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.

Citation

Published Version
http://dx.doi.org/10.3115/980491.980518

Citable link
http://nrs.harvard.edu/urn-3:HUL.InstRepos:2309658

Terms of Use
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 http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA
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—in 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.

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

2This 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.
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 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 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 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 [20]. 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 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 [7,21].

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 constructions 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. 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 [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, 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

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

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 $z \in F$ we have equations of the following forms:

$$(\cdots (x(l_i)) \cdots)(l_m) = (\cdots (x(l_i)) \cdots)(l_n)$$

which we prefer to write as

$$(l_i, \ldots, l_m) = (l_i, \ldots, l_n)$$

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 $\mathcal{D}$ 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 $\Delta$ 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 $\Delta$ 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 $d_1 = d_2$ and both $p_1 = p_2$ and
  $d_1 = p_1 \cdot d_2$ and both $p_1 = p_2$ and
  $d_1 = p_1 \cdot d_2$

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 $e_1 = e_2$, with $e_1$ and $e_2$ 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 $\subseteq$ 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
consider only the subset \( d_{LdJ} \) of \( d \) with elements of the form

more precisely, seeing \( d \) as a set of equations, we
can define \( \delta(d) \) as

value constraints, but does not specify the values them-
selves. More precisely, seeing \( d \) as a set of equations, we
keep only definite information that only states
a feature structure \( f \not\in F \), we keep only definite informa-
tion about values and discard information that only states
properties that increasing information in a description leads
to nondecreasing information in the described structures
(monotonicity) and that if a sequence of descriptions ap-
proximates another description, the same condition holds for
the described structures.

Note that \( \delta \) may map several elements of \( D \) on to one

element of \( F \). For example, the elements given by the two
sets of equations

describe the same structure, because the description map-
ing 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 \( \delta \). 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 fi-
nite, if for any ascending sequence \( \{d_i\} \) such that \( f \subseteq \bigcup d_i \),
there is an \( i \) such that \( f \subseteq d_i \). Then the cyclic elements
of \( D \) are those finite elements that are mapped by \( \delta \) 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 con-
sisting of a context-free part over a nonterminal vocabu-
lar \( N = \{N_1,\ldots,N_k\} \) and a set of equations over paths in
\( ([0..c_\infty]:L') \bigcup C \). A sample rule might be

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 do-
main with an ancillary continuous function concatene-
tion, notated with the symbol \( \cdot \)).
- \( R \) is a set of pairs \( r = (N_t \rightarrow N_t_1 \ldots N_t_m, E_r) \),
where \( E_r \) is a set of equations between elements of
\( ([0..m]:L') \bigcup C \).

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 complic-
ed 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 pow-
erdomains \[14,19,16\]. We will use the Hoare powerdomain
\( P = P_H(S \times D) \) of the domain \( S \times D \) of string-description
pairs. Each element of \( P \) is an approximation of a transduc-
tion 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.
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 non-
terminal and what possible descriptions each such phrase has. Here is a sample lexicon:

$$NP:\{\{\text{"Uther"}, \{(\text{agr num}) = \text{sg}, (\text{agr per}) = 3\}\}\}
\{\{\text{"many knights"}, \{(\text{agr num}) = \text{pl}, (\text{agr per}) = 3\}\}\}\}
$$

$$VP:\{\{\text{"storms Cornwall"}, \{(\text{agr num}) = \text{sg}\}\}\}
\{\{\text{"sit at the Round Table"}, \{(\text{agr num}) = \text{pl}\}\}\}\}
$$

$$S:\{\}$$

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

$$T_G(\ell) := \bigcup_{r \in R} [r](\ell)$$

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 gram-
mar 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 $\ell$ into the
smallest fixed point of $T_G$ containing $\ell$. The fixed point is defined by

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

as $T_G$ is continuous.

It remains to describe the decomposition of a rule's ef-
effect into elementary steps. The main technicality is 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 descrip-
tion. 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
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

$$r = (N_{r_0} \rightarrow N_{r_1} \ldots N_{r_m}, E)$$

to string-description pairs $(s_1, d_1), \ldots, (s_k, d_k)$ are as follows. First, we index each $d_i$ into $d_i^I$ 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:

$$d = E \cup \bigcup_{i=1}^m d_i^I$$

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

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

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

In the case of the sample lexicon and one rule grammar

presented earlier, $[G](\ell)$ would be

$$NP:\{\ldots \text{as before} \ldots\}\}
\{\ldots \text{as before} \ldots\}\}

$$

$$VP:\{\ldots \text{as before} \ldots\}\}
\{\ldots \text{as before} \ldots\}\}

$$

$$S:\{\}$$

$$\ldots$$

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 do-
main $D$ 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
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 [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 phenomena that have not previously been considered.

We plan to develop the approach further to give accounts 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 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


