GRAMMARS AND LOGICS OF PARTIAL INFORMATION

Technical Note 420

April 29, 1987

By: Fernando C. N. Pereira

Artificial Intelligence Center
Computer and Information Sciences Division

Center for the Study of Language and Information
Stanford University

This research has been made possible by a gift from the Systems Development Foundation. This report is a slightly revised version of a paper to be presented at *The International Conference on Logic Programming*, Melbourne, Australia, May, 1987.

Grammars and Logics of Partial Information

Fernando C. N. Pereira

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

Abstract

This paper is an informal survey of models of grammatical categories in unification-based formalisms from computational linguistics and their relationship to current logic programming concepts. The basic notion of partiality in the informational content of grammatical categories is introduced, and specific automata-theoretic, denotational and logical models of partial information are outlined. The expression of disjunction, negation, inheritance and indeterminate constraints in categories is also discussed.

1 Introduction

In parallel with, and for all intents independently of, the development of logic programming and logic grammars, computational linguists, starting with Martin Kay [19], have developed a notion of structure unification remarkably similar to the notion of unification for first-order terms. This line of work has led to the development of a variety of models for linguistic objects and their combination that has remarkable similarities with ideas from logic programming, but also some significant differences. For conciseness, I refer to those models of linguistic objects in the rest of this paper as *partial-information models*, even though related, and sometimes more general, notions of partial information occur in logic, denotational semantics, and knowledge representation.

Kay's functional-unification grammar (FUG) has as its origin an attempt to clarify and give a computational realization to the linguist's notion of *feature*. Broadly speaking, whereas the atomic phrase labels of context-free grammars may be sufficient to characterize the relatively simple grammatical categories[1] of computer languages (but see Algol-68!), natural-language phrases need much richer classification systems because different grammatical rules are sensitive to different phrase features. For example, subject-verb agreement in English appeals to a *number* feature, with possible values *singular* and *plural*. On the other hand, the requirement that a transitive verb have a direct object may be expressed by a list-valued *subcategorization* feature for the verb, where the feature value specifies which complements the verb requires. In general, categories are given by specifying the values of some features and leaving other features unspecified. Furthermore, the category t of phrases of both category t_1 and category t_2 is defined whenever t_1 and t_2 are *compatible*, in the sense that they assign compatible values to features specified in both categories. Still informally, we say that t is the *unification* of t_1 and t_2. A similar notion of category was given further linguistic motivation in the theories of lexical-functional grammar (LFG) [3] and generalized phrase-structure grammar (GPSG) [8].

A precise mathematical characterization of grammatical categories needs a careful distinction between categories and *descriptions*, which are formal expressions representing categories (or, in other terms, formal characterizations that phrases may or may not satisfy). However, this distinction is sometimes blurred because the most convenient mathematical construction of the domain of categories may be in terms of descriptions. Furthermore, in the less expressive models that I discuss, categories and *partial values* can be identified. Thus we have thus a cluster of strongly related notions of description, category and partial value. For this cluster, I use the term *grammatical category*, or just category.

The notion of category implicit in FUG and developed in more recent models, is discussed in more detail in the rest of the paper. For the reader who is familiar with definite-clause-based logic grammars, the above notion of category may seem almost undistinguishable from the that of first-order terms as partially-specified objects in logic grammars. However, this similarity should not be allowed to conceal some important formal and pragmatic differences. A few distinctions are particularly important in this connection:

- Partiality of expression is taken as fundamental in the notion of category, that is, phrases are classified using partial descriptions for which there is no natural notion of most specific (ground) description. This

[1]In this paper, the term "category" is always used in its linguistic sense, never in the mathematical sense of category theory.

contrasts with the space of first-order logic terms, in which the ground terms are the most specific objects.

- Grammatical categories are modular, in the sense that different clusters of features or a category bear information relevant to different parts of the grammar. No rule should have to mention features that are irrelevant to the rule. This point is further elaborated on Section 2.1.

- The burden of representation falls much more heavily on descriptions than on rules. This means that grammatical constructions that have to be expressed by alternative rules in a definite-clause grammar (DCG) might instead be described by *disjunctive* descriptions, that is, disjunctions of other descriptions. In particular, in FUG there are no grammar rules whatsoever: the grammar is just a large disjunctive description.

- Inherent partiality leads to particular problems in the interpretation of inequalities and negation, as will be discussed in Section 6.2.

The paper is organized as follows. I discuss grammatical categories and some of their abstract properties in Section 2. In Section 3, I introduce the feature-value model of categories, together with its informal realization in terms of directed graphs. In Section 4, I outline an equational model for categories based on Scott's concept of information system; in this model, the system of categories has the required mathematical properties. The need for disjunctive descriptions and the difficulties of expressing them in previous models leads to the Rounds-Kasper logic, which is discussed in Section 5. In Section 6 I consider three proposed extensions to the basic feature-value model: inheritance, negation, and the use of regular expressions over features in equations. Finally, in Section 7, I present my suggestions for further work in the area.

As a snapshot of current research, a survey of this kind is both dated and limited in the range of issues it covers. For reasons of space and time, I have deliberately restricted the main discussion to concepts and formalisms derived from the specific needs of computational linguistics. Thus, I should acknowledge here some ideas and techniques that are not discussed in detail, even though they are clearly relevant to the modeling of grammatical categories. Among these I include Ait-Kaci's calculus of type subsumption [1], Cardelli's system of types with inheritance [4], Mukai's CIL [23], and recent

$$\begin{bmatrix} \texttt{cat} & = & \texttt{N} \\ \texttt{bar} & = & \texttt{2} \\ \texttt{agr} & = & \begin{bmatrix} \texttt{number} & = & \texttt{singular} \\ \texttt{person} & = & \texttt{third} \end{bmatrix} \end{bmatrix}$$

Figure 1: Feature-Value Matrix

unpublished work of Goguen and Meseguer on an initial-algebraic account of the calculus of features. A comprehensive coverage including both those models and the linguistically motivated ones must await the development of a unified framework for partial objects and partial information.

2 Grammatical Categories

2.1 Features and Values

Categories in linguistics are usually specified by giving values for certain *features* of the phrases in the category. For example, the category *singular noun phrase*, of which " 'The Man without Qualities' ", "the book" and "an unread book with more than 200 pages" are instances, might be given by specifying the feature `cat` (for category) as `N` (for noun), the feature `bar` (for bar level) as 2, and the feature `agr` (for grammatical agreement) as having a value with its `number` feature (for grammatical number) specified as `singular`. However, other features, such as `definite` that indicates whether the noun phrase is definite (the first two examples are, the third is not), are left unspecified. A partial *description* (or category specification) like the one above is usually given either in matrix form as in Figure 1 or in graph form as in Figure 2.

This direct expression of linguistic features should be compared with the encodings normally used in DCGs and related logic-grammar formalisms. In these formalisms, complex categories are represented by compound nonterminal symbols, with argument positions playing the role of features. Such an encoding is not only unnatural for the linguist, but it also suffers from lack of modularity. Adding a new feature will require changes to all occurrences of complex nonterminals that may require that feature, because a new argument position is needed for the feature. In contrast, with feature-value notation only those rules involving a particular feature need to refer

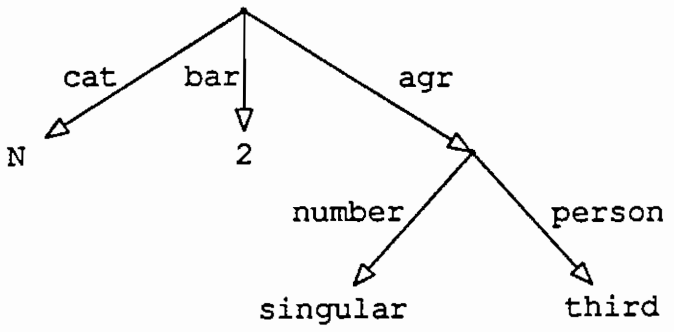

Figure 2: Feature-Value Graph

to it. This reflects the linguistic modularity implicit in the notion of feature: linguistic distinctions are made along several independent dimensions that relate to different parts of a linguistic rule system.

If all the information we have about phrases is in the form used in the above example, we may model categories directly in terms of *feature trees* (as in LFG) constructed from *features* like cat and *atomic values* like N. More formally, given a set of features F and a set of atomic values C, we can define a set V of complex values as the union of the sets V_i where $V_0 = C$ and V_i is the set of all partial functions $v : F - \bigcup_{j<i} V_i$ with finite domain $|v|$. With this construction, a value is either a constant or a partial function with finite domain from features to values. The value of some feature f of a feature tree v is given by applying v (a partial function) to f if $f \in |v|$, and is undefined otherwise.

Certain notational conventions make it easier to discuss grammatical categories. The symbol $\perp$ in equations such as $v(f) = \perp$ is used to mean that the partial function v is undefined for argument f. A sequence of features $\langle f_1 \cdots f_n \rangle$ is called a *path*. The symbol $\prec$ will represent the prefix relation on paths, that is, $p \prec p'$ if p is a strict initial subsequence of p'. The concatenation of two paths p and p' is represented by the expression $p \cdot p'$. For any complex value v and path $p = \langle f_1 \cdots f_n \rangle$, the expression v/p represents the value $(\cdots v(f_1) \cdots)(f_n)$, if the latter is defined.

A feature tree $v \in V$ can be identified with a category, the *principal* *category* of v, which is the set of all the feature trees v' *subsumed* by v. A feature tree v subsumes all feature trees that are at least as informative as itself, that is, all feature trees that specify at least as many features as v with values at least as informative as those given in v. More precisely, $v \sqsubseteq v'$ if both are identical atomic values or if both are partial functions such that $|v| \subseteq |v'|$ and for all $f \in |v|$, $v(f) \sqsubseteq v'(f)$. If v subsumes v', we also say that v' is an *instance* of v.

2.2 Abstract Properties

With the definition given in the previous section, feature trees are partially ordered under the subsumption relation $\sqsubseteq$, which carries over to their principal categories. Feature trees and their principal categories are a particular case of an abstract notion of grammatical category.

A set of objects can be used to classify phrases by means of a relation $s : t$ between phrases s and objects t, read *s is a t*. For the set of classificatory objects to qualify as a set of grammatical categories, it should satisfy at least the following conditions:

- The subsumption relation $t \sqsubseteq t'$ that holds between t and t' if and only if every phrase of category t' is of category t should be a partial order.

- Categories t and t' are called *consistent* if and only if there is t'' with $t \sqsubseteq t''$ and $t' \sqsubseteq t''$. If t and t' are consistent, there should be a least (in the subsumption ordering) category $t \sqcup t'$ with $t \sqsubseteq t \sqcup t'$ and $t' \sqsubseteq t \sqcup t'$, the *least upper bound* (lub) of t and t'. This lub is also called the *unification* of t and t'.

- The subsumption ordering should have a least *trivial* category $\bot$ to which every phrase belongs.

It is easy to see that feature trees and their principal categories satisfy the above conditions. The trivial category corresponds to the partial function with empty domain. The unification of two feature trees (and the corresponding principal categories) can be defined by the condition

$$(v \sqcup v')(f) = \begin{cases} v(f) \sqcup v'(f) & f \in |v| \cap |v'| \\ v(f) & f \in |v| - |v'| \\ v'(f) & f \in |v'| - |v| \end{cases}$$

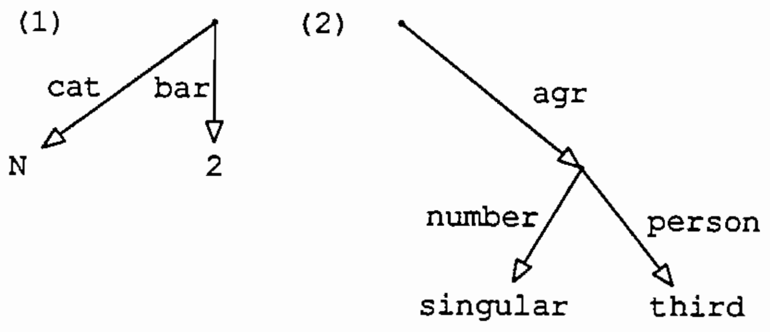

Figure 3: Two Unifiable Trees

for $f \in |v| \cup |v'|$, whenever all the unifications in the right-hand side are possible; two atomic values unify just in case they are identical. For example, the tree in Figure 2 is the unification of trees (1) and (2) of Figure 3.

The description analogue of a phrase s being of a category is the phrase satisfying a description d, $s \models d$. However, the corresponding subsumption relation between descriptions may not be a partial order but just a preorder (a reflexive and transitive relation), because there may be formally distinct descriptions satisfied by exactly the same objects. This situation is familiar in many settings, including the case of the subsumption ordering of first-order terms, in which descriptions are first-order terms and the objects are ground terms, and all renaming variants of a given term are satisfied by exactly the same set of ground instances (which may be identified with the category corresponding to those renaming variants).

3 Feature Graphs

Although feature trees correspond very closely to the linguistic notion of category, principal categories are insufficient for linguistic description. The reason for this is that grammar rules do not normally define principal categories, but rather categories expressed in terms of constraints between parts of feature trees, much in the same way as shared variables in a Prolog clause impose constraints on the extensions of Prolog predicates. Informally, such

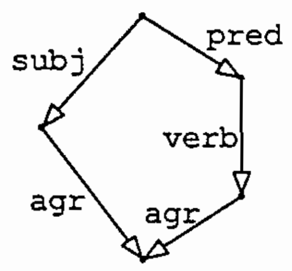

Figure 4: Feature Graph for Subject-Verb Agreement

constraints can be expressed by moving from feature trees to *feature graphs*. For example, the agreement between subject and verb in English sentences might be represented by the graph in Figure 4. The intended interpretation of the graph is that the agreement feature agr of the subject subj of a sentence is the same as the agreement feature of the verb of the predicate pred or the sentence. In category terms, the graph may be thought of as a description of the category of all feature trees with identical subtrees at the ends of the paths ⟨subj agr⟩ and ⟨pred verb agr⟩ from the root. It should be obvious that graph descriptions are analogous to nonground first-order terms with multiple occurrences of variables.

The feature graph above is *acyclic*, as seems to be the case with most feature graphs used in linguistic description. However, certain phenomena seem to be more easily described using *cyclic* graphs. For instance, case-marked noun phrases may be represented by a cyclic graph that indicates that the phrase is the value of a particular grammatical function (say, object) of the embedding clause. As an example, the category of accusative noun phrases might include the subgraph of Figure 5, which states that the given noun phrase is the direct object obj of its embedding clause. Again, cyclic feature graphs have a direct analogue in logic programming, namely the rational terms of Prolog-II [5].

The grammatical categories of Section 2.1 correspond to the special case of tree-shaped feature graphs. Feature graphs that are not trees are called

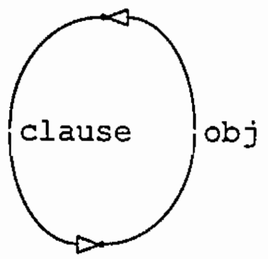

Figure 5: Cyclic Feature Graph

reentrant. Graphs can be modeled by deterministic finite-state acceptors with a single initial state. The initial state is the root of the graph, transition labels are features, and final states may be labeled by atomic values [1,26].. Formally, given a set F of features and a set C of atomic constants, a feature graph is given by a quadruple $\langle S, s_0, \delta, \pi\rangle$ where S is a set of *states* (or *nodes*), $s_0 \in S$ is the *initial state*, $\delta : S \times F \rightharpoonup S$ is the [partial] *next state* function, and $\pi : S \rightharpoonup C$ is the [partial] labeling function, whose domain is contained in the set of *final states*, which are those states s for which there is no feature f such that $\delta(s, f)$ is defined. For any path $p = \langle f_1 \cdots f_n\rangle$ and any feature graph g with initial state s_0, the expression s_0/p represents the state $\delta(\cdots\delta(f_1, s_0)\cdots, f_n)$, if this is defined, and the expression g/p represents the subgraph of g reachable from s_0/p.

A graph $g = \langle S, s_0, \delta, \pi\rangle$ subsumes another $g' = \langle S', s'_0, \delta', \pi'\rangle$ if and only if there is an automaton morphism $m : g \rightarrow g'$, that is, a function from S to S' with $m(s_0) = s'_0$, and for every $f \in F$ and $s \in S$, $m(\delta(s, f)) = \delta'(m(s), f)$ and $\pi'(m(s)) = \pi(s)$. For example, according to this definition graphs (1) and (2) of Figure 6 subsume graph (3), which is in fact their unification.

Graph subsumption as defined here is a preorder, analogous to the subsumption ordering of rational terms. In the same way as with rational terms, it is easy to define equivalence classes of graphs (which are just isomorphism classes for automata) and from them a partially ordered set of graph equivalence classes with a unification operation satisfying the abstract properties

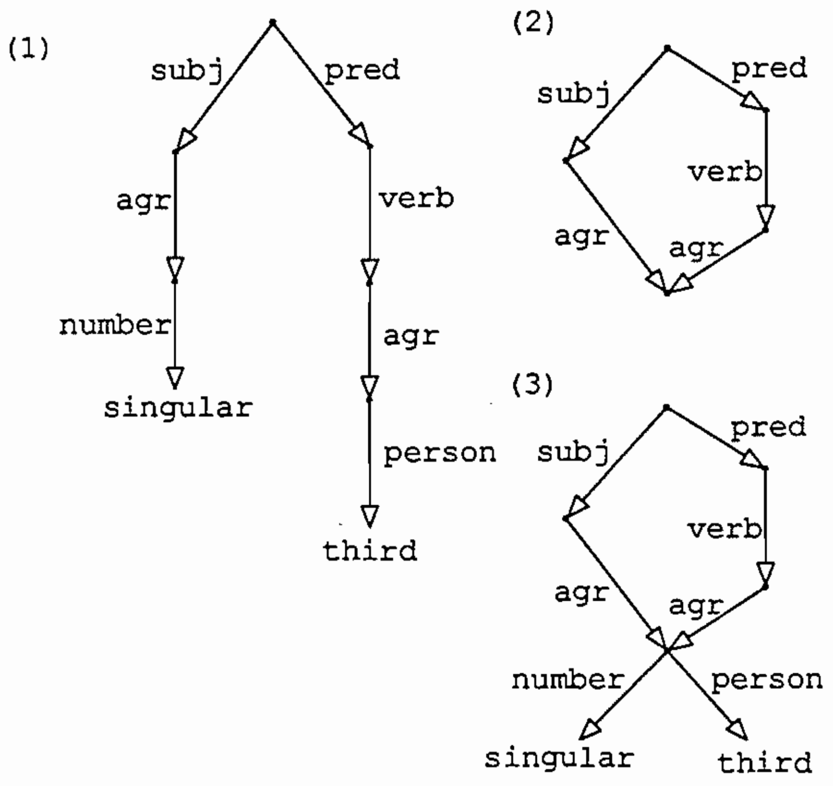

Figure 6: Graph Unification

discussed in the previous section. Furthermore, the UNION-FIND unification algorithm for rational terms [11] has a direct feature graph analogue, which was independently discovered in a somewhat more general setting by Ait-Kaci [1,2].

The above notion of graph subsumption has the consequence that all instances of a reentrant graph are themselves reentrant. Thus, the principal category of a reentrant feature graph contains only reentrant feature graphs. However, it is sometimes convenient, as is done in LFG [3], to think of acyclic feature graphs as defining categories of feature *trees*, rather than categories of feature graphs. More specifically, a feature tree t *satisfies* an acyclic feature graph g with initial state s_0 iff the following three conditions are satisfied:

- For any path p, if s_0/p is defined then t/p is also defined.
- If p is a path with s_0/p a final state with label c, $t/p = c$.
- For any two paths p and p', if $s_0/p = s_0/p'$, then for any two paths q and q' with $p \preceq q$ and $p' \preceq q'$, $t/q = t/q'$.

The idea of this definition is that a tree satisfies an acyclic graph just in case the tree is at least as defined as the graph, and furthermore the identities between features indicated by reentrancies in the graph are satisfied in the tree. The set of trees satisfying the graph is the *tree category* defined by the graph. When moving from graphs to sets of trees, we must be clear about the different notions of identity being used. In a feature graph, an identity is given by a reentrancy, that is, two paths from the root reaching the same node. In a feature tree, subtree identity is just equality of partial functions as sets of ordered pairs. Thus, in feature graphs we have a kind of *intensional* identity (like eq in Lisp) whereas in features trees the identities are *extensional* (like equal in Lisp).

The tree category defined by a graph is analogous to the set of solutions of a finite set of equations over first-order terms. As in this case, in general. given a graph, no finite set of trees will have as its instances exactly the tree category of the graph [21].

The above characterization of feature graphs and their tree categories is rather complicated. Just as in the case of first-order terms, equational characterizations seem to make things substantially simpler. This is the subject of the next section.

4 A Denotational Model

In Section 2.1, grammatical categories were defined in terms of partial functions over features, but only principal categories, corresponding to the categories subsumed by a single tree-shaped category, could be defined. In the preceding section, we saw an automata-theoretic model of feature graphs. Here, I introduce a model of features and values based on deductive operations on sets of formal equations over features. This model was originally introduced in joint work with Stuart Shieber [25] as a basis for a denotational semantics for a variety of unification-based grammar formalisms. I should note from the outset that the relationship between this denotational model and the automata models has not yet been characterized rigorously, so the model is presented here more for its heuristic value than for any specific results tying it to earlier and later models.

The basic idea of the present model is that grammatical categories are defined to be certain sets of *path equations* over the prespecified sets F of features and C of atomic values. Earlier, we used the notation x/p to represent the value of the feature tree or graph x at the end of path p. When the object x is fixed, we can omit it and express conditions on the values of x's features just in terms of paths and atomic values. A path equation is then a formula $u = v$ where u and v may be paths or atomic constants. For example, the graph of Figure 4 can be represented by the path equation

$$\langle \texttt{subj agr} \rangle = \langle \texttt{pred verb agr} \rangle \quad .$$

In what follows, we need a notion of consistency for sets of path equations that captures the notion of a set of equations without any unification clashes. Consistency will be defined in terms of the absence of two kinds of clashes. A *constant* clash is any set consisting of a single *inconsistent* equation of the form $c = c'$ for distinct atomic values c and c'. A *constant/complex* clash is any set of two equations e_1 and e_2 in which e_1 equates some path p to some other path or atomic value and e_2 equates a strict prefix of p to a constant. Constant clashes correspond to attempts to give the same feature different atomic values (analogous to a constant clashes in term unification) and constant/complex clashes correspond to attempts to give both an atomic and a complex value to some feature (analogous to constant/function-symbol clashes in term unification).

Path equations obey the usual laws of reflexivity, symmetry, transitivity and substitutivity of equality. More precisely, the notion of *immediate* *consequence* for path equations is given by the single axiom

$$\langle\rangle = \langle\rangle \quad \text{(Trivial Reflexivity)}$$

and the following rules of inference where u, v and w stand for arbitrary paths or atomic values, p, q for arbitrary paths and r for an arbitrary nonempty path:

$$\frac{u = v}{v = u} \qquad \text{(Symmetry)}$$

$$\frac{u = v \qquad v = w}{u = w} \qquad \text{(Transitivity)}$$

$$\frac{p = q \qquad p \cdot r = v}{q \cdot r = v} \qquad \text{(Substitutivity)}$$

$$\frac{p \cdot q = v}{p = p} \qquad \text{(Reflexivity)} \quad .$$

An equation e is a *consequence* of a finite set E of equations ($E \vdash e$) if it can be derived from elements of E and the reflexivity axiom by repeated applications of these rules. A finite set of equations E is consistent if and only if the set of its consequences does not include constant or constant/complex clashes. An arbitrary set of equations is consistent whenever all of its finite subsets are consistent, and it is *deductively closed* whenever it contains all the consequences of its finite subsets.

The preceding machinery is just what is needed to define the set of grammatical categories in terms of Scott's information systems construction [27]. Scott's model is built from a set of basic *propositions* with notions of consistency and consequence for finite sets of propositions and a trivial proposition. With certain reasonable conditions, which I shall not cover here, the set of deductively closed sets of propositions partially ordered by inclusion satisfies the abstract properties of Section 2.2. [2] In our specific

[2]In fact, it satisfies the much stronger condition of being *ω-algebraic consistently complete cpo* or *domain*. A cpo (complete partial order) is a partial order with a least element in which every infinite increasing sequence has a lub. A cpo is consistently complete iff if every consistent set of elements has a lub, where a set of elements is consistent if it has an upper bound. An element of a cpo is ω-finite iff whenever it is less than the lub of an ascending sequence it is also less than some element in the sequence. A cpo is ω-algebraic if the set of its ω-finite elements is denumberable; furthermore the set of all ω-finite elements less than any element in the cpo is *directed* and has the element as its lub, where a set of elements is directed if it contains an upper bound of each of its finite subsets.

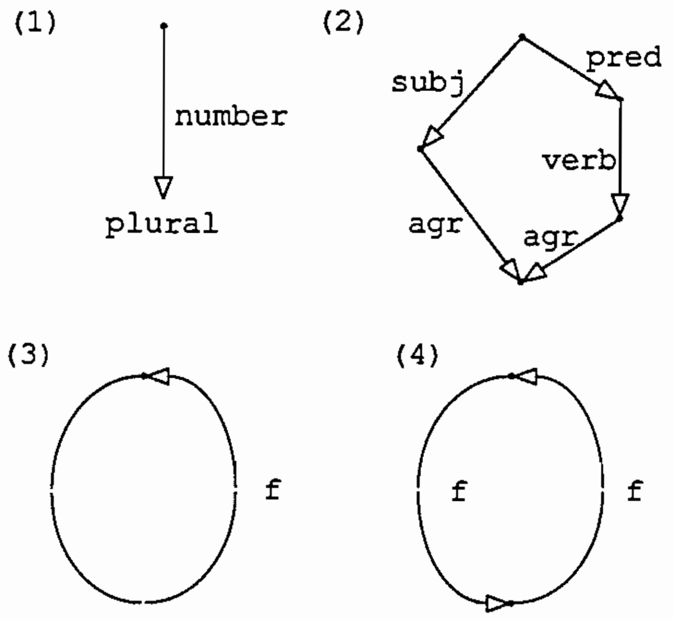

Figure 7: Example Categories

case, the propositions are the consistent equations, the trivial proposition is the trivial equation $\langle\rangle = \langle\rangle$, and the notions of consistency and consequence are as defined above.

We have thus a model of grammatical categories in which a category is identified with a consistent, deductively closed set of path equations. The unification of two categories, if it is possible, corresponds to the deductive closure of the union of the two sets of equations representing the categories. Unification failure corresponds to the inconsistency of that union, that is, two categories fail to unify if and only if a clash occurs the set of consequences of the two categories taken together.

In the following examples, I have omitted from equation sets the trivial equation and one-step applications of the symmetry rule. The four categories depicted in Figure 7 can be represented by the following sets of equations

(1) $\{\langle\texttt{number}\rangle = \texttt{plural}\}$

(2) $\{\langle\texttt{subj agr}\rangle = \langle\texttt{pred verb agr}\rangle\}$

$$(3) \qquad \left\{ \begin{array}{l} \langle\rangle = \langle \mathtt{f} \rangle, \langle \mathtt{f} \rangle = \langle \mathtt{f}\ \mathtt{f} \rangle, \ldots \\ \langle\rangle = \langle \mathtt{f}\ \mathtt{f} \rangle, \ldots \\ \vdots \end{array} \right\}$$

$$(4) \qquad \left\{ \begin{array}{l} \langle\rangle = \langle \mathtt{f}\ \mathtt{f} \rangle, \langle \mathtt{f} \rangle = \langle \mathtt{f}\ \mathtt{f}\ \mathtt{f} \rangle, \ldots \\ \langle\rangle = \langle \mathtt{f}\ \mathtt{f}\ \mathtt{f}\ \mathtt{f} \rangle, \ldots \\ \vdots \end{array} \right\} .$$

It is worth noting that the subsumption ordering for cyclic categories is treated in this model in exactly the same way as the ordering of acyclic categories. In the example above, the equation set of category (3) contains that of category (4), and thus category (3) is an instance of category (4), which corresponds exactly to the subsumption ordering for rational terms [5]. It is also interesting to note that, in contrast with some models of infinite trees in logic programming [13], the present model provides a simple algebraic property distinguishing rational categories (those that are defined by finite sets of equations) from irrational ones, namely that rational categories are exactly the ω-finite elements in the domain of categories. In other words, any rational category will be reached in a finite number of steps by any ascending sequence of categories that converges to the element.

It is very easy to define tree categories associated to the notion of category just introduced. A feature tree is said to satisfy a category if and only if it satisfies all the equations in the category. A tree t satisfies an equation $p = p'$ if and only if t/p and t/p' are defined and $t/p = t/p'$, and it satisfies an equation $p = c$ if and only if t/p is the atomic value c. With a suitable model of infinite trees, it is easy to extend these notions to define categories of rational trees satisfying equational categories.

5 The Kasper-Rounds Logic of Feature Graphs

The notions of grammatical categories discussed so far in this paper do not differ that much from first-order terms in their expressive power. However, certain linguistic analyses, and in fact the whole framework of FUG, require more expressive notions of grammatical category, in which disjunctive classifications can be expressed. In FUG, the natural expression of the agreement and case information for the German definite article form "die" might be

$$\left[\begin{array}{lcl} \texttt{agr} & = & \left\{\begin{array}{l} \left[\begin{array}{lcl} \texttt{number} & = & \texttt{singular} \\ \texttt{gender} & = & \texttt{feminine} \end{array}\right] \\ [\texttt{number} = \texttt{plural}] \end{array}\right\} \\ \texttt{case} = \{\texttt{nom acc}\} & & \end{array}\right]$$

where the square brackets indicate conjunction and the braces disjunction. The meaning of this expression can be given by a formula over path equations

$$\begin{array}{l} [(\langle \texttt{agr number}\rangle = \texttt{singular} \wedge \langle \texttt{agr gender}\rangle = \texttt{feminine})\vee \\ \quad \langle \texttt{agr number}\rangle = \texttt{plural}]\wedge \\ \quad (\langle \texttt{case}\rangle = \texttt{nom} \vee \langle \texttt{case}\rangle = \texttt{acc}) \quad . \end{array}$$

However, the situation becomes more complicated when disjunction and reentrancy interact. To give a precise characterization of the relation between disjunctive categories and feature graphs, Rounds and Kasper [26,18,17] developed a logical language to be interpreted over acyclic feature graphs. Because the properties of the Rounds-Kasper logic are covered in detail in their work, I give only an outline of the syntax and semantics of the logic here to connect it with the other models under discussion.

The language of feature graphs is reminiscent of the Hennessy-Milner modal logic for synchronization trees [10]. Each atomic value in the underlying feature graphs is a formula in the logic. If ϕ and ψ are formulas and f a feature, then $\phi \vee \psi$, $\phi \wedge \psi$ and $f : \phi$ are formulas. The role of the feature prefix $f :$ is that of a possibility modality, which would be written $\langle f\rangle$ in the Hennessy-Milner logic or in propositional dynamic logic [9]. The intended interpretation of $f : \phi$ is that it holds at a node s if and only if $s' = \delta(s, f)$ is defined and ϕ holds at s'. Any finite set of paths $\{p_1, \ldots, p_n\}$ is also a formula in the language. Its intended interpretation is that the paths in the set are *Nerode equivalent* relative to the graph, that is, a single node is reached by following any of those paths from the current node. Finally, the constants *NIL* and *TOP* are formulas in the logic.

The disjunctive category given above can then be represented in this logic by the formula

$$\begin{array}{l} \texttt{agr} : ((\texttt{number} : \texttt{singular} \wedge \texttt{gender} : \texttt{feminine})\vee \\ \quad \texttt{number} : \texttt{plural})\wedge \\ \quad \texttt{case} : (\texttt{nom} \vee \texttt{acc}) \quad . \end{array}$$

The semantics of the logic is as follows, where d is a feature graph with labeling function π and initial state s_0, c is an atomic value, f is a feature, and E is a finite set of paths:

- $d \models NIL$
- $d \not\models TOP$
- $d \models c$ iff s_0 is final and $\pi(s_0) = c$
- $d \models E$ iff s_0/p is the same state for every $p \in E$
- $d \models l : \phi$ iff $d/\langle l \rangle \models \phi$
- $d \models \phi \vee \psi$ iff $d \models \phi$ and $d \models \psi$.

Conjunction plays thus the role of unification in our previous models. The constants *NIL* and *TOP* represent the trivial graph (truth) and unification failure (falsity). Disjunction appears only in the logic, feature graphs are purely conjunctive as in previous models. The identification of unification with conjunction is justified by the fact that the least graph that satisfies a purely conjunctive formula (one without disjunctions) is exactly the graph unification of the least graphs satisfying each of the conjuncts in the formula.

The satisfiability problem for this logic is NP-complete, as is unification because two categories (formulas) unify if and only if their conjunction has a model. However, the intractability of the unification problem can be kept in check in many useful cases by using Kasper's unification algorithm [17], which avoids whenever possible distributing conjunctions over disjunctions, that is, it tries to keep disjunctions as localized as possible in the formula under consideration.

6 Problematic Extensions

As with disjunction, the requirements of grammatical categorization seem to demand additional expressive devices in our models of grammatical categories. Among these, I briefly discuss inheritance, negation and functional uncertainty. It turns out that Rounds-Kasper suggests interesting and rigorous ways of dealing with each of these notions. In each case, I discuss a possible approach; however, it should be clear that some of the technical details need still to be worked out to determine the viability of the proposals.

6.1 Inheritance

Although inheritance appears in linguistic contexts, in lexical subcategorization for instance, the simplest examples are from knowledge representation.

Ait-Kaci gives a thorough account of inheritance for his version of feature graphs [1,2], which was motivated by general needs in knowledge representation and logic programming. However, it is interesting to see whether inheritance fits in the models discussed here.

The basic concept of inheritance is very easy to explain. For example, students with the first name John are a subtype of persons with the first name John. If we were to try to represent these types by feature graphs, we would need a means of labeling, not only final states, but also other states, because from the perspective of their specified features (the value "John" of the name feature), the two types above are identical, but the types themselves are not identical. Following Ait-Kaci's approach, any node in a feature graph can be labeled by a type constant such as person or student, and these type constants partially ordered appropriately.

In a possible extension of feature graphs and the Rounds-Kasper logic to support inheritance, atomic values would be replaced by type constants, which could now label any state. Thus the type "student with name John" would be represented by the formula $\mathtt{student} \land \mathtt{name} : \mathtt{John}$ and the type "person with name John" by the formula $\mathtt{person} \land \mathtt{name} : \mathtt{John}$. The fact that "student" is a subtype of "person" would be represented by the separate formula $\mathtt{student} \Rightarrow \mathtt{person}$. The interpretation of type constants would be analogous to the interpretation of propositions in dynamic logic, where a proposition is identified with the set of states in which it holds; that is, a feature graph would satisfy $\mathtt{student} \Rightarrow \mathtt{person}$ if and only if the set of states labeled student were contained in the set of states labeled person.

6.2 Negation

Like disjunction, some form of negation appears naturally in linguistically motivated accounts of grammatical categories. For example, in a Rounds-Kasper logic with negation, the lexical entry for the word "eat" as a present tense verb form might be

$$\begin{array}{l}\mathtt{agr} : \neg(\mathtt{person} : \mathtt{third} \land \mathtt{number} : \mathtt{singular}) \land \\ \mathtt{tense} : \mathtt{present} \quad .\end{array}$$

As another example, the semantic constraint that the subject and object of a clause cannot be correferential unless the object is a reflexive pronoun might be represented by the formula

$$\neg(\mathtt{obj} : \mathtt{refl} : - \land \{\langle \mathtt{subj\ ref}\rangle, \langle \mathtt{obj\ ref}\rangle\}) \quad .$$

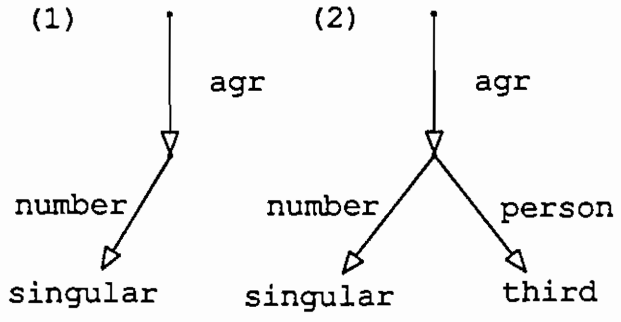

Figure 8: Extending a Graph

The intended partiality of categories makes the question of how negation should be interpreted less obvious that it might seem. According to the classical interpretation of negation, we should have $d \models \neg\phi$ if and only if $d \not\models \phi$. However, this interpretation has the disagreeable consequence that the satisfaction relation is not preserved by instantiation and thus the evaluation of negations cannot be freely interleaved with unifications. For example, graph (1) in Figure 8 satisfies the formula

$$\mathtt{agr} : \neg(\mathtt{person} : \mathtt{third} \wedge \mathtt{number} : \mathtt{singular}) \quad ,$$

but its instance (2) does not.

This problem led Moshier and Rounds [22] to propose a interpretation of negation inspired in Kripke's semantics for intuitionistic logic [20]. In the simplest form of the proposal, which will be qualified later, a graph satisfies $\neg\phi$ if and only if none of its instances satisfies ϕ. Because unification only "moves up" in the subsumption ordering, it is clear from this definition that the satisfiability of negations does not depend on the order in which negations and conjunctions are checked. However, this desirable property of negation comes at a price, namely that complex graphs hardly ever satisfy negative formulas. For example, graph (1) of Figure 9 fails to satisfy

$$\neg\{\langle \mathtt{subj\ agr}\rangle, \langle \mathtt{pred\ verb\ agr}\rangle\}$$

as shown by its instance (2), but it also fails to satisfy

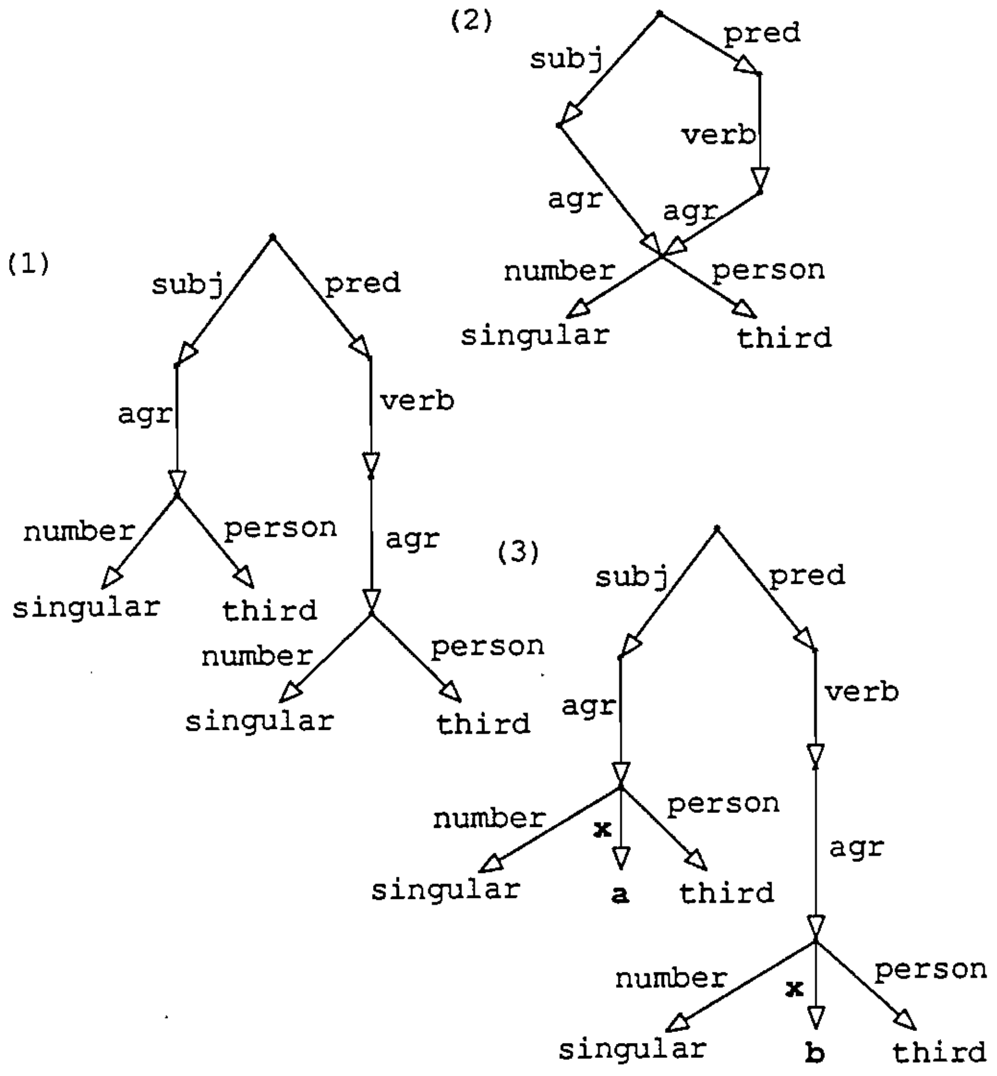

Figure 9: Graph with Incompatible Instances

$$\neg\neg\{\langle \texttt{subj agr}\rangle, \langle \texttt{pred verb agr}\rangle\} \tag{1}$$

as shown by its instance (3).

The problem with this version of intuitionistic negation is that, as shown by the example, even if two subgraphs have at some point exactly the same features and values, they may later by made incompatible by adding to the two subgraphs some arbitrary new feature with incompatible values. However, this addition of arbitrary features to nodes seems artificial. In our informal understanding of the agr feature, it has number and person subfeatures, but clearly no others. Thus the problem is that the model allows the addition of arbitrary features to nodes where we want to circumscribe the possible features of a node to only the meaningful ones.

Moshier and Rounds address the above difficulty by defining satisfaction relative to a set of feature graphs that are allowed as instances of a graph when interpreting negation, following the model-theoretic notion of forcing [7]. For instance, in the example of Figure 9, we might only allow instances in which the features allowed in the value of agr are number and person with atomic values, thus satisfying formula (1). However, from a practical point of view, the forcing construction is not sufficient because it does not give us a formal language for describing the classes of allowed instances for a feature graph. One possible attack on this problem would involve defining a language of *types* and giving types to features to constrain the classes of graphs allowed as their values.

It is interesting to note the close connection between the intuitionistic interpretation of negation in feature graphs and the implementation of inequality (dif) in Prolog-II [5]. In Prolog-II, the execution of an inequality between terms x and y is delayed until the terms are sufficiently instantiated to either verify the inequality or replace it by a disjunction of inequalities of subterms of the given terms. This is possible because the theoretical model states that x and y are different if and only if they have no common instance, which means that they are different if and only if they either have different principal function symbols or have the form $f(x_1, \ldots, x_n)$ and $f(y_1, \ldots, y_n)$ respectively and x_i is different from y_i for some i.[3] The identification of inequality with lack of a common instance is a direct counterpart or the intuitionistic notion of negation for feature graphs. However, the conversion of inequalities between terms into finite disjunctions of inequalities between their subterms is not available for feature graphs because complex feature graph nodes do not have a definite arity. The definite arity of Prolog terms

[3] I am assuming acyclic terms here, even though Prolog-II also supports rational terms.

makes it thus possible to discharge inequalities much sooner, and the proposal of feature types above is a way of imposing a similar property on feature graphs.

6.3 Functional Uncertainty

Long-distance dependencies in natural-language, such as the dependency between the interrogative pronoun "whom" and the direct object of the verb "meet" in

Whom$_i$ did John ask Mary to promise Susan to meet$_i$?

are normally captured in logic grammars by having rules pass around non-terminal arguments encoding the dependency. As there is some uniformity in the use of such arguments, it is possible to use syntactic sugar to disguise their repeated occurrence [24,6], but the approach lacks modularity because it spreads the treatment of one phenomenon, long-distance dependencies, throughout the grammar.

Kaplan and Zaenen's notion of *functional uncertainty* [16,15] leads to improved modularity in the expression of long distance dependencies by allowing the direct expression of dependencies as constraints on categories.

The basic idea of functional uncertainty is to move from features and paths to arbitrary regular expressions over features, called *regular paths*. Thus, the dependency between "whom" and the direct object of "meet" in the example above might be expressed by the regular-path equation

$$\langle \texttt{wh} \rangle = \langle \texttt{pred}\ \texttt{to-compl}^{+}\ \texttt{obj} \rangle \quad ,$$

which should be interpreted as stating that the value of the left-hand side is the same as the value at the end of some path in the set of paths defined by the regular expression in the right-hand side.

Functional uncertainty might be encompassed within the Rounds-Kasper framework by generalizing feature modalities and paths to regular expressions. With r a regular path expression, R a finite set of regular path expressions, $[r]$ the path language defined by the regular expression r, d a feature graph and s_0 its initial state, the semantics of regular path expressions might be given by the following revised semantic equations

- $d \models r : \phi$ iff there is $p \in [r]$ such that $d/p \models \phi$
- $d \models R$ iff $(\exists s)(\forall r \in R)(\exists p \in [r]) s_0/p = s$, that is, it is possible to find a set of Nerode-equivalent paths in which each path is taken from the language of a different expression in R.

However, the properties of the extended language are still unclear.

7 Further Work

In this survey, I have tried to give the flavor of recent research on the notion of grammatical category and its connections with corresponding concepts in the semantics of logic programs. Although I have emphasized the linguistic motivation of this work, similar questions arise in developing models of partially specified data structures in programming languages and knowledge representation. I finish with a number of suggestions of work that I think would further clarify our understanding of partial objects and their role in logic programming. The suggestions are ordered from the more specific and concrete to the more general and speculative.

First, inheritance, negation and functional uncertainty, as discussed in the last section, are by no means fully understood. In particular, it should be possible to give a precise characterization of the properties of the proposed extensions of the Rounds-Kasper logic to handle inheritance and functional inheritance. The work of Moshier and Rounds suggests the need for introducing a notion of types in the formal language of grammatical categories. Also, the Rounds-Kasper logic is currently restricted to the description of acyclic graphs, and it is not clear how it could be extended to the rational case, particularly if functional uncertainty is also present [14].

With respect to the denotational model discussed in Section 4, we need to show its exact relationship to the graph models. There should be some analogy here to the connection between sets of equations and their solution sets for rational terms, but partiality may cause difficulties.

The graph models of grammatical categories are too concrete because they involve the explicit presentation of categories as abstractions of data structures. In particular, Rounds-Kasper logic does not describe categories directly but instead goes indirectly through a specific graph model of categories. This situation is reminiscent of older logic programming models, in which programs are interpreted on the syntactic domain of the Herbrand universe or on its infinite term extension, rather than in terms of the specific information domains of interest to the user, such as the real numbers. This need for a more abstract way of modeling grammatical categories leads to the last and most long term suggestion, namely the application of the ideas of constraint logic programming [12] to the definition of grammatical categories and their constraints, which is suggested by the fact that constraint logic programming is precisely the result of the move to abstract away from syntactic domains in logic programming.

Acknowledgments

This research was made possible by a gift from the Systems Development Foundation. Stuart Shieber suggested important corrections and improvements of presentation, and the material of Section 4 is based on our joint work. I also thank Mark Johnson, Lauri Karttunen, Robert Kasper, Martin Kay and William Rounds for many conversations over the last four years that helped build my understanding of unification grammar formalisms. The remaining errors and obscurities, are, of course, my own.

References

[1] H. Ait-Kaci. *A Lattice Theoretic Approach to Computation based on a Calculus of Partially Ordered Type Structures.* PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 1984.

[2] H. Ait-Kaci and R. Nasr. LOGIN: a logic programming language with built-in inheritance. *Logic Programming*, 3(3):185–217, 1986.

[3] J. Bresnan and R. Kaplan. Lexical-functional grammar: a formal system for grammatical representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281, MIT Press, Cambridge, Massachusetts, 1982.

[4] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, 1985.

[5] A. Colmerauer. Theoretical model of Prolog II. In M. van Caneghem and D. H. D. Warren, editors, *Logic Programming and Its Applications*, chapter 1, pages 3–31, Ablex, Norwood, New Jersey, 1986.

[6] V. Dahl and M. McCord. Treating coordination in logic grammars. *Computational Linguistics*, 9(2):69–91, 1983.

[7] M. Fitting. *Intuitionistic Logic and Model Theoretic Forcing.* North-Holland, Amsterdam, Holland, 1969.

[8] G. Gazdar, E. Klein, G. K. Pullum, and I. A. Sag. *Generalized Phrase Structure Grammar.* Harvard University Press, Cambridge, Massachusetts, 1985.

[9] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic, Vol. II*, chapter II.10, pages 497–604, D. Reidel, Dordrecht, Holland, 1984.

[10] M. Hennessy and R. Milner. On observing nondeterminism and concurrency. In *Automata, Languages and Programming*, pages 299–309, Springer-Verlag, Berlin, Germany, 1980.

[11] J. Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2(3):207–219, 1984.

[12] J. Jaffar and J. Lassez. Constraint logic programming. In *ACM Symposium on Principles of Programming Languages*, Association for Computing Machinery, Munich, Germany, 1987.

[13] J. Jaffar and P. J. Stuckey. Semantics of infinite tree logic programming. *Theoretical Computer Science*, 46:141–158, 1986.

[14] M. Johnson. Computing with regular path formulas. October 1986. Draft.

[15] R. M. Kaplan and J. Maxwell. Functional uncertainty. Forthcoming.

[16] R. M. Kaplan and A. Zaenen. Wh-constructions and constituent structure. In M. Baltin and A. Kroch (eds.) *Alternative Conceptions of Phrase Structure*, forthcoming.

[17] R. T. Kasper. *Feature Structures: A Logical Theory with Application to Language Analysis.* PhD thesis, University of Michigan, Ann Arbor, Michigan, 1987.

[18] R. T. Kasper and W. C. Rounds. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting*, pages 257–266, Association for Computational Linguistics, Columbia University, New York, 1986.

[19] M. Kay. Parsing in functional unification grammar. In D. R. Dowty, L. Karttunen, and A. M. Zwicky, editors, *Natural Language Parsing*, chapter 7, pages 251–278, Cambridge University Press, Cambridge, England, 1985.

[20] S. Kripke. Semantical analysis of intuitionistic logic. In J. N. Crossley and M. A. E. Dummett, editors, *Formal Systems and Recursive Functions*, pages 92–130, North-Holland, Amsterdam, Holland, 1965.

[21] J. Lassez and K. Marriot. Explicit representation of terms defined by counter examples. *Automated Reasoning*, 1987. Forthcoming.

[22] M. D. Moshier and W. C. Rounds. A logic for partially specified data structures. In *ACM Symposium on the Principles of Programming Languages*, Association for Computing Machinery, Munich, Germany, 1987.

[23] K. Mukai and H. Yasukawa. Complex indeterminates in Prolog and their application to discourse models. *New Generation Computing*, 3(4):441–466, 1985.

[24] F. C. N. Pereira. Extraposition grammars. *Computational Linguistics*, 7(4):243–256, October-December 1981.

[25] F. C. N. Pereira and S. M. Shieber. The semantics of grammar formalisms seen as computer languages. In *Proc. of Coling84*, pages 123–129, Association for Computational Linguistics, 1984.

[26] W. C. Rounds and R. Kasper. A complete logical calculus for record structures representing linguistic information. In *Symposium on Logic in Computer Science*, IEEE Computer Society, 1986.

[27] D. S. Scott. Domains for denotational semantics. In *ICALP 82*, Springer-Verlag, Heidelberg, 1982.