ROBERT T. KASPER AND WILLIAM C. ROUNDS

THE LOGIC OF UNIFICATION IN GRAMMAR[1]

1. Introduction

By *unification*, we understand a family of algorithms employed by computational versions of certain grammar formalisms to combine information in *feature structures*. The use of these formalisms has become widespread, and several extensions to the basic notion of feature structure have been proposed. Although algorithms for unification of these extended feature structures have been written, they are complicated, and a precise model of feature structures is desirable to give an adequate specification of what the algorithms do. We have developed a model in which *descriptions* of feature structures can be regarded as logical formulas, and interpreted by sets of directed graphs which satisfy them. We identify a feature structure with such a directed graph, but our mathematical work is facilitated by considering the graphs to be transition graphs for a special type of deterministic finite automaton.

This semantics for feature structures extends the ideas of Pereira and Shieber [11], by providing a way to model feature values which are specified by *disjunctions* and *non-local path values* embedded within disjunctions. Our interpretation differs from that of Pereira and Shieber by using a logical model in place of a denotational semantics. The model yields a calculus of equivalences between formulas, which can be used to simplify them. A similar use of logic to describe feature structures was first proposed in Generalized Phrase Structure Grammar (GPSG) [3], in order to describe feature co-occurrence restrictions. Our formulation, which was developed independently, is for the purpose of understanding the properties of unification. Recently Gazdar et al. [2] have extended their logic in order to give uniform descriptions of linguistic categories. The two logics have much in common when presented formally, and it should be possible to combine them in a uniform way. This, however, will be left for future work.

Unification is attractive, because of its generality, but it can be computationally inefficient. Kay's treatment [10] of unification in Functional Unification Grammar is a representative example. He analyzes descriptions

[1] This work represents revision and extension of research reported in [12] and [7].

containing disjunctions by expanding them to a kind of *disjunctive normal form*, a process which introduces an exponential explosion in both space and time. Our model allows a careful examination of the computational complexity of unification. What we show is that the generalized notion of unification can be phrased as the problem of telling whether or not one of our formulas is satisfiable. We show that the satisfiability problem for formulas with disjunctive values is NP-complete. This problem is essentially the one which must be solved for each parse, so one needs to solve it as efficiently as possible. It is not surprising that Kay's algorithm is exponential, because many people believe that NP-complete problems cannot be solved in polynomial time in the worst case. However, NP-complete problems can often be solved efficiently on the average, and we describe how disjunctive values can be specified in a way in which expansion to disjunctive normal form may be delayed. Although this expansion may be desirable to interpret final results, a full expansion is not needed during unification, resulting in a substantial improvement in computational efficiency. In fact, Kasper's thesis contains an algorithm which, in many cases arising in practice, solves the consistency problem in cubic time [8]. Thus the formalism shows how to solve a seemingly impossibly complex problem in a reasonably efficient manner.

It is worth emphasizing that we might not have discovered any of these algorithms or equivalences without giving a precise formal definition both to feature structures and to their descriptions. We hope that the calculus we have developed will continue to be useful. In a sense, the calculus is a kind of Boolean algebra, not about Boolean-valued functions and switching circuits, but about finite partial functions over a finite set of atomic values. The intent of the calculus is still the same, however: to allow flexibility in the design and specification of algorithms dealing with feature structures, while at the same time ensuring that these algorithms are correctly specified.

The paper is organzied into four parts. In Section 2 we discuss the use of feature structures in various linguistic and computational theories. We show how our formalization using automata is like the formalism in GPSG [3]. We also explain the distinction between the feature structures themselves and descriptions of feature structures. In FUG, for example, we view feature matrices containing disjunctive values not as feature structures, but as assertions about such structures. It is this distinction which allows us to make sense of disjunctive values in general. Section 3 contains the logical formalism itself, and a presentation of the equivalences satisfied by our logical formulas. In Section 4 we show how our formalism solves the problem of *non-local values*, one of the thorny computational issues in FUG. Finally, Section 5 contains some of the mathematical properties of our system. In particular, we show that the laws of Section 3 are complete: any true equivalence of our logic can be provided using just these laws. (Thus we have another analogy with Boolean algebra.)

2. Background: unification in grammar

Several different approaches to natural language grammar have developed the notion of feature structures to describe linguistic objects. These approaches include linguistic theories, such as Generalized Phrase Structure Grammar (GPSG) [3], Lexical Functional Grammar (LFG) [5], and Systemic Grammar [4]. They also include grammar formalisms which have been developed as computational tools, such as Functional Unification Grammar (FUG) [9], and PATR-II [13]. In these computational formalisms, *unification* is the primary operation for matching and combining feature structures.

Feature structures are called by several different names, including *f-structures* in LFG, and *functional descriptions* in FUG. Although they differ in details, each approach uses structures containing sets of attributes. Each attribute is composed of a label/value pair. A value may be an atomic symbol, but it may also be a nested feature structure.

The intuitive interpretation of feature structures may be clear to linguists who use them, even in the absence of a precise definition. Often, a precise definition of a useful notation becomes possible only after it has been applied to the description of a variety of phenomena. Then, greater precision may become necessary for clarification when the notation is used by many different investigators. Our model has been developed in the context of providing a precise interpretation for the feature structures which are used in FUG and PATR-II. Some elements of this logical interpretation have been partially described in Kay's work [10]. We extend this work to give a complete algebraic account of the logical properties of feature structures, which can be used explicitly for computational manipulation and mathematical analysis.

2.1. *Disjunction and Non-Local Values*

Karttunen [6] has shown that disjunction and negation are desirable extensions to PATR-II which are motivated by a wide range of linguistic phenomena. He discusses specifying attributes by disjunctive values, as shown in Figure 1. A *value disjunction* specifies alternative values of a single

$$\text{die} = \begin{bmatrix} \textit{case} : \{\textit{nom acc}\} & \\ \textit{agreement}: & \left\{ \begin{matrix} \begin{bmatrix} \textit{gender}: \textit{fem} \\ \textit{number}: \textit{sing} \end{bmatrix} \\ [\textit{number}: \textit{pl}] \end{matrix} \right\} \end{bmatrix}$$

Fig. 1. A feature matrix containing value disjunction.

attribute. These alternative values may be either atomic or complex. Thus in Figure 1, which describes lexical properties of the German article 'die', we learn that the word has an *agreement* feature of possibly two different kinds: either feminine singular, or plural. Disjunction of a more general kind is an essential element of FUG. *General disjunction* is used to specify alternative groups of multiple attributes, as shown in Figure 2.

Karttunen describes a method by which the basic unification procedure can be extended to handle disjunctive values, and explains some of the complications that result from introducing value disjunction. When two values, A and B, are to be unified or combined, and A is a disjunction, we cannot actually unify B with both alternatives of A, because one of the alternatives may become incompatible with B through later unifications in the course of parsing. Instead we need to remember a constraint that at least one of the alternatives of A must remain compatible with B.

An additional complication arises when one of the alternatives of a

$$\begin{bmatrix} \text{cat} = \text{s} \\ \text{subj} = [\text{case} = \text{nominative}] \\ \left\{ \begin{matrix} \begin{bmatrix} \text{voice} = \text{active} \\ \text{actor} = \langle\text{subj}\rangle \end{bmatrix} \\ \begin{bmatrix} \text{voice} = \text{passive} \\ \text{goal} = \langle\text{subj}\rangle \\ \text{adjunct} = \begin{bmatrix} \text{cat} = \text{pp} \\ \text{prep} = \text{by} \\ \text{obj} = \langle\text{actor}\rangle = [\text{case} = \text{objective}] \end{bmatrix} \end{bmatrix} \end{matrix} \right\} \\ \left\{ \begin{matrix} [\ \text{mood} = \text{declarative}] \\ [\ \text{mood} = \text{interrogative}] \end{matrix} \right\} \end{bmatrix}$$

Fig. 2. Disjunctive specification containing non-local values, using the notation of FUG.

disjunction contains a value which is specified by a non-local path, a situation which occurs frequently in Functional Unification Grammar. Consider Figure 2.

From this description, we learn the following information. In a sentence, the subject has nominative case. Additionally, the voice aspect of the sentence can be active or passive, and the mood can be declarative or interrogative. If the voice is active, then the *actor* is identified with the subject. If the voice is passive, however, then the *goal* is identified with the subject, and is therefore the constituent with nominative case. Also, in the passive voice, the actor *of the whole sentence* is the object of the preposition 'by' in the adjunct prepositional phrase, and must have objective case. The non-local values are enclosed in angle brackets. Each such value is an attribute of the entire sentence, even though that value may be found at an inner level of the description. Consider now the value, say v, of the object of the adjunct phrase. The (nonlocal) value of the actor attribute of the whole sentence must be unified, or made equal to, the value v. During the parsing of a sentence, this unification with a non-local value can be performed only when the alternative which contains it (i.e., the passive alternative) is the only alternative remaining in the disjunction. Otherwise, the *case* = *objective* attribute might be added to the value of ⟨*actor*⟩ prematurely, when the active alternative is the one to be used. This would require the subject to have both nominative and objective case, which would be incorrect. Thus, the constraints on alternatives of a disjunction must also apply to any non-local values contained within those alternatives. These complications, and the resulting proliferation of constraints, tend to lead to mistakes in the programming of these systems, and this fact provides a practical motivation for the logical treatment given in this paper. We suggest a solution to the problem of representing non-local path values in Section 4.

2.2. *What is a Feature Structure*?

As Pereira and Shieber [11] have pointed out, a grammatical formalism can be regarded in a way similar to other representation languages. Often it is desirable to use a representation language which is distinct from the objects it represents. Thus, it can be useful to make a distinction between the domain of feature structures and the domain of their descriptions. As we shall see, this distinction allows a variety of notational devices to be used in descriptions, and interpreted in a consistent way with a uniform kind of structure.

The PATR-II system uses directed acyclic graphs (dags) as an underlying representation for feature structures. In order to build complex feature structures, two primitive domains are required:

1. Atoms (A)
2. Labels (L)

The elements of both domains are symbols, usually denoted by character strings. Attribute labels (e.g., 'case') are used to mark edges in a dag, and atoms (e.g., 'gen') are used as primitive values at vertices which have no outgoing edges.

These dag structures are graphical representations of the notion of *categories* found in GPSG. Thus in [3, Chapter 2], and in [2], we find categories described as finite partial functions. The set of attribute labels becomes the set of feature names; and feature values may either be atomic (i.e., in the set A) or may themselves be categories. To quote from [3, page 24]:

... Ignoring feature specifications that include categories as values for the moment, suppose we assume two finite, nonempty sets: F, the set of features, and V, the set of feature-values. Then a syntactic category is a partial function from F into V. We say 'partial' because we often want to allow the possibilitity that a category C is undefined for some particular feature $f \in F$.

It is clear how to think of a dag as a finite partial function of this kind. At the top level, think of the dag as the set $\{\langle f_1, v_1\rangle, \ldots, \langle f_n, v_n\rangle\}$, where the f_i are the arc labels emanating from the root node, and the v_i are the subdags at the end of these arcs. What is not so clear, however, is how to represent the fact that two arcs in a dag can come together at the same node. This property of dags is useful in the unification-based theories, and our logic has a type of formula which says that explicitly named paths may coincide. Ait-Kaci [1] treats exactly this problem, showing how to impose an equivalence relation (the coreference relation) on a finite partial function description so that paths may be regarded as identical. The notation gets complicated, however, and we prefer the representation which follows.

A dag may also be regarded as a transition graph for a partially specified deterministic finite automaton (DFA). This automaton recognizes strings of labels, and has final states which are atoms, as well as final states which encode no information. An automaton is formally described by a tuple

$$\mathscr{A} = (Q, L, \delta, q_0, F)$$

where Q is the set of states of the automaton, L is the set of labels above, δ is a partial function from $Q \times L$ to Q, and where certain elements of F

may be atoms from the set A. We require that $\mathscr{A}$ be connected, acyclic, and have no transitions from any final states.

In this representation, the nodes of a dag are the states of the automaton. The interpretation of the δ function is as follows. Let $l \in L$. Then l denotes a feature and if the root node of the dag is a state q_0, then the state $\delta(q_0, l)$ is the root node of the subdag at the end of the l-arc. In fact we can refer to this subdag by the name $\delta(q_0, l)$, so that the value of the feature l can be given by this very expression. Additionally, if x is a string of feature labels, then the state $\delta(q_0, x)$ is identified with the subdag obtained by following the path x in the obvious way.

DFAs have several desirable properties as a domain for feature structures:

(1) the value of any defined path can be denoted by a state of the automaton;

(2) finding the value of a path is interpreted by running the automaton on the path string;

(3) the automaton captures the crucial properties of shared structure;
   (a) two paths which are unified have the same state as a value,
   (b) unification is equivalent to a state-merge operation;

(4) the techniques of automata theory become available for use with feature structures.

A consequence of item 3 above is that the distinction between type identity and token identity, mentioned in [14] is clearly revealed by an automaton; two objects are necessarily the same token, if and only if they are represented by the same state.

One construct of automata theory, the Nerode relation, is useful to describe equivalent paths. If $\mathscr{A}$ is an automaton, we let $P(\mathscr{A})$ be the set of all paths of $\mathscr{A}$, namely the set $\{x \in L^* : \delta(q_0, x) \text{ is defined}\}$. The *Nerode relation* $N(\mathscr{A})$ is the equivalence relation defined on paths of $P(\mathscr{A})$ by letting

$$xN(\mathscr{A})\, y \Leftrightarrow \delta(q_0, x) = \delta(q_0, y).$$

This completes our discussion of feature structures. Our logic will describe such structures. We should re-emphasize that the structures themselves are wholly functional. Disjunctive values are not allowed in the structures; disjunction enters into the description of the structures. Taking this point of view has allowed us to give a consistent formalization of the kind of disjunction implicit in FUG, but it is probably not the only way to formalize disjunctive values. Our initial attempts to formalize 'nondeter-

NIL
TOP
a where $a \in A$
$[\![\langle p_1 \rangle, \ldots, \langle p_n \rangle]\!]$ where each $p_i \in L^*$
$l : \phi$ where $l \in L$ and $\phi \in$ FDL
$\phi \wedge \psi$
$\phi \vee \psi$

Fig. 3. The domain, FDL, of logical formulas.

ministic' feature structures were unsuccessful, however, and for this reason we have adopted the present approach.

3. Domain of descriptions: logical formulas

We now define the domain FDL (Feature Description Logic) of logical formulas which describe feature structures. Figure 3 defines the syntax of well formed formulas. In the following sections symbols from the Greek alphabet are used to stand for arbitrary formulas in FDL. The formulas *NIL* and TOP are intended to convey 'no information' and 'inconsistent information' respectively. Thus, *NIL* describes a unification variable; any structure will satisfy it, and the minimal such structure is a degenerate automaton with just one state which has no atomic value. Similarly, *TOP* corresponds to unification failure, in that no structure will satisfy it. A formula $l : \phi$ would indicate that a structure has attribute l, the value of which is a structure satisfying the condition ϕ.

Conjunction and disjunction will have their ordinary logical meaing as operators in formulas. An interesting result is that conjunction can be used to describe unification. Unifying two structures requires finding a structure which has all features of both structures; the conjunction of two formulas describes the structures which satisfy all conditions of both formulas. Also, a difference between feature structures and their descriptions should be noted. In a feature structure it is required that a particular attribute have a unique value, while in descriptions it is possible to specify, using conjunction, several values for the same attribute, as in the formula

$$\text{subj} : (\text{person} : 3) \wedge \text{subj} : (\text{number} : \text{sing}).$$

A feature structure satisfying such a description will contain a unique value for the attribute, which can be found by unifying all of the values that are specified in the description. We will illustrate these points in a later section.

Formulas may also contain sets of paths, denoting equivalence classes. Each element of the set represents an existing path starting from the initial state of an automaton, and all paths in the set are required to have a common endpoint. If $E = [\![\langle x\rangle, \langle y\rangle]\!]$, we can think of E as the equation $\langle x\rangle = \langle y\rangle$. This is the notation of PATR-II for pairs of equivalent paths. In subsequent sections we use E (sometimes with subscripts) to stand for a set of paths that belong to the same equivalence class. Also, for brevity we will sometimes omit the angle brackets around path strings.

3.1. *Interpretation of Formulas*

We can now state inductively the exact conditions under which an automaton $\mathscr{A}$ satisfies a formula:

(1) $\mathscr{A} \models \mathrm{NIL}$ always;
(2) $\mathscr{A} \models \mathrm{TOP}$ never;
(3) $\mathscr{A} \models a \Leftrightarrow \mathscr{A}$ is the one-state automaton a with no transitions;
(4) $\mathscr{A} \models E \Leftrightarrow E$ is a subset of an equivalence class of $N(\mathscr{A})$;
(5) $\mathscr{A} \models l : \phi \Leftrightarrow \mathscr{A}/l$ is defined and $\mathscr{A}/l \models \phi$;
(6) $\mathscr{A} \models \phi \vee \psi \Leftrightarrow \mathscr{A} \models \phi$ or $\mathscr{A} \models \psi$;
(7) $\mathscr{A} \models \phi \wedge \psi \Leftrightarrow \mathscr{A} \models \phi$ and $\mathscr{A} \models \psi$.

where $\mathscr{A}/l$ is defined by a subgraph of the automaton $\mathscr{A}$ with start state $\delta(q_0, l)$; that is, if $\mathscr{A} = (Q, L, \delta, q_0, F)$, then $\mathscr{A}/l = (Q', L, \delta, \delta(q_0, l), F')$, where Q' and F' are formed from Q and F by removing any states which are unreachable from $\delta(q_0, l)$.

Any formula can be regarded as a specification for the set of automata which satisfy it. In the case of conjunctive formulas (containing no occurrences of disjunction) the set of automata satisfying the formula has a unique minimal element, with respect to subsumption. For disjunctive formulas there may be several minimal elements, but always a finite number.[2]

3.2. *A Calculus of Equivalences for Formulas*

It is possible to write many formulas which have an identical interpretation. For example, the two formulas given in the equation below are

[2] This fact, and the formal definition of the subsumption order, will be discussed in Section 5.

satisfied by the same set of automata.

$$\text{case} : (\text{gen} \vee \text{acc} \vee \text{dat}) \wedge \text{case} : \text{acc} = \text{case} : \text{acc}$$

In this simple example it is clear that the right side of the formula is equivalent to the left side, and that it is simpler. In more complex examples it is not always obvious when two formulas are equivalent, i.e., when they have the same set of satisfying structures.

In this section we will show how general observations about operations on feature structures can be stated formally as laws of logical equivalence for formulas.

Conjunction models unification. Unifying two structures requires finding a structure which has all features of both structures; the conjunction of two formulas describes the structures which satisfy all conditions of both formulas.

Since unifying two incompatible structures results in a failure, it should be the case that conjoining two descriptions of incompatible values yields an unsatisfiable formula. Thus, the following equivalences should hold:

$$a \wedge b = TOP, \quad \forall a, b \in A \text{ where } a \neq b$$
$$a \wedge l : \phi = TOP.$$

Since a feature structure must have a unique value for any attribute, it should also be the case that conjoining two descriptions containing specifications of the same attribute should require the values specified for that attribute to be conjoined, as stated in:

$$l : \phi \wedge l : \psi = l : (\phi \wedge \psi).$$

We return to the distinction between structures and their descriptions noted in the previous section: a feature structure is required to have a unique value for each attribute, while in descriptions it is possible to specify, using conjunction, several values for the same attribute, as in the formula

$$\text{subj} : (\text{person} : 3) \wedge \text{subj} : (\text{number} : \text{sing})$$
$$\wedge\ \text{subj} : (\text{gender} : \text{neut}).$$

A feature structure satisfying this description contains just one value for the *subj* attribute, which can be found by unifying all of the values that are specified for it in the description. The minimal satisfying feature structure is shown in Figure 4.

Disjunctive values can be converted to general disjunctions. In FUG disjunctions are often used to denote alternative portions to be included

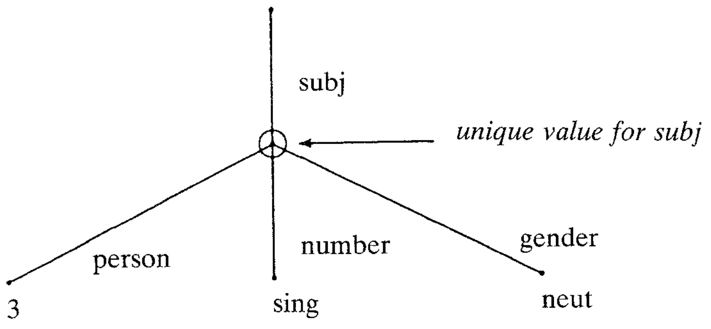

Fig. 4. Feature structures have unique values for their attributes.

in a functional description containing many attributes. It is also possible to specify the value of a particular attribute as a disjunction. If that value contains non-local dependencies, then it may be desirable to factor the label of that attribute into the disjuncts, when comparing the disjuncts with other descriptions. A description of the attribute l containing a disjunctive value is equivalent to a disjunction where the label l is prefixed to each disjunct of the value describing the attribute. Stated formally, we have the following law of equivalence:

$$l : \phi \vee l : \psi = l : (\phi \vee \psi).$$

Distributive laws hold for conjunction and disjunction. If conjunction were regarded operationally as unification, then it would not be superficially obvious that the distributive laws of ordinary propositional logic hold for the formulas of FDL. Yet they are clearly true in the model defined here. These laws are crucial for simplifying formulas containing disjunction:

$$(\phi \vee \psi) \wedge \chi = (\phi \wedge \chi) \vee (\psi \wedge \chi)$$
$$(\phi \wedge \psi) \vee \chi = (\phi \vee \chi) \wedge (\psi \vee \chi).$$

The equations stated above can be regarded as laws of equivalence with respect to the interpretation of formulas given earlier. Once these laws are established, then they can be used as a basis for simplifying formulas, by converting them to a form which represents the same facts more efficiently. Several other laws can be added to form a complete calculus of equivalences. A complete set of equivalences for formulas of FDL is shown in Figure 5.

The propositional laws (1–19) are clearly true in the intended interpretation. Some explanation of the laws for path equivalence classes, how-

Failure:

$$l : \mathrm{TOP} = \mathrm{TOP} \tag{1}$$

Conjunction (unification):

$$\phi \wedge \mathrm{TOP} = \mathrm{TOP} \tag{2}$$
$$\phi \wedge \mathrm{NIL} = \phi \tag{3}$$
$$a \wedge b = \mathrm{TOP},\ \forall a, b \in A \text{ and } a \neq b \tag{4}$$
$$a \wedge l : \phi = \mathrm{TOP} \tag{5}$$
$$l : \phi \wedge l : \psi = l : (\phi \wedge \psi) \tag{6}$$

Disjunction:

$$\phi \vee \mathrm{NIL} = \mathrm{NIL} \tag{7}$$
$$\phi \vee \mathrm{TOP} = \phi \tag{8}$$
$$l : \phi \vee l : \psi = l : (\phi \vee \psi) \tag{9}$$

Commutative:

$$\phi \wedge \psi = \psi \wedge \phi \tag{10}$$
$$\phi \vee \psi = \psi \vee \phi \tag{11}$$

Associative:

$$(\phi \wedge \psi) \wedge \chi = \phi \wedge (\psi \wedge \chi) \tag{12}$$
$$(\phi \vee \psi) \vee \chi = \phi \vee (\psi \vee \chi) \tag{13}$$

Idempotent:

$$\phi \wedge \phi = \phi \tag{14}$$
$$\phi \vee \phi = \phi \tag{15}$$

Distributive:

$$(\phi \vee \psi) \wedge \chi = (\phi \wedge \chi) \vee (\psi \wedge \chi) \tag{16}$$
$$(\phi \wedge \psi) \vee \chi = (\phi \vee \chi) \wedge (\psi \vee \chi) \tag{17}$$

Absorption:

$$(\phi \wedge \psi) \vee \phi = \phi \tag{18}$$
$$(\phi \vee \psi) \wedge \phi = \phi \tag{19}$$

Path Equivalence:

$$E_1 \wedge E_2 = E_2 \text{ whenever } E_1 \subseteq E_2 \tag{20}$$
$$E_1 \wedge E_2 = E_1 \wedge (E_2 \cup \{zy \mid z \in E_1\}) \text{ for any } y \text{ such that } \exists x : x \in E_1 \text{ and } xy \in E_2 \tag{21}$$
$$E \wedge x : c = E \wedge \left(\bigwedge_{y \in E} y : c \right) \text{ where } x \in E \tag{22}$$
$$E = E \wedge \{x\} \text{ if } x \text{ is a prefix of a string in } E \tag{23}$$
$$l : E = \{lw \mid w \in E\} \tag{24}$$
$$\{\epsilon\} = \mathrm{NIL} \tag{25}$$
$$E = \mathrm{TOP} \text{ for any } E \text{ such that there are strings } x,\ xy \in E \text{ and } y \neq \epsilon \tag{26}$$

Fig. 5. Laws of equivalence for formulas.

ever, is in order. For example, law 21 states that the Nerode equivalence relation is a right congruence relation with respect to defined transitions. The formulas E_1 and E_2 state that all path strings in those respective sets lead to the same point in the feature structure. If there is a string x in E_1 with an extension $xy \in E_2$, then x is congruent to all strings in E_1, and so, since the Nerode relation is a right congruence, xy must be equivalent to zy. We have used the E notation for reasons of mathematical succinctness;

we could have as well introduced simple path equations $x = y$, which would have made for readability, at the cost of making the axioms a bit clumsier.

Some laws may be regarded as optional depending on the type of structures which are being described. For example, the equivalence (26) is added to prevent cyclic graphs by making descriptions of cyclic structures unsatisfiable.

We will discuss the meaning of completeness in Section 5.

4. Solution of the non-local value problem

The logic presented in the previous section contains no direct representation for non-local path values of the type described in FUG descriptions. Recall that in Functional Unification Grammar a non-local path denotes the value found by extracting each of the attributes labeled by the path in successively embedded feature structures, beginning with the entire structure currently under consideration. In order to give the intended interpretation for such values, it is necessary to make reference to the global context of the structure in which they occur.

In this section the logic will be extended to include descriptions containing non-local values. It will also be shown that these descriptions can be converted into formulas of the simpler logic by using only path equivalence classes to represent the non-local dependencies.

4.1. *Extending the Logic*

The syntax for expressing a non-local path value will be a path expression enclosed in angle brackets:

$$\langle p \rangle \text{ where } p \in L^*.$$

This type of formula denotes the value found by traversing the path p in a satisfying feature structure. Adding this type of formula to the other types previously defined, we have the well-formed formulas of extended FDL, as shown in Figure 6.

The domain of feature structures remains the same as before, but the satisfaction relation must be extended with structures which represent the global context. This is done by specifying pairs consisting of automata and one of their states as structures which satisfy a formula:

(1) $(\mathscr{A}, q) \models \text{NIL}$ always;

(2) $(\mathscr{A}, q) \models TOP$ never;

(3) $(\mathscr{A}, q) \models a \Leftrightarrow q$ is a final state denoted by the atom a with no outgoing transitions;

NIL
TOP
a where $a \in A$
$[\![\langle p_1\rangle, \ldots, \langle p_n\rangle]\!]$ where each $p_i \in L^*$
$\langle p\rangle$ where $p \in L^*$
$l : \phi$ where $l \in L$ and $\phi \in \mathrm{FDL}$
$\phi \wedge \psi$
$\phi \vee \psi$

Fig. 6. The domain, extended FDL, of logical formulas.

(4) $(\mathscr{A}, q) \models E \Leftrightarrow \forall p \in E, \delta(q, p)$ gives the same value;
(5) $(\mathscr{A}, q) \models \langle p\rangle \Leftrightarrow \delta_{\mathscr{A}}(q_0, p) = q$;
(6) $(\mathscr{A}, q) \models l : \phi \Leftrightarrow \delta(q, l)$ is defined and $(\mathscr{A}, \delta(q, l)) \models \phi$;
(7) $(\mathscr{A}, q) \models \phi \vee \psi \Leftrightarrow (\mathscr{A}, q) \models \phi$ or $(\mathscr{A}, q) \models \psi$;
(8) $(\mathscr{A}, q) \models \phi \wedge \psi \Leftrightarrow (\mathscr{A}, q) \models \phi$ and $(\mathscr{A}, q) \models \psi$.

Given a pair $(\mathscr{A}, q)$, the subautomaton of $\mathscr{A}$ having q as an initial state can be regarded as the part of the structure being described by the formula, and the automaton $\mathscr{A}$ can be regarded as the global context of the entire structure. We may recapture the original notion of satisfaction by stipulating that $\mathscr{A} \models \phi$ if and only if $(\mathscr{A}, q_0) \models \phi$ in the new sense. Note that this interpretation of formulas differs significantly from the simpler one only in two clauses, those pertaining to formulas of the type $\langle p\rangle$ and the type $l : \phi$, above. The clause for $\langle p\rangle$ is the only one which refers to the global context, and the clause for $l : \phi$ is the only one which changes the state under consideration.

It can be shown that the laws of equivalence given in Section 5 still hold for the extended logic. This should be obvious for the laws which have no mention of labeled formulas, since the satisfiability conditions of other types of formulas do not refer to the global context, and are essentially identical to the conditions of the simpler logic. For proof of the laws involving labeled formulas, consider (6): $l : \phi \wedge l : \psi = l : (\phi \wedge \psi)$. Given $(\mathscr{A}, q) \models l : \phi \wedge l : \psi$, we can derive the following:

$(\mathscr{A}, q) \models l : \phi$ and $(\mathscr{A}, q) \models l : \psi$
$\Leftrightarrow \delta(q, l)$ is defined, and $(\mathscr{A}, \delta(q, l)) \models \phi$ and $(\mathscr{A}, \delta(q, l)) \models \psi$
$\Leftrightarrow \delta(q, l)$ is defined, and $(\mathscr{A}, \delta(q, l)) \models \phi \wedge \psi$
$\Leftrightarrow (\mathscr{A}, q) \models l : (\phi \wedge \psi)$ (as desired).

Law (9) and the path equivalence laws can be derived in an analogous manner.

4.2. *Eliminating Non-local Path Expressions*

It is possible to transform a formula containing non-local path expressions into a logically equivalent formula containing path equivalence classes, but no non-local paths. Then the simpler interpretation of FDL (with no global context) may be used to determine the structures which satisfy the formula.

The first step of the transformation uses the path expansion algorithm of Figure 7. This algorithm applies equivalences (6 and 9) of the calculus exhaustively to a formula, and all of its subformulas. Therefore, by the correctness of the formal equivalences, the result produced by the algorithm must be equivalent to the original formula.

The second step of the transformation converts a sequence of labels followed by a path into a path equivalence class, using the schema

(27) $$l_1 : \ldots : l_n : \langle p\rangle \Rightarrow [\![\langle l_1 \ldots l_n\rangle, \langle p\rangle]\!].$$

Note that path expansion does not require an expansion to full DNF, since disjunctions are not multiplied. While the DNF expansion of a formula may be exponentially larger than the original, the path expansion is at most quadratically larger.

THEOREM 1. Path expansion requires at most $O(n^2)$ space and time.

Proof. The size of the formula with paths expanded is at most $n \times p$, where n is the length of the original formula, and p is the length of the longest path. Since p is generally much less than n the size of the path expansion is usually not a very large quadratic.

```
Function PATH-EXPAND ($\phi$) Returns formula:
Select form of $\phi$:
  · $l : (\psi \vee \chi) \Rightarrow$ Return $l$ : PATH-EXPAND($\psi$) $\vee$ $l$ : PATH-EXPAND($\chi$);
  · $l : (\psi \wedge \chi) \Rightarrow$ Return $l$ : PATH-EXPAND($\psi$) $\wedge$ $l$ : PATH-EXPAND($\chi$);
  · $l : \psi \Rightarrow$ Return $l$ : PATH-EXPAND($\psi$);
  · $\psi \wedge \chi \Rightarrow$ Return PATH-EXPAND($\psi$) $\wedge$ PATH-EXPAND($\chi$);
  · $\psi \vee \chi \Rightarrow$ Return PATH-EXPAND($\psi$) $\vee$ PATH-EXPAND($\chi$);
  · Atomic (otherwise) $\Rightarrow$ Return $\phi$.
```

Fig. 7. An algorithm for path-expansion.

THEOREM 2. After path expansion all complete strings of labels in a formula denote trasitions from a common initial state of an automaton.

Proof. Observe that after path expansion no subformulas of the form $l:(\psi \vee \chi)$ or of the form $l:(\psi \wedge \chi)$ remain. Therefore any subformula containing labels must be a string of labels followed by an atomic formula. Let us generalize the definition of satisfiability for formulas having the form $l:\phi$ to those of the form $p:\phi$, where p may be any string of labels:

$$(\mathscr{A}, q) \models p : \phi \Leftrightarrow \delta(q, p) \text{ is defined and } (\mathscr{A}, \delta(q, p)) \models \phi.$$

Consider the case where ϕ is a non-local path expression denoted by $\langle w \rangle$:

$$(\mathscr{A}, \delta(q, p)) \models \langle w \rangle \Leftrightarrow \delta_{\mathscr{A}}(q_0, w) = \delta_{\mathscr{A}}(q, p).$$

When $q = q_0$, then $\delta(q_0, w) = \delta(q_0, p)$ and $(\mathscr{A}, q_0) \models [\![\langle w \rangle, \langle p \rangle]\!]$. After path expansion, the only subformulas which are interpreted with respect to a state different from the initial state of the entire formula are those denoted by ϕ in $l:\phi$. Thus, all strings of labels followed by a non-local path expression $\langle w \rangle$ must be interpreted by a pair containing the initial state of the automaton. In this case $q = q_0$, and

$$(\mathscr{A}, q_0) \models p : \langle w \rangle \Leftrightarrow (\mathscr{A}, q_0) \models [\![\langle p \rangle, \langle w \rangle]\!].$$

This ends the proof.

Therefore, by Theorem 2, the expressions containing non-local paths can be converted to the equivalence class notation, using the schema (27).

As an example of this transformation, consider the passive voice alternative of the description of Figure 2, shown here in Figure 8. This description is also represented by the first formula of Figure 9. The following formulas in Figure 9 are formed successively by

(1) applying path expansion,
(2) converting the attributes containing non-local path values to formulas representing equivalence classes of paths.

By following this procedure, the entire functional description of Figure 2 can be represented by the logical formula given in Figure 10.

$$\left[\begin{array}{ll} \text{voice} = \text{passive} & \\ \text{goal} = \langle\text{subj}\rangle & \\ \text{adjunct} = & \left[\begin{array}{l} \text{cat} = \text{pp} \\ \text{prep} = \text{by} \\ \text{obj} = \langle\text{actor}\rangle \\ \qquad = [\text{case} = \text{objective}] \end{array}\right] \end{array}\right]$$

Fig. 8. Functional description containing non-local values.

voice : passive
∧ goal : ⟨subj⟩
∧ adjunct : (cat : pp
∧ prep : by
∧ obj : ⟨actor⟩
∧ obj : case : objective)

path expansion
⇒

voice : passive
∧ goal : ⟨subj⟩
∧ adjunct : cat : pp
∧ adjunct : prep : by
∧ adjunct : obj : ⟨actor⟩
∧ adjunct : obj : case : objective

path equivalence
⇒

voice : passive
∧ ⟦⟨goal⟩, ⟨subj⟩⟧
∧ adjunct : cat : pp
∧ adjunct : prep : by
∧ ⟦⟨adjunct obj⟩, ⟨actor⟩⟧
∧ adjunct : obj : case : objective

Fig. 9. Conversion of non-local values to equivalence classes of paths.

cat : *s*
∧ subj : case : nominative
∧
((voice : active
∧ ⟦⟨actor⟩, ⟨subj⟩⟧)
∨
(voice : passive
∧ ⟦⟨goal⟩, ⟨subj⟩⟧
∧ adjunct : cat : pp
∧ adjunct : prep : by
∧ ⟦⟨adjunct obj⟩, ⟨actor⟩⟧
∧ adjunct : obj : case : objective))
∧
(mood : declarative
∨
mood : interrogative)

Fig. 10. Logical formula representing the description of Figure 2.

5. Mathematical results

In this section we present some of the mathematical justification for the claims made earlier. We begin with the notion of subsumption.

5.1. *Subsumption*

Let $\mathscr{A}$ and $\mathscr{B}$ be two automata. We say $\mathscr{A} \sqsubseteq \mathscr{B}$ ($\mathscr{A}$ subsumes $\mathscr{B}$; $\mathscr{B}$ extends $\mathscr{A}$) iff there is a *homomorphism* from $\mathscr{A}$ to $\mathscr{B}$; that is, a map $h: Q_{\mathscr{A}} \to Q_{\mathscr{B}}$ such that (for all existing transitions)

(1) $h(\delta_{\mathscr{A}}(q, l)) = \delta_{\mathscr{B}}(h(q), l)$;
(2) $h(c) = c$ for all $c \in A \cap F_{\mathscr{A}}$;
(3) $h(q_{0_{\mathscr{A}}}) = q_{0_{\mathscr{B}}}$.

This definition captures the notion of extension (see [2, page 5]) precisely. We intend that $\mathscr{A} \sqsubseteq \mathscr{B}$ if $\mathscr{B}$ has more information than $\mathscr{A}$. This may happen in two ways. The automaton $\mathscr{B}$ may have more transitions than $\mathscr{A}$, and here is where the different kinds of final state are important: $\mathscr{B}$ can extend $\mathscr{A}$ if $\mathscr{B}$ has transitions out of the nonatomic final states of $\mathscr{A}$. Also, if $\mathscr{B}$ identifies two paths which in $\mathscr{A}$ are distinct, then $\mathscr{B}$ has more information than $\mathscr{A}$, so $\mathscr{A} \sqsubseteq \mathscr{B}$ in this case too. It is not hard to show that if two automata subsume one another, then they are isomorphic, so that $\sqsubseteq$ is a partial order on the isomorphism classes of finite acyclic automata.

Next, we show that if $\mathscr{A}$ is a finite acyclic automaton, then we may write a formula in FDL describing it up to isomorphism.

DEFINITION. (Nerode canonical formula of an automaton). Let $\mathscr{A}$ be an automaton, and let $N(\mathscr{A})$ be its Nerode relation. Then we define

$$\phi(\mathscr{A}) = \left(\bigwedge_{c \in F} \left(\bigwedge_{\delta(q_0, x) = c} x : c \right) \right) \wedge E_1 \wedge \ldots \wedge E_n$$

where c is an atom, and the E_i are the Nerode classes of $N(\mathscr{A})$, determined by nonatomic states.

DEFINITION. We say $\phi(\mathscr{A}) \sqsubseteq \phi(\mathscr{B})$ iff for each E occurring in $\phi(\mathscr{A})$, and also for the sets E_c consisting of all the strings x such that $x : c$ occurs in the formula, there is a corresponding F occurring in $\phi(\mathscr{B})$ with $E \subseteq F$. (To say that this inclusion and its reverse holds is to say that the formulas are the same.)

THEOREM 3. The following are equivalent:

(1) $\mathscr{A} \sqsubseteq \mathscr{B}$;

(2) $\forall\phi(\mathscr{A} \vDash \phi \Rightarrow \mathscr{B} \vDash \phi)$;

(3) $\mathscr{B} \vDash \phi(\mathscr{A})$;

(4) $\phi(\mathscr{A}) \subseteq \phi(\mathscr{B})$.

Proof. (1) $\Rightarrow$ (2). We proceed by induction on the structure of ϕ. If this formula is an atom or NIL, the result is trivial. If ϕ is E, then there is a state $r \in Q_{\mathscr{A}}$ such that $\delta_{\mathscr{A}}(q_0, x) = r$ for all $x \in E$. Then $\delta_{\mathscr{B}}(q_0, h(x)) = h(r)$, proving that $\mathscr{B} \vDash E$. The proof for the propositional connectives $\vee$ and $\wedge$ is standard, so consider the case $\phi = l : \psi$. If $\mathscr{A} \vDash l : \psi$, then $\mathscr{A}/l$ is defined and satisfies ψ. But by (1), $\mathscr{A} \sqsubseteq \mathscr{B}$. It easily follows that $\mathscr{A}/l \sqsubseteq \mathscr{B}/l$, and by inductive hypothesis $\mathscr{B}/l \vDash \psi$. Thus $\mathscr{B} \vDash l : \psi$.

(2) $\Rightarrow$ (3). Trivial, since $\mathscr{A} \vDash \phi(\mathscr{A})$.

(3) $\Rightarrow$ (4). Let $x : c$ be a conjunct in $\phi(\mathscr{A})$. Then $\delta_{\mathscr{A}}(q_0, x) = c$. Since $\mathscr{B} \vDash \phi(\mathscr{A})$, $\mathscr{B}$ has an x-path to c, and so $x : c$ is a conjunct in $\phi(\mathscr{B})$. In a similar way, it follows that each E_i class occurring in $\phi(\mathscr{A})$ is contained in a class of $\phi(\mathscr{B})$. Thus $\phi(\mathscr{A}) \subseteq \phi(\mathscr{B})$.

(4) $\Rightarrow$ (1). We must define $h : Q_{\mathscr{A}} \to Q_{\mathscr{B}}$. Let $r \in Q_{\mathscr{A}}$ and choose x such that $\delta_{\mathscr{A}}(q_0, x) = r$. Then $x \in E$ for some Nerode class E of $\mathscr{A}$. By (4), this class is contained in a Nerode class of $\mathscr{B}$, represented by some state s of $\mathscr{B}$. Let $h(r) = s$. Standard arguments show that h is well-defined and a homomorphism. This completes the proof.

We can now use this result to show semantically that if two automata are unifiable (i.e., have an upper bound in the subsumption ordering,) then they have a most general unifier (i.e., a least upper bound in the ordering.)

DEFINITION. The automaton $\mathscr{A} \times \mathscr{B}$ is the product automaton with state set $Q_{\mathscr{A}} \times Q_{\mathscr{B}}$, and with transitions defined when both components can do so on a particular input. We also need to identify a pair of states (c, d), where c and d are distinct atoms, with a nonatomic final state q, and to identify (c, q) and (q, d) with q as well. The pair (c, c) is identified with c. $\mathscr{A} \times \mathscr{B}$ can be thought of in linguistic terms as the generalization of $\mathscr{A}$ and $\mathscr{B}$. It is in fact the greatest lower bound of $\mathscr{A}$ and $\mathscr{B}$ in the subsumption ordering.

DEFINITION. A *conjunctive formula* contains no uses of the $\vee$ connective.

THEOREM 4. Let ϕ be a conjunctive formula. If $\mathscr{A} \vDash \phi$ and $\mathscr{B} \vDash \phi$, so does $\mathscr{A} \times \mathscr{B}$.

Proof. Again use induction on ϕ. The base cases and the inductive step for the connective $\wedge$ are all easy. Consider the case $\phi = l : \psi$. If $\mathcal{A}$ and $\mathcal{B}$ satisfy ϕ, we must show that $(\mathcal{A} \times \mathcal{B})/l$ exists and satisfies ψ. But it is easy to show that $(\mathcal{A} \times \mathcal{B})/l$ is equal to $\mathcal{A}/l \times \mathcal{B}/l$. The result follows by inductive hypothesis.

THEOREM 5 (existence of most general unifier). If ϕ is conjunctive and satisfiable, then there is a subsumption-minimum automaton $\mathcal{A}$ such that $\mathcal{A} \models \phi$.

Proof. Use condition (4) in the characterization of subsumption to show that subsumption restricted to the set of automata satisfying ϕ is well-founded (in fact, that there are only a finite number of automata which subsume a given one), and the previous theorem to show that two minimal elements are the same. This proves the theorem, and the fact that it also shows the existence of a most general unifier can be seen as follows. Assume that $\mathcal{A}$ and $\mathcal{B}$ are two automata which have a common upper bound in the subsumption ordering. Then their most general unifier is that automaton $\mathcal{C}$ which is the least one satisfying $\phi(\mathcal{A}) \wedge \phi(\mathcal{B})$.

5.2. *Disjunctive Normal Form*

A formula is in *disjunctive normal form* if and only if it has the form $\phi_1 \vee \ldots \vee \phi_n$, where each disjunct ϕ_i is either

(1) a, where $a \in A$;

(2) $\psi_1 \wedge \ldots \wedge \psi_m$, where each conjunct ψ_i is either

  (a) $l_1 : \ldots : l_k : a$, where $a \in A$, and no path (i.e., sequence of labels) occurs more than once;

  (b) $[\![\langle p_1\rangle, \ldots, \langle p_k\rangle]\!]$, where each $p_i \in L^*$, and each set denotes an equivalence class of paths, and all such sets, within a disjunct, are disjoint.

Another way to express the same idea, and an equivalent definition of disjunctive NF, is to use the notion of the canonical formula associated with an automaton. It is easy to see that the individual disjuncts in the normal form just defined are just the $\phi(\mathcal{A}_i)$ defined in the previous section. Thus, one way of stating a normal form result is as follows.

THEOREM 6 (Nerode normal form). Let ϕ be satisfiable; then there is a finite set of automata $\mathcal{A}_1, \ldots, \mathcal{A}_n$, depending only on the logical equivalence class of ϕ, such that the $\mathcal{A}_i$ are subsumption-incomparable, and such that

$$\phi \Leftrightarrow \phi(\mathcal{A}_1) \vee \ldots \vee \phi(\mathcal{A}_n).$$

One way to prove this result is as follows. Given a formula, apply first the path expansion laws, and then the distributive laws until the formula is in a disjunctive form; that is, a disjunction of conjunctive formulas. Then use the results of the previous section to find, for each conjunctive formula, the minimal automaton satisfying it. This will be a finite set, and it can be shown that this set of automata depends only on the original formula. This gives a semantic proof of the NF result, but a more algorithmic approach is preferable for the completeness result. The formal equivalences given in Figure 5 allows us to transform any satisfiable formula into its disjunctive normal form, or to *TOP* if it is not satisfiable. Now our calculus is seen to be complete, because the logical equivalence of any two formulae can be shown just by equationally transforming the two formulae into their Nerode Normal Form (NNF), which form is unique up to associativity and commutativity.

THEOREM 7. There is an algorithm for computing the NNF of any formula within $2^{0(n)}$ time; further, this algorithm derives the NNF using only the equational laws.

Proof. The algorithm is given in Figure 11, and requires exponential time, where the exponent depends on the number of disjunctions in the formula. The applications of the distributive laws are responsible for the exponential running time. This algorithm will not be used in practice, but it is given here to illustrate the decidability of equivalence in the calculus.

We end the section with our NP-completeness result.

THEOREM 8. The satisfiability problem for FDL is NP-complete.

Proof. We reduce the classical satisfiability problem to that for our formulas. This reduction is not trivial, because our formulas do not have negations. However, the reduction is still quite easy, and makes use of the fact that paths can have distinct atomic values. We therefore reduce the satisfiability of CNF formulas in propositional calculus to the satisfiability problem in our logic. Let P be a formula of propositional calculus in CNF. For each literal x_i in P, create a label l_i in L. For each literal $\neg x_i$, create a label l_i'. (If the set of labels is finite, this will have to be done using paths instead of labels.) Let a and b be two distinct atoms. Transform P as follows. Replace each x_i by $l_i : a$, and each $\neg x_i$ by $l_i' : a$. (Think of 'a' standing for 'true'.) Now, to the formula just obtained, conjoin the following:

$$\bigwedge_i ((l_i : a \wedge l_i' : b) \vee (l_i' : a \wedge l_i : b)).$$

```
Function NNF($\phi$) Returns formula:
  where $\phi$ is a formula.
1. Apply laws (6) and (9) from right to left, moving all labels to the
   inside.
2. Apply the distributive law (16) from left to right, until no more appli-
   cations are possible.
3. Eliminate redundancies and inconsistencies within disjuncts.

  For each disjunct ($\psi$) do:
    Apply (24) to all conjuncts of the form $w : E$.
    Apply (23) to all conjuncts which are E formulas.

    Using the commutative (10) and associative (12) laws, bring together
    pairs of conjuncts as necessary for the following steps:
    If some conjunct is an atom,
      Then apply (14) to eliminate redundant conjuncts,
        or apply (4) and (5) to detect inconsistent information.
    Apply (21) to close path equivalence classes under right invariance.
    Apply (20) to make the path equivalence classes disjoint.
    Apply (22) to add all atomic values required by path equivalence.
    Apply (6) for each pair of conjuncts having a common prefix of labels:
      Apply (2–5) and (14) to eliminate redundant
        and inconsistent conjuncts.

    Apply (1) to propagate any failure to the top level of the disjunct.
    If any conjunct is TOP, then use (2) to propagate failure.
    Apply (3) to eliminate any NIL conjuncts.

    Now, $\psi$ must be one of the following types:
      NIL : then Return (NIL), by law (7);
      TOP : then eliminate this disjunct, by law (8);
      otherwise: leave $\psi$ as a disjunct in $\phi$.

    Return the modified value of $\phi$.
```

Fig. 11. Algorithm to convert a formula to Nerode Normal Form.

This clause forces l_i and l_i' to have differing values. It follows that the constructed formula is satisfiable iff P is.

The satisfiability problem is in NP, because given a formula, we can guess which disjuncts are to be satisfied, obtaining a conjunctive formula after discarding unwanted disjuncts. The conjunctive formula can be put into NF and checked for consistency in polynomial time. This completes the proof.

6. Conclusion

We have seen how the techniques of logic have helped us to understand the complicated grammatical notion of unification, especially in situations involving functional descriptions which have shared functional values, nonlocal values, and disjunction. This approach has helped us to design reasonably efficient algorithms dealing with these descriptions, and it is mathematically simple. There is much more to the approach than we can describe here, but we hope that this article clarifies some of the basic issues in the syntactical metatheory of unification grammar.

References

[1] Ait-Kaci, H.: 1984, *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially Ordered Type Structures*, PhD thesis, University of Pennsylvania.
[2] Gazdar, G., G. K. Pullum, R. Carpenter, E. Klein, T. Hukari, and R. Levine: 1986, 'Category Structures', Technical Report SRC-86-01(2), Syntax Research Center, University of California, Santa Cruz.
[3] Gazdar, G., E. Klein, G. K. Pullum, and I. A. Sag: 1985, *Generalized Phrase Structure Grammar*, Blackwell Publishing, Oxford, England, and Harvard University Press, Cambridge, Massachusetts.
[4] G. R. Kress (ed.): 1976, *Halliday: System and Function in Language*, Oxford University Press, London, England.
[5] Kaplan, R. and J. Bresnan: 1983, 'Lexical Functional Grammar: A Formal System for Grammatical Representation', in J. Bresnan (ed.), *The Mental Representation of Grammatical Relations*, MIT Press, Cambridge, Massachusetts.
[6] Karttunen, L.: 1984, 'Features and Values', in *Proceedings of the Tenth International Conference on Computational Linguistics: COLING* 84, Stanford University, Stanford, California.
[7] Kasper, R., and W. Rounds: 1986, 'A Logical Semantics for Feature Structures', in *Proceedings of the 24th Annual Meeting of the ACL*, New York, N.Y.
[8] Kasper, R.: 1987, *Feature Structures: A Logical Theory with Applications to Natural Language Analysis*, Ph.D. dissertation, University of Michigan.
[9] Kay, M.: 1979, 'Functional Grammar', in *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley Linguistics Society, Berkeley, California.
[10] Kay, M.: 1985, 'Parsing in Functional Unification Grammar', in D. Dowty, L. Karttunen, and A. Zwicky (eds.), *Natural Language Parsing*, Cambridge University Press, Cambridge, England.
[11] Pereira, F. C. N. and S. M. Shieber: 1985, 'The Semantics of Grammar Formalisms Seen as Computer Languages', in *Proceedings of the 23rd Annual Meeting of the ACL*, Chicago, Ill.
[12] Rounds, W. C. and R. Kasper: 1986, 'A Complete Logical Calculus for Record Structures Representing Linguistic Information', *Proc. IEEE Symposium on Logic in Computer Science*.

[13] Shieber, S. M.: 1984, 'The Design of a Computer Language for Linguistic Information', in *Proceedings of the Tenth International Conference on Computational Linguistics: COLING* 84, Stanford University, Stanford, California.
[14] Shieber, S. M.: 1986, *An Introduction to Unification-based Approaches to Grammar*, University of Chicago Press, Chicago, CSLI Lecture Notes Series.

Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, Michigan 48109
U.S.A.