HORN EXTENDED FEATURE STRUCTURES: FAST UNIFICATION WITH NEGATION AND LIMITED DISJUNCTION[†]

Stephen J. Hegner
Department of Computer Science and Electrical Engineering
Votey Building
University of Vermont
Burlington, VT 05405 USA
telephone: (802)656-3330
internet: hegner@uvm.edu
uucp: ..uunet!uvm-gen!hegner

ABSTRACT

The notion of a *Horn extended feature structure* (*HoXF*) is introduced, which is a feature structure constrained so that its only allowable extensions are those satisfying some set of Horn clauses in feature-term logic. HoXF's greatly generalize ordinary feature structures in admitting explicit representation of negative and implicational constraints. In contradistinction to the general case in which arbitrary logical constraints are allowed (for which the best known algorithms are exponential), there is a highly tractable algorithm for the unification of HoXF's.

1. PRELIMINARY CONCEPTS

1.1 Unification-based grammar formalisms

Unification-based grammar formalisms constitute a cornerstone of many of the most important approaches to natural-language understanding (Shieber, 1986), (Colban, 1988), (Fenstad *et al.*, 1989). The basic idea is that the parser generates a number of partial representations of the total parse, which are subsequently checked for consistency and combined by a second process known as a *unifier*. A common form of representation for the partial representations is that of *feature structures*, which are record-like data structures which are allowed to grow in three distinct ways: by adding missing values, by adding attributes, and by *coalescing* existing attributes (forcing them to be the same). The last operation may lead to cyclic structures, which we do *not* exclude. If the feature structure S_2 is an extension of S_1 (*i.e.*, S_1 grows into S_2 by application of some sequence of the above rules), we write $S_1 \sqsubseteq S_2$ and say that S_1 *subsumes* S_2. Intuitively, if $S_1 \sqsubseteq S_2$, S_2 contains more information than does S_1. It is easy to show that $\sqsubseteq$ is a partial order on the class of all feature structures.

Each feature structure represents partial information generated during the parse. To obtain the total picture, these partial components must be combined into one consistent piece of knowledge. The formal process of *unification* is precisely this operation of combination. The *most general unifier* (*mgu*) $S_1 \sqcup S_2$ of feature structures S_1 and S_2 is the least feature structure (under $\sqsubseteq$) which is larger than both S_1 and S_2. Such an mgu exists if and only if S_1 and S_2 are *consistent*; that is, if and only if they subsume a common feature structure.

1.2 Unification algorithms and this paper

While the idea of a most general unifier is a pleasing theoretical notion, its real utility rest with the fact that there are efficient algorithms for its computation. The fastest known algorithm, identified by Aït-Kaci (1984), runs in time which is, for all practical purposes, linear in the size of the input (*i.e.*, the combined sizes of the structures to be unified). In proposing any extension to the basic framework, a primary consideration must be the complexity of the ensuing unification algorithm. The principal contribution of the research summarized here is to provide an extension of ordinary feature structures, admitting negation and limited disjunction, while at the same time continuing to admit a *provably* efficient unification algorithm.

Due to space limitations, we must omit substantial background material from this paper. Specifically, we assume that the reader is familiar with the notation and definitions surrounding feature structures (Shieber, 1986; Fenstad *et al.*, 1989), as well as the traditional unification algorithm (Colban, 1990). We also have been forced to omit much detail from the description and verification of our algorithm. A full report on this work will be available in the near future.

2. UNIFICATION IN THE PRESENCE OF CONSTRAINTS

2.1 Constraints on feature structures

Not every feature structure is a possibility as the ultimate output of the parsing mechanism. Typically, there are constraints which must be observed. One way of ensuring this sort of consistency is to build the checks right into the grammar, so that the feature structures generated are always legitimate substructures of the final output. The CLG formalism (Damas and Varile, 1989) is an example of such a philosophy. In many ways, this is an attractive option, because it provides a

[†]The research reported herein was performed while the author was visiting the COSMOS Computational Linguistics Group of the Mathematics Department at the University of Oslo. He wishes to thank Jens Erik Fenstad and the members of that group for providing a stimulating research environment. Particular thanks are due Tore Langholm for many invaluable discussions regarding the interplay of logic, feature structures, and unification.

unified context for expressing all aspects of the grammar. However, this approach has the disadvantage that it limits the use of independent parsing subalgorithms whose results are subsequently unified, since the consistency checks must be performed before the feature structures are presented to the unifier. Therefore, to maintain such independence, it would be a distinct advantage if some of the constraint checking could be relegated to the unification process.

To establish a formal framework in which this is possible, we must start by extending our notion of a feature structure. Following the ideas of Moshier and Rounds (1987) and Langholm (1989), we define an *extended feature structure* to be a pair $\langle N, \mathcal{K}\rangle$ in which $\mathcal{K}$ is a set of feature structures and N is the least element of $\mathcal{K}$ under the ordering $\sqsubseteq$. (Thus, by definition, $\mathcal{K}$ has a least element, and $\mathcal{K}$ determines N.) Think of N as the "current" feature structure, and $\mathcal{K}$ as the set of all structures into which N is allowed to grow. We define $\langle N_1, \mathcal{K}_1\rangle \sqsubseteq_x \langle N_2, \mathcal{K}_2\rangle$ to mean precisely that $\mathcal{K}_2 \subseteq \mathcal{K}_1$. In other words, the set of all structures which N_2 can grow into is a subset of those which N_1 can grow into. (It follows necessarily that $N_1 \sqsubseteq N_2$ in this case.) Note that if we identify the ordinary feature structure N with the pair $\langle N, \{M \mid N \sqsubseteq M\}\rangle$, we precisely recapture ordinary subsumption. Finally, the notion of unification associated with $\sqsubseteq_x$ is given by

$$\langle M_1, \mathcal{K}_1\rangle \sqcup_x \langle M_2, \mathcal{K}_2\rangle = \begin{cases} \langle M, \mathcal{K}_1 \cap \mathcal{K}_2\rangle & \text{if } \mathcal{K}_1 \cap \mathcal{K}_2 \\ & \text{has a least element } M; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

2.2 Logical feature structures with constraints To operate on pairs of the form $\langle N, \mathcal{K}\rangle$ algorithmically, we must have in place an appropriate representation for the set $\mathcal{K}$. There are many possible choices; ours is to let it be the set of all structures satisfying a set of sentences in a particular logic. The logic which we use is a simple modification of the language of Rounds and Kasper (1986) (see also (Kasper and Rounds, 1990)) admitting negation but only binary path equivalences. Specifically, an *atomic feature term* is one of the following.

Formula	Semantics
$\top$	The identically true term.
$\bot$	The identically false term.
$(\alpha : a)$	The path (nesting of attributes) α exists and terminates with label a.
$(\alpha \asymp \beta)$	The paths α and β have a common end point (coalesced end points).

In $(\alpha : a)$, the label a may be $\top$, denoting a missing value. The notation $(\alpha \asymp \beta)$ is borrowed from (Langholm, 1989), and has the same semantics as $\{\alpha, \beta\}$ of (Rounds and Kasper, 1986). A *(general) feature term* is built up from atomic feature terms using the connectives $\wedge$, $\vee$, and $\neg$, with the usual semantics. In particular, the negation we use is the classical notion; a structure satisfies $(\neg\varphi)$ if and only if it does not satisfy φ. For any set Φ of feature terms, $\mathsf{Mod}(\Phi)$ denotes the set of all feature structures for which each $\varphi \in \Phi$ is true. For a formal definition of satisfaction, we refer the reader to the above-cited references. Intuitively, any set of terms which defines a consistent rooted, directed graph is satisfiable. However, let us specifically remark that only nodes with no outgoing edges may have labels other than $\top$, that labels other than $\top$ may occur at at most one end point, that no two outgoing edges from the same node may have the same label, and that any term of the form $(\alpha : \bot)$ is equivalent to $\bot$, and so inconsistent.

Now we define a *logical extended feature structure* (*LoXF*) to be an extended feature structure $\langle N, \mathcal{K}\rangle$ in which $\mathcal{K} = \mathsf{Mod}(\Phi)$ for some consistent finite set Φ of feature terms. In particular, $\mathsf{Mod}(\Phi)$ must have a least model. We also denote this pair by $\mathcal{F}\langle\Phi\rangle = \langle N_\Phi, \mathsf{Mod}(\Phi)\rangle$. Now $\mathcal{F}\langle\Phi_1\rangle \sqsubseteq_x \mathcal{F}\langle\Phi_2\rangle$ reduces to $\mathsf{Mod}(\Phi_2) \subseteq \mathsf{Mod}(\Phi_1)$, and

$$\mathcal{F}\langle\Phi_1\rangle \sqcup_x \mathcal{F}\langle\Phi_2\rangle = \begin{cases} \mathcal{F}\langle\Phi_1 \cup \Phi_2\rangle & \text{if } \mathsf{Mod}(\Phi_1 \cup \Phi_2) \\ & \text{has a least element under } \sqsubseteq; \\ \text{undefined} & \text{otherwise.} \end{cases}$$

2.3 Remark on negation A full discussion of the nature of negation in LoXF's is complex, and will be the focus of a separate paper. However, because this topic has received a great deal of attention (Moshier and Rounds, 1987), (Langholm, 1989), (Dawar and Vijay-Shanker, 1990), we feel it essential to remark here that $\mathcal{F}\langle\Phi\rangle$ does not have the "classical" negation semantics which can be determined by looking solely at the least element. Indeed, the appropriate definition is that $\mathcal{F}\langle\Phi\rangle$ satisfies $\neg\varphi$ precisely when no member of $\mathsf{Mod}(\Phi)$ satisfies φ; in other words, the structure N_Φ is not allowed to be extended to satisfy φ.

2.4 Unification algorithms for logical extended feature structures In view of the definition immediately above, it is easy to see that that any unification algorithm for LoXF's must solve the following two problems in the course of attempting to unify $\mathcal{F}\langle\Phi_1\rangle$ and $\mathcal{F}\langle\Phi_2\rangle$.

(u1) It must decide whether or not $\Phi_1 \cup \Phi_2$ is consistent; *i.e.*, whether or not there is a feature structure satisfying all sentences of both Φ_1 and Φ_2.

(u2) In case that $\Phi_1 \cup \Phi_2$ is satisfiable, it must also determine if there is a least model, and if so, identify it.

Now it is well known that (u1) is an *NP*-complete problem, even if we disallow negation and path equivalence (Rounds and Kasper, 1986, Thm. 4). Therefore, barring the eventuality that $P = NP$, we cannot expect to allow Φ_1 and Φ_2 to be arbitrary finite sets of feature terms and still have a tractable algorithm for unification. One solution, which has been taken by a number of authors, such as Kasper (1989) and Eisele and Dörre (1988), is to devise clever algorithms which apply to the general case and appear empirically to work well on "typical" inputs, but still are provably exponential in the worst case. While such work is undeniably of great value, we here propose a companion strategy; namely, we restrict attention to pairs (N, Φ) such that the very nature of Φ *guarantees* a tractable algorithm.

3. HORN FEATURE LOGIC

In the field of mathematical logic in general, and in the computational logic relevant to computer science in particular, Horn clauses play a very special rôle (Makowsky, 1987). Indeed, they form the basis for the programming language *Prolog* (Sterling and Shapiro, 1986) and the database language *Datalog* (Ceri *et al.*, 1989). This is due to the fact that while they possess substantial representational power, tractable inference algorithms are well known. It is perhaps *the* main thesis of this work that the utility of Horn clauses carries over to computational linguistics as well.

3.1 Horn feature clauses A *feature literal* is either an atomic feature term (*e.g.*, $(\alpha : a)$, $(\alpha \asymp \beta)$, or $\bot$) or its negation. A *feature clause* is a finite disjunction $\ell_1 \vee \ell_2 \vee \ldots \vee \ell_m$ of feature literals. A feature clause is *Horn* if at most one of the ℓ_i's is not negated. A *Horn extended feature structure* (*HoXF*) is a LoXF $\mathcal{F}(\Phi)$ such that Φ is a finite set of Horn feature clauses.

3.2 A taxonomy of Horn feature clauses Before moving on to a presentation of algorithms on HoXF's, it is appropriate to provide a brief sketch of the utility and limits of restricting our attention to collections of Horn clauses. Implication here is classical; as in the case of ordinary propositional logic, we use the notation $\sigma_1 \wedge \sigma_2 \wedge \ldots \wedge \sigma_m \Rightarrow \rho$ to denote the clause $\neg\sigma_1 \vee \neg\sigma_2 \vee \ldots \vee \neg\sigma_m \vee \rho$. Horn feature clauses may then be thought of as falling into one of the following four categories.

(H1) A clause of the form σ, consisting of a single positive literal, is just a *fact*.

(H2) A clause of the form $\neg\sigma$, consisting of a single negative literal, is a *negated fact*. In terms of HoXF's, if $\neg\sigma \in \Phi$, this means that within $\mathcal{F}(\Phi)$, no extension of N_Φ in which σ is true is permitted. As a concrete example, a constraint stating that a subject may not have an attribute named "tense" would be of this form.

(H3) A clause of the form $\sigma_1 \wedge \sigma_2 \ldots \sigma_m \Rightarrow \rho$ is called a *rule* or an *implication*. Numerous examples of the utility of implication in linguistics are identified in (Wedekind, 1990, Sec. 1.3). Kasper's *conditional descriptions* (Kasper, 1988) are also a form of implication. More concretely, the requirement that a transitive verb requires a direct object is easily expressed in this form.

(H4) A clause of the form $\sigma_1 \wedge \sigma_2 \wedge \ldots \wedge \sigma_m \Rightarrow \bot$ is called a *compound negation*. The formalization of the constraint that a verb cannot be both intransitive and take a direct object is an example of the use of such a clause.

The type of knowledge which is not recapturable using Horn feature logic is positive disjunction; *i.e.*, formulas of the form $\sigma_1 \vee \sigma_2$, with both σ_1 and σ_2 feature terms. Of course, this has nothing in particular to do with feature-term logic, but is well-known limitation of Horn clauses in general. However, in accepting this limitation, we also obtain many key properties, including tractable inference and the following important property of genericity.

3.3 Totally generic LoXF's Let now Φ be any finite set of feature terms. We say that Φ is *totally generic* if, for any set Ψ of facts (see (H1) above), if $\mathsf{Mod}(\Phi \cup \Psi)$ is nonempty then it contains a least element under $\sqsubseteq$. Intuitively, if we use Φ to define the LoXF $\mathcal{F}(\Phi)$, total genericity says that however we extend the base feature structure N_Φ (consistently with Φ), we will continue to have a LoXF. Remarkably, we have the following.

3.4 Theorem *A set of feature terms Φ is totally generic if and only if it is equivalent to a set of Horn feature clauses.*

Proof outline: This result is essentially a translation of (Makowsky, 1987, Thm. 1.9) to the logic of feature structures. In words, it says that if (and only if) we work with HoXF's, condition (u2) on page 4 becomes superfluous (except for explicitly identifying the least model.) □

4. THE EXTENDED UNIFICATION ALGORITHM

It has been shown by Dowling and Gallier (1984) that satisfiability for finite sets of propositional Horn formulas can be tested in time linear in the length of the formulas. Their algorithms can easily be modified to deliver the least model as well. Since unification of HoXF's *is* essentially testing for satisfiability plus identifying the least model (see (u1)–u(2) on the previous page), a natural approach would be to adapt one of their algorithms. Essentially, this is what we do. Like theirs, our algorithm is *forward chaining*; we start with the facts and "fire" rules until no more can be fired, or until a contradiction appears. However, the adaptation is not trivial, because feature-term logic is more expressive than propositional logic. In particular, feature-term logic contains countably many tautologies which have no correlates in ordinary propositional logic. The main contribution of our algorithm is to implicitly recapture the full semantics of these tautologies while keeping the time complexity within reasonable bounds. Due to space limitations, we cannot present the full formality of the rather complex data structures. Rather, to highlight the key features, we step through an annotated example. We focus only upon the special problems inherent in the extension to feature-term logic, and assume familiarity with the forward-chaining algorithm in (Dowling and Gallier, 1984) and the graph unification algorithm in (Colban, 1990).

4.1 An example theory and extended feature graphs The set Ξ contains the following eight Horn feature clauses.

(ξ_1) $(AA : a)$.

(ξ_2) $(B : a)$.

(ξ_3) $(AA : a) \wedge (B : a) \Rightarrow (CCDDG : t)$.

(ξ_4) $(A : \top) \wedge (C : \top) \Rightarrow (ABDDG : \top)$.

(ξ_5) $(AA \asymp B) \wedge (ABDDG : \top) \Rightarrow (ABDDEF : \top)$.

(ξ_6) $(ABDD : \top) \wedge (B : \top) \Rightarrow (CCD \asymp ABD)$.

(ξ_7) $(CCDD \asymp ABDD) \Rightarrow (AC : \top)$.

(ξ_8) $(ACD : \top) \Rightarrow (ACC : t)$.

Just as we may represent a set of atomic feature terms with a *feature graph*, so too may we represent, in part, a set of Horn feature clauses with an *extended feature graph*. Shown in Figure 1 below is the initial extended feature graph for the set Ξ, representing the state of inference before any deductions are made.

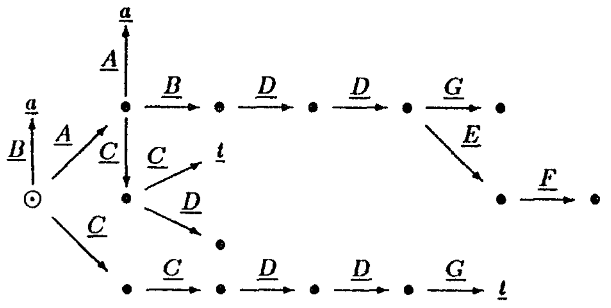

Figure 1: Initial extended feature structure for Ξ.

Every path and every node label which occurs in some literal of Ξ is represented. The labels of all edges, as well as all non-T node labels, are underscored, denoting that they are *virtual*, which means that they are only possibilities for the minimal model, and not yet actually part of it. The root node is denoted by $\odot$, and nodes with value T are denoted with a •. Note that paths with common virtual end labels (*e.g.*, AA and B) are not coalesced; virtual nodes and edges are never unified. As a result, the predecessors (along any directed path) of any actual node or edge is itself actual. As inferences are made, edges and nodes become actual (depicted by deleting underscores), and actual nodes with common labels are ultimately coalesced. The final extended feature graph is shown in Figure 2 below. For easier visibility, actual edges are also highlighted with heavier lines.

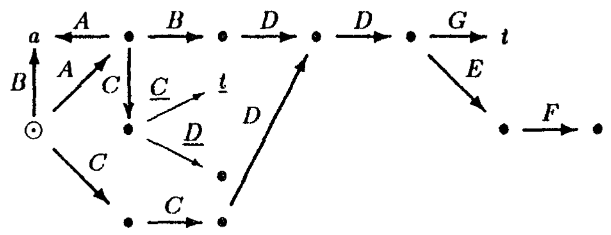

Figure 2: Final extended feature structure for Ξ.

If we delete the remaining virtual nodes and edges, we obtain the graphical representation of the least model of Ξ.

4.2 Computing the minimal model of the example Now let us consider the process of actually obtaining the structure of Figure 2 from Ξ. In the propositional forward chaining approach, we start by pooling the facts that we know — in this case $\{\xi_1, \xi_2\}$. We then look for rules whose left-hand sides have been satisfied. In the example, the left-hand side of ξ_3 is satisfied, so we may *fire* that rule and add $(CCDDG : t)$ to our list of known facts, exactly as in the propositional case. We may also conclude that $(AA \asymp B)$, because both are actual paths which terminate with the same label a, and non-T labels are unique. The representative extended feature graph at this point is shown in Figure 3 below.

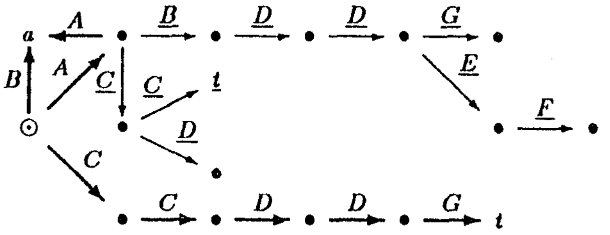

Figure 3: Intermediate structure for Ξ.

There are other things which we may implicitly conclude, and which we must conclude to fire the other rules. For example, we may fire rule ξ_4 at this point, because $(AA : a) \Rightarrow (A : \top)$ and $(B : a) \Rightarrow (B : \top)$ are both tautologies in the logic of feature terms, and so its left-hand side is satisfied. Thus, we may add $(ABDDG : \top)$ to our list of known facts. Similarly, since, as noted above, $(AA \asymp B)$ holds, we may fire rule ξ_5 to conclude $(ABDDEF : \top)$. Likewise, we may now fire rule ξ_6 and conclude $(CCD \asymp ABD)$. The representative extended graph structure at this point is shown in Figure 4 below.

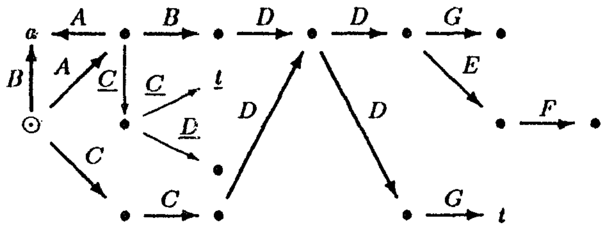

Figure 4: Intermediate structure for Ξ.

We must eventually invoke a unification at the common end point of CCD and ABD. Such unification implicitly entails the tautology $(CCD \asymp ABD) \Rightarrow (CCDD \asymp ABDD)$ and permits us to conclude that rule ξ_7 should fire and add $(AC : \top)$ to the set of facts of the least model. The result represented by the final extended feature graph of Figure 2. Note that rule ξ_8 never fires, and that there are virtual edges and nodes left at the conclusion of the process.

4.3 A taxonomy of implicit rules for sets of Horn feature clauses As we remarked in the introduction to this section, to correctly adapt forward chaining to the context of HoXF's, we must implicitly include the semantics of countably many tautologies. These fall into three classes.

(i1) Whenever an atomic term of the form $(\alpha\beta : a)$ is determined to be true ($\alpha\beta$ denotes the concatenation of α and β), and another term of the form

$(\alpha : \top)$ occurs as an antecedent of a Horn feature clause, (with either β not the empty string or else $a \neq \top$), we must be able to automatically make the deduction of the tautology $(\alpha\beta : a) \Rightarrow (\alpha : \top)$ to conclude that $(\alpha : \top)$ is now true. We call this *node and path subsumption*. In computing the least model of Ξ, the deductions $(AA : a) \Rightarrow (A : \top)$ and $(B : a) \Rightarrow (B : \top)$ are examples of such rules.

(i2) Whenever we deduce two terms of the form $(\alpha : a)$ and $(\beta : a)$ to be true, with $a \neq \top$, we must implicitly realize the semantics of the rule $(\alpha : a) \wedge (\beta : a) \Rightarrow (\alpha \asymp \beta)$, due to the constraint that non-$\top$ labels are unique. We call this *label matching*. In computing the least model of Ξ, the deduction $(AA : a) \wedge (B : a) \Rightarrow (AA \asymp B)$ is a specific example.

(i3) Whenever we coalesce two paths, we must perform local unification on the subgraph rooted at the point of coalescence. More precisely, if we coalesce the paths α and β, and the atom $(\alpha\gamma : a)$ is true, we must deduce that both $(\alpha\gamma \asymp \beta\gamma)$ and $(\beta\gamma : a)$ are true; *i.e.*, we must implicitly realize the compound rule $(\alpha \asymp \beta) \wedge (\alpha\gamma : a) \Rightarrow (\alpha\gamma \asymp \beta\gamma) \wedge (\beta\gamma : a)$. This is just a logical representation of *local unification*. In computing the least model of Ξ, a specific example is the deduction $(CCD \asymp ABD) \wedge (CCDDG : t) \Rightarrow (CCDDG \asymp ABDDG) \wedge (ABDDG : t)$.

4.4 Data structures To support these inferences, several specific data structures are supported. They are sketched below.

(d1) There is the list of clauses. Each clause has a counter associated with it, indicating the number of literals which remain to be fired before its left-hand side is satisfied. When this count drops to zero, the clause fires and its consequent becomes true.

(d2) There is a list of atoms which occur in the antecedents of clauses. With each literal is associated a set of pointers, one to each clause of which it is an antecedent literal. When an atom becomes true, the appropriate clauses are notified, so they may decrement their counters.

(d3) The *working extended feature structure*, as illustrated in Figures 1-4, is maintained throughout.

(d4) For each node in the working extended feature structure, a list of atoms is maintained. If the node label is a, then each such atom in the list is of the form $(\alpha : a)$, with α a path from the root node to the node under consideration. When that node becomes actual, that atom is notified that is is now satisfied.

(d5) For each non-$\top$ node label a which occurs in some atom, a list of all virtual nodes with that label is maintained. When one such node becomes actual, the other are checked to see if an inference of the form (i2) should be made.

(d6) For each atom of the form $(\alpha \asymp \beta)$ occurring as an antecedent in some clause, the nodes at the ends of these paths in the working extended feature structure are endowed with a common tag. Whenever nodes are coalesced, a check for such common tags is made, so the appropriate atom may be notified that it is now true.

4.5 Independent processes and unification The algorithm also maintains a *ready queue* of available processes. These processes are of three types. A process of the form Actual$(\alpha : a)$, when executed, makes the identified path and label actual in the extended feature graph. A process of the form Coalesce(n_1, n_2) coalesces the end points of the two nodes n_1 and n_2 in the extended feature graph. A process of the form Unify(n) performs a local unification at the subgraph rooted at node n, using an algorithm such as identified in (Colban, 1990). All processes in the ready queue commute; they may be executed in any order.

To unify two distinct sets of terms (perhaps generated by independent parts of a parser), we join their two extended feature graphs at the root, merge the corresponding data structures, and add the command Unify(root) to the merged process queue. In other words, we perform a unification to match common information, and then continue with the inference process.

4.6 The complexity of the unification algorithm Define the length of a literal to be the number of attribute name and attribute value occurrences in it. Thus, for example, length$((AB \asymp CD)) = 4$ and length$((ABCD : a)) = 5$. For a set Φ of Horn feature clauses, we further define the following quantities.

L = The length of Φ; *i.e.*, the sum of the lengths of all literals occurring in Φ.

P = The number of *distinct* terms of the form $(\alpha \asymp \beta)$ which occur as the right-hand side of a rule in Φ. (Facts are not considered to be rules here.)

m = The number of distinct attributes in the input. (If we collect all of the literals occurring in the clauses of Φ and discard any negation to yield a large pool of facts, then m is the number of edges in the graph representing the associated feature structure. If Φ is a set of positive literals to begin with, and hence represents an ordinary feature structure, then m represents the size of this feature structure.)

We then have the following theorem.

4.7 Theorem *The worst-case time complexity of our HoXF unification algorithm is $\Theta(L + (P+1) \cdot m \cdot \omega(m))$, where $\omega(m)$ is an inverse Ackermann function (which grows more slowly than than any primitive recursive function – for all practical purposes $\omega(n) \leq 5$).* □

This may be compared to the worst-case complexity of the usual algorithm for unifying ordinary feature structures, which is $\Theta(m \cdot \omega(m))$. The increase in complexity over this simpler case is due to two factors.

(c1) We must read the entire input; since literals may be repeated, it is possible that $L > m$; hence the L term.

(c2) Each time that we deduce that two nodes must be coalesced, we must perform a unification. This can occur at most P times – the number of times that a rule can assert a distinct coalescing of nodes.

4.8 Further remarks on the algorithm Note in particular that there are no restrictions on where path equivalences (*e.g.*, $(\alpha \asymp \beta)$) may occur in Horn feature clauses. In particular, unlike (Kasper, 1988), we do allow negated path equivalences. However, if we disallow path equivalences as consequents of rules, then the complexity of our algorithm becomes essentially that of the traditional unification algorithm (see (c2) above). It is primarily *deducing* path equivalences on the fly which results in the additional computational burden.

5. CONCLUSIONS, FURTHER DIRECTIONS, AND PROJECT STATUS

5.1 Conclusions and further directions We have identified HoXF's as an attractive compromise between ordinary feature structures (in which there is no way to express constraints on growth) and full logical feature theories (for which the unification problem is *NP*-complete). We view HoXF's not as the "best" approach, but rather as a tool to be used to build better overall unification-based grammar formalisms. The obvious next step is to develop an integrated framework in which HoXF's are employed to handle negation and the disjunction arising from implication, while other techniques handle more general disjunction and term subsumption (Smolka, 1988). Such an optimized approach could lead to much faster overall handling of negation and disjunction, but further work is clearly needed to bear this out.

5.2 Status of the project While the algorithm has been spelled out in considerable detail, we have just begun to build an actual implementation of the HoXF unifier in the programming language Scheme. We expect to complete the implementation by the summer of 1991.

References

Aït-Kaci, Hassan (1984), A lattice-theoretic approach to computation based on a calculus of partially-ordered type structures, PhD thesis, University of Pennsylvania, Philadelphia.

Ceri, Stefano; Gottlob, Georg; and Tanca, Letizia (1989), "What you always wanted to know about Datalog (and never dared to ask)," *IEEE Trans. Knowledge Data Engrg.*, 1, 146–166.

Colban, Erik A. (1988), Simplified unification based grammar formalisms, COSMOS Report No. 05, University of Oslo, Department of Mathematics.

Colban, Erik A. (1990), Unification algorithms, COSMOS Report No. 16, University of Oslo, Department of Mathematics.

Damas, Luis and Varile, Giovanni B. (1989), "CLG: a grammar formalism based on constraint resolution," in: Martins, João P. and Morgado, Ernesto M., eds., *EPIA 89: 4th Portugese Conference on Artificial Intelligence, Lisbon, Portugal, September 1989, Proceedings*, 175–186, Springer-Verlag.

Dawar, A. and Vijay-Shanker, K. (1990), "An interpretation of negation in feature structure descriptions," *Computational Linguistics*, **16**, 11–21.

Dowling, William F. and Gallier, Jean H. (1984), "Linear-time algorithms for testing the satisfiability of propositional Horn clauses," *J. Logic Programming*, **3**, 267–284.

Eisele, Andreas and Dörre, Jochen (1988), "Unification of disjunctive feature descriptions," in: *Proceedings of the 26th Annual Meeting of the ACL.*

Fenstad, Jens Erik; Langholm, Tore; and Vestre, Espen (1989), Representations and interpretations, COSMOS Report No. 09, University of Oslo, Department of Mathematics, To appear in *Proceedings of the Workshop on Computational Linguistics and Formal Semantics, Lugano, August-September 1988.*

Kasper, Robert T. (1988), "Conditional descriptions in functional unification grammar," in: *Proceedings of the 26th Annual Meeting of the ACL, Buffalo*, 233–240.

Kasper, Robert T. (1989), "A unification method for disjunctive feature descriptions," in: *Proceedings of the 25th Annual Meeting of the ACL*, 235–242.

Kasper, Robert T. and Rounds, William C. (1990), "The logic of unification in grammar," *Linguistics and Phil.*, **13**, 35–58.

Langholm, Tore (1989), How to say no with feature structures, COSMOS Report No. 13, University of Oslo, Department of Mathematics.

Makowsky, Johann A. (1987), "Why Horn formulas matter in computer science: initial structures and generic examples," *J. Comput. System Sci.*, **34**, 266–292.

Moshier, M. Drew and Rounds, William C. (1987), "A logic for partially specified data structures," in: *Conference Record of the 14th Annual ACM POPL Symposium*, 155–167.

Rounds, William C. and Kasper, Robert (1986), "A complete logical calculus for record structures representing linguistic information," in: *Proceedings of the First IEEE Symposium on Logic in Computer Science*, 38–43.

Shieber, Stuart M. (1986), *An Introduction to Unification-Based Approaches to Grammar*, University of Chicago Press.

Smolka, Gert (1988), A feature logic with subsorts, LILOG-Report 33, IBM Deutschland GmbH, Stuttgart.

Sterling, Leon and Shapiro, Ehud (1986), *The Art of Prolog*, MIT Press.

Wedekind, Jürgen (1990), A survey of linguistically motivated extensions to unification-based formalisms, Deliverable R3.1.A, DYANA.