S \rightarrow NP \ VP\]
\[\{0 \ subj\} = (1)\]
\[\{0 \ predicate\} = (2)\]
\[\{1 \ agr\} = (2 \ agr)\]

This is a simplification of the rule format used in the PATR-II formalism [18,17]. The rule can be read as "an S is an NP followed by a VP, where the subject of the S is the NP, its predicate the VP, and the agreement of the NP the same as the agreement of the VP."

More formally, a grammar is a quintuple \(G = (N, S, L, C, R)\), where

• \(N\) 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, noted with the symbol \(\cdot\)).
• \(R\) is a set of pairs \(r = (N_{m_0} \rightarrow N_{m_1} \ldots N_{m_n}, E_r)\), where \(E_r\) is a set of equations between elements of \(([0..m] \cdot L^*) \cup C\).

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 [14,19,16]. We will use the Hoare powerdomain \(P = PM(S \times D)\) 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\(^3\) more precisely a rational tree, that is, a tree with a finite number of distinct subtrees.

\(^3\)More precisely a rational tree, that is, a tree with a finite number of distinct subtrees.
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:

\[
\begin{align*}
NP: & \{ \text{"Uther"}, \{(agr \text{ num}) = sg, (agr \text{ per}) = 3\} \} \\
& \{ \text{"many knights"}, \{(agr \text{ num}) = pl, (agr \text{ per}) = 3\} \} \\
VP: & \{ \text{"storms Cornwall"}, \{(agr \text{ num}) = sg\} \} \\
& \{ \text{"sit at the Round Table"}, \{(agr \text{ num}) = pl\} \} \\
S: & \{ \}
\end{align*}
\]

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

\[ [r]: P^k \rightarrow P^k \]

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

\[ T_G(t) \overset{\text{def}}{=} \bigcup_{r \in R} [r](t) \]

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 \( t \) into the smallest fixed point of \( T_G \) containing \( t \). The fixed point is defined by

\[ [G](t) = \bigcup_{i=0}^{\infty} T_G^i(t) \]

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 \( D \) constrains but a single description. This mismatch is solved by embedding the tuple \( (d_0, \ldots, d_m) \) of descriptions in a single larger description, as expressed by

\[ (i) = d_i, \quad 0 \leq i \leq m \]

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

The four steps for applying a rule

\[ r = (N_{r_0} \rightarrow N_{r_1} \ldots N_{r_m}, E_r) \]

to string-description pairs \( (s_1, d_1), \ldots, (s_k, d_k) \) are as follows. First, we index each \( d_i \) into \( d_i' \) 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 \( E \), with all the indexed descriptions:

\[ d = E_r \cup \bigcup_{i=1}^{m} d_i' \]

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 \( d' \) is still deductively closed. Finally, \( d' \) is deindexed into \( d_0 \) by removing 0 from the front of all paths \( 0 \cdot p \) in its equations. The pair associated with \( N_{r_0} \) is then \( (s_1, \ldots, s_m, d_0) \).

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

\[ [r] = \text{deindex} \circ \text{project} \circ \text{combine} \circ \text{index}, \]

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

\[
\begin{align*}
NP: & \{ \text{"Uther"}, \{(agr \text{ num}) = sg, (agr \text{ per}) = 3\} \} \\
& \{ \text{"many knights"}, \{(agr \text{ num}) = pl, (agr \text{ per}) = 3\} \} \\
VP: & \{ \text{"storms Cornwall"}, \{(agr \text{ num}) = sg\} \} \\
& \{ \text{"sit at the Round Table"}, \{(agr \text{ num}) = pl\} \} \\
S: & \{ \}
\end{align*}
\]

6. Applications

We have used the techniques discussed here to analyze the feature systems of GPSG [15], LFG [2] and PATR-II [17]. All of them turn out to be specializations of our domain \( D \) of descriptions. Figure 1 provides a 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 [13] thereby guaranteeing decidability.
that while the systems make crucial use of unification, gen-
semantic umbrella. Furthermore, this analysis elicits the
devices are glossed over here, the comparison is useful as
of coordination and agreement are discussed in greater de-
of coordination facts in English now being used in GPSG
eralization is also a well-defined notion therein and might
those formalisms.

imprecision is responsible for a number of uncertain-
need to distinguish carefully between the domain of fea-
tures 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, gen-
eralization 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
[15] and in an extension of PATR-II [8]. Though the issues of
coordination and agreement are discussed in greater de-
tail in these two works, we present here a simplified view of
the use of generalization in a GPSG coordination analysis.

Circa 1982 GPSG [6] 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 agree-
ment 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 \((0 \text{ head}) = (n \text{ head})\) for \(n\) the head of 0.

A more recent analysis [4,15] replaced the conjunct rea-
ralization principle with a modified head feature convention
that required a head to be more instantiated than the
parent, that is: \((0 \text{ head}) \subseteq (n \text{ head})\) 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 re-
quired 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:

\[(0 \text{ head}) = \bigcap (i \text{ head}) \]

where \(i\) is heads of 0.

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 do-
main of constants \(C\), it may be possible to model that trou-
blesome coordination phenomenon, number agreement in
coordinated noun phrases [8,15].

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 phenom-
  ena that have not previously been considered.

We plan to develop the approach further to give ac-
counts of negative and disjunctive constraints [8], 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 lan-
guages offers considerable potential for investigation. But,
even more importantly, the linguistic discipline
enforced by a rigorous approach to the design and analysis of gram-
mar formalisms may make possible a hitherto unachievable
standard of research in this area.

References

[1] Ait-Kaci, H. "A New Model of Computation Based
on a Calculus of Type Subsumption." Dept. of Com-
puter and Information Science, University of Penn-
sylvania (November 1983).

Grammar: A Formal System for Grammatical Repre-
sentation." In J. Bresnan, Ed., The Mental Repre-
sentation of Grammatical Relations, MIT Press, Cam-


