A LOGICAL SEMANTICS FOR FEATURE STRUCTURES

Robert T. Kasper and William C. Rounds
Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, Michigan 48109

Abstract

Unification-based grammar formalisms use structures containing sets of features to describe linguistic objects. Although computational algorithms for unification of feature structures have been worked out in experimental research, these algorithms become quite complicated, and a more precise description of feature structures is desirable. 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. These graphs are, in fact, 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 an interpretation for values which are specified by disjunctions and 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. This logical model yields a calculus of equivalences, which can be used to simplify formulas.

Unification is attractive, because of its generality, but it is often computationally inefficient. Our model allows a careful examination of the computational complexity of unification. We have shown that the consistency problem for formulas with disjunctive values is NP-complete. To deal with this complexity, we describe how disjunctive values can be specified in a way which delays expansion to disjunctive normal form.

1 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) [2], Lexical Functional Grammar (LFG) [4], and Systemic Grammar [3]. They also include grammar formalisms which have been developed as computational tools, such as Functional Unification Grammar (FUG) [7], and PATR-II [14]. 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 [8]. Our contribution is to give a more complete algebraic account of the logical properties of feature structures, which can be used explicitly for computational manipulation and mathematical analysis. Proofs of the mathematical soundness and completeness of this logical treatment, along with its relation to similar logics, can be found in [12].

2 Disjunction and Non-Local Values

Karttunen [5] has shown that disjunction and negation are desirable extensions to PATR-II which are motivated by a wide range of linguistic

$$\text{die} = \left[\begin{array}{l} case : \{nom\ acc\} \\ agreement : \left\{ \left[\begin{array}{l} gender : fem \\ number : sing \end{array} \right] \right. \\ \qquad\qquad \left. \left[number : pl \right] \right\} \end{array} \right]$$

Figure 1: A Feature Structure containing Value Disjunction.

phenomena. He discusses specifying attributes by disjunctive values, as shown in Figure 1. A *value disjunction* specifies alternative values of a single attribute. These alternative values may be either atomic or complex. 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 negative and disjunctive values, and explains some of the complications that result from introducing value disjunction. When two values, A and B, are to be unified, 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. 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 disjunction contains a value which is specified by a non-local path, a situation which occurs frequently in Functional Unification Grammar. In Figure 2 the *obj* attribute in the description of the *adjunct* attribute is given the value $< actor >$, which means that the *obj* attribute is to be unified with the value found at the end of the path labeled by $< actor >$ in the outermost enclosing structure. This unification with a non-local value can be performed only when the alternative which contains it is the only alternative remaining in the disjunction. Otherwise, the $case = objective$ attribute might be added to the value of $< actor >$ prematurely, when the alternative containing *adjunct* is not used. 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, provide 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 5.4.

3 Logical Formulas for Feature Structures

The feature structure of Figure 1 can also be represented by a type of logical formula:

$$\begin{aligned} \text{die} \quad = \quad & case : (nom \vee acc) \\ & \wedge\ agreement : (\\ & (gender : fem \ \wedge \ number : sing) \\ & \vee\ number : pl) \end{aligned}$$

This type of formula differs from standard propositional logic in that a theoretically unlimited set of atomic values is used in place of boolean values. The labels of attributes bear a superficial resemblance to modal operators. Note that no information is added or subtracted by rewriting the feature matrix of Figure 1 as a logical formula. These two forms may be regarded as notational variants for expressing the same facts. While feature matrices seem to be a more appealing and natural notation for displaying linguistic descriptions, logical formulas provide a precise interpretation which can be useful for computational and mathematical purposes.

Given this intuitive introduction we proceed to a more complete definition of this logic.

4 A Logical Semantics

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 useful to use a representation language which is disctinct 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.

4.1 Domain of Feature Structures

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:

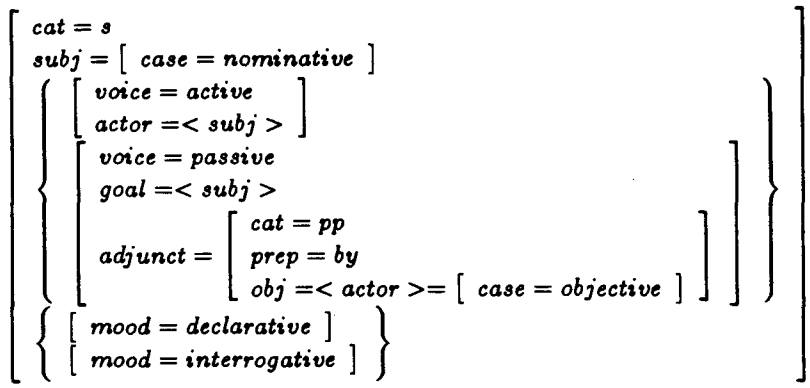

Figure 2: Disjunctive specification containing non-local values, using the notation of FUG.

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.

A dag may also be regarded as a transition graph for a partially specified deterministic finite automaton (DFA). This automaton recognises 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

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

where 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 $\mathcal{A}$ be connected, acyclic, and have no transitions from any final states.

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 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 $\mathcal{A}$ is an automaton, we let $P(\mathcal{A})$ be the set of all paths of $\mathcal{A}$, namely the set $\{x \in L^* : \delta(q_0, x)$ is defined $\}$. The *Nerode relation* $N(\mathcal{A})$ is the equivalence relation defined on paths of $P(\mathcal{A})$ by letting

$$x \ N(\mathcal{A}) \ y \iff \delta(q_0, x) = \delta(q_0, y).$$

4.2 Domain of Descriptions: Logical Formulas

We now define the domain FML 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 FML. The formulas NIL and TOP are intended to convey "no information" and "inconsistent information" respectively. Thus, NIL corresponds to a unification variable, and TOP corresponds to unification failure. A formula $l : \phi$ would indicate that a value has attribute l, which is itself a value satisfying the condition ϕ.

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

Figure 3: The domain, FML, of logical formulas.

Conjunction and disjunction will have their ordinary logical meaning 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.

One 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

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

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.

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 $E = [\![< x >, < y >]\!]$, we will sometimes write E as $< x >=< y >$. 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.

4.3 Interpretation of Formulas

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

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

where $\mathcal{A}/l$ is defined by a subgraph of the automaton $\mathcal{A}$ with start state $\delta(q_0, l)$, that is
if $\mathcal{A} = (Q, L, \delta, q_0, F)$,
then $\mathcal{A}/l = (Q', L, \delta, \delta(q_0, l), F')$;
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 occurences of disjunction) the set of automata satisfying the formula has a unique minimal element, with respect to subsumption.[1] For disjunctive formulas there may be several minimal elements, but always a finite number.

4.4 Calculus of Formulas

It is possible to write many formulas which have an identical interpretation. For example, the formulas given in the equation below are satisfied by the same set of automata.

$$case : (gen \vee acc \vee dat) \wedge case : acc \;=\; case : acc$$

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. Thus, we are led to state the laws of equivalence shown in Figure 4. Note that equivalence (26) is added only to make descriptions of cyclic structures unsatisfiable.

[1] A subsumption order can be defined for the domain of automata, just as it is defined for dags by Shieber [15]. A formal definition of subsumption for this domain appears in [12].

Failure:			
$l : TOP$	$=$	TOP	(1)
Conjunction (unification):			
$\phi \wedge TOP$	$=$	TOP	(2)
$\phi \wedge NIL$	$=$	ϕ	(3)
$a \wedge b$	$=$	TOP, $\forall a, b \in A$ and $a \neq b$	(4)
$a \wedge l : \phi$	$=$	TOP	(5)
$l : \phi \wedge l : \psi$	$=$	$l : (\phi \wedge \psi)$	(6)
Disjunction:			
$\phi \vee NIL$	$=$	NIL	(7)
$\phi \vee TOP$	$=$	ϕ	(8)
$l : \phi \vee l : \psi$	$=$	$l : (\phi \vee \psi)$	(9)
Commutative:			
$\phi \wedge \psi$	$=$	$\psi \wedge \phi$	(10)
$\phi \vee \psi$	$=$	$\psi \vee \phi$	(11)
Associative:			
$(\phi \wedge \psi) \wedge \chi$	$=$	$\phi \wedge (\psi \wedge \chi)$	(12)
$(\phi \vee \psi) \vee \chi$	$=$	$\phi \vee (\psi \vee \chi)$	(13)
Idempotent:			
$\phi \wedge \phi$	$=$	ϕ	(14)
$\phi \vee \phi$	$=$	ϕ	(15)
Distributive:			
$(\phi \vee \psi) \wedge \chi$	$=$	$(\phi \wedge \chi) \vee (\psi \wedge \chi)$	(16)
$(\phi \wedge \psi) \vee \chi$	$=$	$(\phi \vee \chi) \wedge (\psi \vee \chi)$	(17)
Absorption:			
$(\phi \wedge \psi) \vee \phi$	$=$	ϕ	(18)
$(\phi \vee \psi) \wedge \phi$	$=$	ϕ	(19)
Path Equivalence:			
$E_1 \wedge E_2$	$=$	E_2 whenever $E_1 \subseteq E_2$	(20)
$E_1 \wedge E_2$	$=$	$E_1 \wedge (E_2 \cup \{xy \mid z \in E_1\})$ for any y such that $\exists x : x \in E_1$ and $xy \in E_2$	(21)
$E \wedge x : c$	$=$	$E \wedge (\bigwedge_{y \in E} y : c)$ where $x \in E$	(22)
E	$=$	$E \wedge \{x\}$ if x is a prefix of a string in E	(23)
$l : E$	$=$	$\{lw \mid w \in E\}$	(24)
$\{\epsilon\}$	$=$	NIL	(25)
E	$=$	TOP for any E such that there are strings x, $xy \in E$ and $y \neq \epsilon$	(26)

Figure 4: Laws of Equivalence for Formulas.

5 Complexity of Disjunctive Descriptions

To date, the primary benefit of using logical formulas to describe feature structures has been the clarification of several problems that arise with disjunctive descriptions.

5.1 NP-completeness of consistency problem for formulas

One consequence of describing feature structures by logical formulas is that it is now relatively easy to analyze the computational complexity of various problems involving feature structures. It turns out that the satisfiability problem for CNF formulas of propositional logic can be reduced to the consistency (or satisfiability) problem for formulas in FML. Thus, the consistency problem for formulas in FML is NP-complete. It follows that any unification algorithm for FML formulas will have non-polynomial worst-case complexity (provided P $\neq$ NP!), since a correct unification algorithm must check for consistency.

Note that disjunction is the source of this complexity. If disjunction is eliminated from the domain of formulas, then the consistency problem is in P. Thus systems, such as the original PATR-II, which do not use disjunction in their descriptions of feature structures, do not have to contend with this source of NP-completeness.

5.2 Disjunctive Normal Form

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

1. $a \in A$
2. $\psi_1 \wedge \ldots \wedge \psi_m$, where each ψ_i is either
   (a) $l_1 : \ldots : l_k : a$, where $a \in A$, and no path occurs more than once
   (b) $[< p_1 >, \ldots, < p_k >]$, where each $p_i \in L^*$, and each set denotes an equivalence class of paths, and all such sets disjoint.

The formal equivalences given in Figure 4 allow us to transform any satisfiable formula into its disjunctive normal form, or to *TOP* if it is not satisfiable. The algorithm for finding a normal form requires exponential time, where the exponent depends on the number of disjunctions in the formula (in the worst case).

5.3 Avoiding expansion to DNF

Most of the systems which are currently used to implement unification-based grammars depend on an expansion to disjunctive normal form in order to compute with disjunctive descriptions.[2] Such systems are exemplified by Definite Clause Grammar [10], which eliminates disjunctive terms by multiplying rules which contain them into alternative clauses. Kay's parsing procedure for Functional Unification Grammar [8] also requires expanding functional descriptions to DNF before they are used by the parser. This expansion may not create much of a problem for grammars containing a small number of disjunctions, but if the grammar contains 100 disjunctions, the expansion is clearly not feasible, due to the exponential size of the DNF.

Ait-Kaci [1] has pointed out that the expansion to DNF is not always necessary, in work with type structures which are very similar to the feature structures that we have described here. Although the NP-completeness result cited above indicates that any unification algorithm for disjunctive formulas will have exponential complexity in the worst case, it is possible to develop algorithms which have an average complexity that is less prohibitive. Since the exponent of the complexity function depends on the number of disjunctions in a formula, one obvious way to improve the unification algorithm is to reduce the number of disjunctions in the formula *before expansion to DNF*. Fortunately the unification of two descriptions frequently results in a reduction of the number of alternatives that remain consistent. Although the fully expanded formula may be required as a final result, it is expedient to delay the expansion whenever possible, until after any desired unifications are performed.

The algebraic laws given in Figure 4 provide a sound basis for simplifying formulas containing disjunctive values without expanding to DNF. Our calculus differs from the calculus of Ait-Kaci by providing a uniform set of equivalences for formulas, including those that contain disjunction. These equivalences make it possible to

[2] One exception is Karttunen's implementation, which was described in Section 2, but it handles only value disjunctions, and does not handle non-local path values embedded within disjunctions.

eliminate inconsistent terms before expanding to DNF. Each term thus eliminated may reduce, by as much as half, the size of the expanded formula.

5.4 Representing Non-local Paths

The logic contains no direct representation for non-local paths of the type described in Section 2. The reason is that these cannot be interpreted without reference to the global context of the formula in which they occur. 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. Stated formally, the desired interpretation of $l :< p >$ is

$$\mathcal{A} \models l :< p > \text{ in the context of } \phi \iff$$
$$\exists \mathcal{B} \models \phi \text{ and } \exists w \in L^* :$$
$$\mathcal{B}/w = \mathcal{A} \text{ and } \delta(q_{0_\mathcal{A}}, l) = \delta(q_{0_\mathcal{B}}, p).$$

This interpretation does not allow a direct comparison of the non-local path value with other values in the formula. It remains an unknown quantity unless the environment is known.

Instead of representing non-local paths directly in the logic, we propose that they can be used within a formula as a shorthand, but that all paths in the formula must be expanded before any other processing of the formula. This *path expansion* is carried out according to the equivalences 9 and 6.

After path expansion all strings of labels in a formula denote transitions from a common origin, so the expressions containing non-local paths can be converted to the equivalence class notation, using the schema

$$l_1 : \ldots : l_n :< p > = [< l_1, \ldots, l_n >, < p >].$$

Consider the passive voice alternative of the description of Figure 2, shown here in Figure 5. This description is also represented by the first formula of Figure 6. The formulas to the right in Figure 6 are formed 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 7.

$$\begin{bmatrix} voice = passive \\ goal =< subj > \\ adjunct = \begin{bmatrix} cat = pp \\ prep = by \\ obj =< actor > \\ \quad = [\ case = objective\ ] \end{bmatrix} \end{bmatrix}$$

Figure 5: Functional Description containing non-local values.

$$voice : passive \wedge goal :< subj > \wedge adjunct : (cat : pp \wedge prep : by \wedge obj :< actor > \wedge obj : case : objective)$$

path expansion $\Longrightarrow$

$$voice : passive \wedge goal :< subj > \wedge adjunct : cat : pp \wedge adjunct : prep : by \wedge adjunct : obj :< actor > \wedge adjunct : obj : case : objective$$

path equivalence $\Longrightarrow$

$$voice : passive \wedge [< goal >, < subj >] \wedge adjunct : cat : pp \wedge adjunct : prep : by \wedge [< adjunct\ \ obj >, < actor >] \wedge adjunct : obj : case : objective$$

Figure 6: Conversion of non-local values to equivalence classes of paths.

$$
\begin{array}{l}
cat : s \\
\wedge\ subj : case : nominative \\
\wedge \\
((voice : active \\
\wedge\ [< actor >, < subj >]) \\
\vee \\
(voice : passive \\
\wedge\ [< goal >, < subj >] \\
\wedge\ adjunct : cat : pp \\
\wedge\ adjunct : prep : by \\
\wedge\ [< adjunct\ \ obj >, < actor >] \\
\wedge\ adjunct : obj : case : objective)) \\
\wedge \\
(mood : declarative \\
\vee \\
mood : interrogative)
\end{array}
$$

Figure 7: Logical formula representing the description of Figure 2.

It is now possible to unify the description of Figure 7 (call this X in the following discussion) with another description, making use of the equivalence classes to simplify the result. Consider unifying X with the description

$$Y = actor : case : nominative.$$

The commutative law (10) makes it possible to unify Y with any of the conjuncts of X. If we unify Y with the disjunction which contains the *voice* attributes, we can use the distributive law (16) to unify Y with both disjuncts. When Y is unified with the term containing

$$[< adjunct\ obj >, < actor >],$$

the equivalence (22) specifies that we can add the term

$$adjunct : obj : case : nominative.$$

This term is incompatible with the term

$$adjunct : obj : case : objective,$$

and by applying the equivalences (6, 4, 1, and 2) we can transform the entire disjunct to TOP. Equivalence (8) specifies that this disjunction can be eliminated. Thus, we are able to use the path equivalences during unification to reduce the number of disjunctions in a formula without expanding to DNF.

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. 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.

5.5 Value Disjunction and General Disjunction

The path expansion procedure illustrated in Figure 6 can also be used to transform formulas containing value disjucntion into formulas containing general disjunction. For the reasons given above, value disjunctions which contain non-local path expressions must be converted into general disjunctions for further simplification.

While it is possible to convert value disjunctions into general disjunctions, it is not always possible to convert general disjunctions into value disjunctions. For example, the first disjunction in the formula of Figure 7 cannot be converted into a value disjunction. The left side of equivalence (9) requires both disjuncts to begin with a common label prefix. The terms of these two disjuncts contain several different prefixes (*voice*, *actor*, *subj*, *goal*, and *adjunct*), so they cannot be combined into a common value.

Before the equivalences of section 4 were formulated, the first author attempted to implement a facility to represent disjunctive feature structures with non-local paths using only value disjunction. It seemed that the unification algorithm would be simpler if it had to deal with disjunctions only in the context of attribute values, rather than in more general contexts. While it was possible to write down grammatical definitions using only value disjunction, it was very difficult to achieve a correct unification algorithm, because each non-local path was much like an unknown variable. The logical calculus presented here clearly demonstrates that a representation of general disjunction provides a more direct method to determine the values for non-local paths.

6 Implementation

The calculus described here is currently being implemented as a program which selectively applies the equivalences of Figure 4 to simplify formulas. A strategy (or algorithm) for simplifying formulas corresponds to choosing a particular order in which to apply the equivalences whenever more than one equivalence matches the form of the formula. The program will make it possible to test and evaluate different strategies, with the correctness of any such strategy following directly from the correctness of the calculus. While this program is primarily of theoretical interest, it might yield useful improvements to current methods for processing feature structures.

The original motivation for developing this treatment of feature structures came from work on an experimental parser based on Nigel [9], a large systemic grammar of English. The parser is being developed at the USC/Information Sciences Institute by extending the PATR-II system of SRI International. The systemic grammar has been translated into the notation of Functional Unification Grammar, as described in [6]. Because this grammar contains a large number (several hundred) of disjunctions, it has been necessary to extend the unification procedure so that it handles disjunctive values containing non-local paths without expansion to DNF. We now think that this implementation of a relatively large grammar can be made more tractable by applying some of the transformations to feature descriptions which have been suggested by the logical calculus.

7 Conclusion

We have given a precise logical interpretation for feature structures and their descriptions which are used in unification-based grammar formalisms. This logic can be used to guide and improve implementations of these grammars, and the processors which use them. It has allowed a closer examination of several sources of complexity that are present in these grammars, particularly when they make use of disjunctive descriptions. We have found a set logical equivalences helpful in suggesting ways of coping with this complexity.

It should be possible to augment this logic to include characterizations of negation and implication, which we are now developing. It may also be worthwhile to integrate the logic of feature structures with other grammatical formalisms based on logic, such as DCG [10] and LFP [13].

References

[1] Ait-Kaci, H. *A New Model of Computation Based on a Calculus of Type Subsumption.* PhD thesis, University of Pennsylvania, 1984.

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

[3] G.R. Kress, editor. *Halliday: System and Function in Language.* Oxford University Press, London, England, 1976.

[4] Kaplan, R. and J. Bresnan. Lexical Functional Grammar: A Formal System for Grammatical Representation. In J. Bresnan, editor, *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts, 1983.

[5] Karttunen, L. Features and Values. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, July 2-7, 1984.

[6] Kasper, R. Systemic Grammar and Functional Unification Grammar. In J. Benson and W. Greaves, editors, *Proceedings of the 12th International Systemics Workshop*, Norwood, New Jersey: Ablex (forthcoming).

[7] Kay, M. Functional Grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, Berkeley Linguistics Society, Berkeley, California, February 17-19, 1979.

[8] Kay, M. Parsing in Functional Unification Grammar. In D. Dowty, L. Kartunnen, and A. Zwicky, editors, *Natural Language Parsing.* Cambridge University Press, Cambridge, England, 1985.

[9] Mann, W.C. and C. Matthiessen. Nigel: A Systemic Grammar for Text Generation. USC / Information Sciences Institute, RR-83-105. Also appears in R. Benson and J. Greaves, editors, *Systemic Perspectives on Discourse: Selected Papers Papers from the Ninth International Systemics Workshop*, Ablex, London, England, 1985.

[10] Pereira, F. C. N. and D. H. D. Warren. Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13:231-278, 1980.

[11] Pereira, F. C. N. and S. M. Shieber. The semantics of grammar formalisms seen as computer languages. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, July 2-7, 1984.

[12] Rounds, W. C. and R. Kasper. A Complete Logical Calculus for Record Strucutres Representing Linguistic Information. Submitted to the *Symposium on Logic in Computer Science*, to be held June 16-18, 1986.

[13] Rounds, W. C. LFP: A Logic for Linguistic Descriptions and an Analysis of its Complexity. Submitted to *Computational Linguistics*.

[14] Shieber, S. M. The design of a computer language for linguistic information. In *Proceedings of the Tenth International Conference on Computational Linguistics*, Stanford University, Stanford, California, July 2-7, 1984.

[15] Shieber, S. M. *An Introduction to Unification-based Approaches to Grammar.* Chicago: University of Chicago Press, CSLI Lecture Notes Series (forthcoming).