The Specification and Implementation of Constraint-Based Unification Grammars*

Bob Carpenter Carl Pollard† Alex Franz

Philosophy Department, Carnegie Mellon University, Pittsburgh, PA 15213
(412) 268-8573 carp@lcl.cmu.edu
†Linguistics Department, Ohio State University

Summary

Our aim is to motivate and provide a specification for a unification-based natural language processing system where grammars are expressed in terms of principles which constrain linguistic representations. Using typed feature structures with multiple inheritance for our linguistic representations and definite attribute-value logic clauses to express constraints, we will develop the bare essentials required for an implementation of a parser and generator for the Head-driven Phrase Structure Grammar (HPSG) formalism of Pollard and Sag (1987).

1 Introduction

In the past decade, two competing approaches to the scientific study of natural language grammar have become predominant, the *rule-based* approach and the *principle/constraint-based* approach. Within the rule-based approach, exemplified by Lexical Functional Grammar (LFG) (Bresnan 1982) and Generalized Phrase Structure Grammar (GPSG) (Gazdar *et al.* 1985), rules are taken to correspond to grammatical *constructions* and are modeled as more or less schematic productions with the well-formed structures of the language generated over a finite set of *lexical items* by recursively applying the rules. Both LFG and GPSG are based upon context-free skeletons and explain syntactic dependencies in terms of informational consistency constraints that can be solved using feature structure unification. There has been a great deal of success in implementing these formalisms, in part due to their declarative nature and natural semantics, but also due to the existence of general unification-based grammar processing systems such as Functional Unification Grammar (FUG) (Kay 1985) and PATR-II (Shieber *et al.* 1983).

Principle-based approaches to grammar have become predominant in theoretical linguistics, primarily due to the influence of Chomsky's (1981) Government-Binding (GB) framework. The novel aspect of GB considered as a grammar formalism is that it advocates the total abandonment of construction-specific rules in favor of a collection of interacting principles which serve to delimit the well-formed linguistic structures. Candidate structures are generated according to extremely general, universal, phrasal immediate dominance (ID) schemata ($\bar{X}$ Theory) and then iteratively transformed using movement rules (Move-α) in accordance with a number of highly tuned principles to deal with case (Case Theory), complementation (Projection Principle), pronominal and other coreference (Binding Theory), long-distance dependencies

*The authors would like to thank Bob Kasper for a number of useful comments on an earlier draft of this paper. The research of Pollard and Franz was supported by a grant from the National Science Foundation (IRI-8806913).

(Empty Category Principle and Subjacency) and so forth. Patterns of cross-linguistic variation are accounted for by means of the *parametrization* of these principles.

The methodological distinction between these two approaches is widely supposed to be that rules enumerate possibilities, while principles eliminate possibilities. But it is quite difficult to distinguish formally between a parametrized disjunctive principle and a collection of schematic rules only one of which can apply to a given structure. Consider, for example, the distinction between categorial grammar application schemata, basic ID rules of GPSG, and the C-structure constraints of LFG, on the one hand, and the disjunctive clauses of $\bar{X}$ Theory or the Empty Category Principle on the other. It should also be borne in mind that so-called rule-based approaches often employ not only rules but also global constraints on representations which behave similarly to principles, such as the Head Feature Convention and the Control Agreement Principle of GPSG or the Completeness and Function-Argument Biuniqueness Conditions of LFG.

HPSG belongs to the "unification-based" family of linguistic theories, but differs from LFG and GPSG in that grammars are formulated entirely in terms of universal and language-specific principles expressed as constraints on feature structures, which in turn are taken to represent possible linguistic objects. As shown by Pollard and Sag (1987), constraints on feature structures can be used to do the same duty as many of the principles and rules of GPSG, LFG and GB. Unlike rule-based theories, in HPSG, immediate dominance and linear precedence conditions (traditional phrase-structure) are not modeled any differently than other constraints. But like the rule-based approaches, there is no appeal to derivational notions such as movement; the work of transformations in GB is taken over by declarative constraints stated at a single level of representation.

Departing from more traditional formalisms which employ phrase-structure trees as the primary device for linguistic representation, we follow HPSG (and to some extent LFG) in representing linguistic objects as feature structures. To this end, we show how a natural type discipline can be imposed on feature structures allowing for multiple inheritance and the specification of feature appropriateness and value restrictions. Our typing will be strong in the sense that every feature structure must be associated with a type. Strong typing carries with it the usual benefits of early error detection and enhanced control over crucial memory allocation, access and reclamation functions. The use of multiple inheritance allows a sophisticated network of constraints to be expressed at the appropriate level of detail. This is especially important for the development of large lexicons (Flickinger *et al.* 1985).

Our types can be used to represent information that must be encoded by expensive structural unification or inference steps in untyped systems. In automatic deduction systems, this has been found to provide a significant run-time gain due to the fact that useless branches in the search space can be efficiently detected and pruned before the creation of expensive structural copies or binding frames (Walther 1985, 1988; Aït-Kaci and Nasr 1986).

In a constraint-based linguistic theory such as HPSG, parsing and generation reduces to solving constraints. We allow constraints to be expressed by a feature logic analogue of definite clauses. The benefit of this approach is that it admits a natural and effective method paralleling SLD-resolution (see Lloyd 1984) for enumerating the solutions to a system of constraints.

2 Inheritance and Appropriateness

Type declarations in our system contain information concerning subtyping and *appropriateness conditions* which state the features that are appropriate for each type and the values that they can take.

Definition 1 (Type Scheme) *A* type scheme *is a tuple* $\Sigma = \langle \mathsf{Type}, \sqsubseteq, \mathsf{Feat}, Approp \rangle$ *where*

- $\langle \mathsf{Type}, \sqsubseteq \rangle$ *is a finite consistently complete*[‡] *partial order of the* types *by* subsumption, *called the* inheritance hierarchy
- Feat *is a finite set of* features
- $Approp : \mathsf{Feat} \times \mathsf{Type} \rightarrow \mathsf{Type}$ *is a partial function such that:*
  - (Minimal Introduction)
    for every f *there is a least type* σ *such that* $Approp(f, \sigma)$ *is defined*
  - (Upward Closure and Monotonicity)
    if $\sigma \sqsubseteq \tau$ *and* $Approp(f, \sigma)$ *is defined then* $Approp(f, \tau)$ *is defined and* $Approp(f, \sigma) \sqsubseteq Approp(f, \tau)$

If $\sigma \sqsubseteq \tau$ we say that σ *subsumes*, is *more general* than or a *supertype* of τ. We refer to the least upper bound operation in our inheritance hierarchy as *(type) unification* since the least upper bound of a set of objects representing partial information is the object which represents the most general piece of information that is more specific than each member of the set. The least upper bound of the empty set is written $\bot$, read "bottom", and is the unique *universal* or most general type. Our restrictions on appropriateness are analogous to the condition of regularity in the signatures of order-sorted algebras (Meseguer *et al.* 1987); taken together, the conditions on the inheritance hierarchy and appropriateness function will ensure that unification is well-defined and produces a unique result, which is crucial for efficient and natural unification-based processing (Pereira 1987).

3 Feature Structures

We will begin by introducing an untyped collection of *feature structures* which are similar to the ψ-terms of Aït-Kaci (1984) and the sorted feature structures of Smolka (1988) and Pollard and Moshier (1990).

Definition 2 (Feature Structure) *A* feature structure *is a tuple* $F = \langle Q, \bar{q}, \theta, \delta \rangle$ *where*

- $\bar{q}$ *: the* root node *in* Q
- Q *: a finite set of* nodes *rooted at* $\bar{q}$ *so that* $Q = \{\delta(\pi, \bar{q}) \mid \pi \in \mathsf{Path}\}$ *(see below for definition of* $\delta(\pi, q)$ *and* Path*)*

[‡]A subset X of a partial ordering $\langle S, \sqsubseteq \rangle$ is said to be ***consistent*** if it has an upper bound. A partial order is ***consistently complete*** if every (possibly empty) consistent set X has a ***least*** upper bound, which we write $\bigsqcup X$, or $x \sqcup y$ when $X = \{x, y\}$.

- $\theta : Q \rightarrow \mathsf{Type}$ *: a total node* type assignment
- $\delta : \mathsf{Feat} \times Q \rightarrow Q$ *: a partial* feature value function

Thus a feature structure is a rooted, connected, directed graph with vertices labeled by types and edges labeled by features. We will write $q : \sigma \xrightarrow{f} q' : \sigma'$ if $\delta(f, q) = q'$ and $\theta(q) = \sigma$ and $\theta(q') = \sigma'$. We think of each node as representing a partial frame or record with values for its slots given by its outgoing arcs.

We let $\mathsf{Path} = \mathsf{Feat}^*$ be the set of *paths*, which consist of finite sequences of features. We let ϵ denote the empty path and extend δ to paths by setting $\delta(\epsilon, q) = q$ and $\delta(f\pi, q) = \delta(\pi, \delta(f, q))$. Our definition requires that every node be reachable from the root node $\bar{q}$, where $\delta(\pi, \bar{q})$ is the node that can be reached from $\bar{q}$ along the path π.

Note that we have not disallowed *cyclic* feature structures, in which there is some non-empty path π and node q such that $\delta(\pi, q) = q$.

We extend our ordering on types to an ordering of the feature structures in the usual way (see Pollard and Moshier 1990).

Definition 3 (Subsumption) $F = \langle Q, \bar{q}, \theta, \delta \rangle$ subsumes $F' = \langle Q', \bar{q}', \theta', \delta' \rangle$, $F \sqsubseteq F'$, *iff there is a total* $h : Q \rightarrow Q'$ *such that*

- $h(\bar{q}) = \bar{q}'$
- $\theta(q) \sqsubseteq \theta'(h(q))$ *for every* $q \in Q$
- $h(\delta(f, q)) = \delta'(f, h(q))$ *for every* $q \in Q$ *and feature* f *such that* $\delta(f, q)$ *is defined*

The last two conditions on h can be stated graphically as requiring that if $q : \sigma \xrightarrow{f} q' : \sigma'$ in F then $h(q) : \tau \xrightarrow{f} h(q') : \tau'$ in F' and furthermore, $\sigma \sqsubseteq \tau$ and $\sigma' \sqsubseteq \tau'$. Such a mapping takes each node of the more general structure onto a node of the more specific structure in a way that preserves structure sharing and does not lose any type information.

Subsumption is only a *pre-ordering*, so we write $F \sim F'$ if $F \sqsubseteq F'$ and $F' \sqsubseteq F$ and say that F and F' are *alphabetic variants*. We could work in the collection of feature structures modulo alphabetic variance, which is guaranteed to be a partial order, but this becomes tedious (for an elegant approach to representing these equivalence classes, see Moshier (1988)). In our situation, with only a pre-order, we define a *unifier* of a pair of feature structures F and F' to be any feature structure F'' such that $F \sqsubseteq G$ and $F' \sqsubseteq G$ if and only if $F'' \sqsubseteq G$. The primary result concerning subsumption is stated as follows:

Theorem 4 (Unification) *Unique unifiers exist for pairs of consistent feature structures, up to alphabetic variance.*

Proof: The usual unification algorithm for feature structure works with the addition of a step that unifies the types of the inputs to produce the type of the result and fails if the types are not consistent. See Aït-Kaci (1984) or Pollard and Moshier (1990). □

In theory, the asymptotic behavior of the unification algorithm is not affected; type unification can be carried out by table look-up. In practice, the negligible constant overhead of type unification at every step of the process will actually save time in that inconsistencies can be detected before any recursive structures need to be inspected.

We now define a notion of typing which singles out some of the feature structures as being well-typed. Intuitively, a feature structure is well-typed if every feature that appears is appropriate and takes an appropriate value.

Definition 5 (Well-Typing) *A feature structure* $F = \langle Q, \bar{q}, \theta, \delta \rangle$ *is* well-typed *if* $q : \sigma \xrightarrow{f} q' : \sigma'$ *in* F *implies* $Approp(f, \sigma) \sqsubseteq \sigma'$.

If F is a feature structure and F' a well-typed feature structure such that $F \sqsubseteq F'$ then we say that F' is a *well-typed extension* of F and that F is *typable*.

Fortunately, the user does not need to specify all of the values for appropriate features about which nothing is known; a type inference procedure can be defined that determines the minimum possible types that will extend a feature structure so that it is well-typed.

Theorem 6 (Type Inference) *There is an effectively computable partial function* $TypInf$ *from the feature structures onto the well-typed feature structures such that* $TypInf(F)$ *is defined if and only if* F *is typable. In that case* $F \sqsubseteq F'$ *for a well-typed* F' *if and only if* $TypInf(F) \sqsubseteq F'$.

Proof: A constructive type inference procedure can proceed by successively increasing the types on those nodes which do not yet meet the appropriateness conditions. All that is required is the iteration of the following steps until a closure is reached:

- if a feature is defined at a node, the type of the node should be unified with the minimal type appropriate for the feature.
- if a feature is defined and its value is not of great enough type, unify in the type for the minimal value.

Thus every typable feature structure has a minimal well-typed extension which is unique up to alphabetic variance. This process is not sensitive to the order in which nodes and features are chosen. It is also guaranteed to terminate as there are only a finite number of types and nodes to start with. To see that the result is minimal, simply notice that each operation in the iteration was required so that the result is well-typed. □

The function $TypInf$ displays a host of interesting properties. For instance, it can be factored into two separate operations corresponding to the two steps in $TypInf$. It is not hard to see that that $F \sqsubseteq TypInf(F)$, $TypInf(F) = TypInf(TypInf(F))$, and $F \sqsubseteq F'$ implies that $TypInf(F) \sqsubseteq TypInf(F')$. More significantly, we have:

(1) $TypInf(TypInf(F_1) \sqcup \cdots \sqcup TypInf(F_n)) = TypInf(F_1 \sqcup \cdots \sqcup F_n)$

whenever the latter exists. This means that we can be as lazy as we like about type inference at run time without fear of information loss. It also follows that the type inference procedure can be composed with a unification procedure for feature structures to provide a unification procedure for well-typed feature structures.

Theorem 7 (Well-Typed Unification) *If* F *and* F' *are consistent well-typed feature structures such that* $F \sqcup F'$ *is typable, then* $TypInf(F \sqcup F')$ *is their least upper bound in the collection of well-typed feature structures (modulo alphabetic variance).*

The significance of this theorem is that it will be possible to compute the least specific well-typed feature structure that extends a consistent pair of well-typed feature structures.

This notion of well-typing is not the only one possible. It is also sensible to consider a stronger notion of typing whereby every feature that is appropriate must be defined. This notion, called *total* well-typing, corresponds to the composition of $TypInf$ with a second closure operator that adds in features that have not been defined in $TypInf(F)$ and gives them their minimal values. As Franz (1990) points out, the appropriateness conditions must meet a certain acyclicity condition to ensure the termination of type inference for this stronger notion of typing. These more strongly typed systems allow better management of memory since feature structures of a given type are of a known size and can have their feature values indexed positionally rather than by feature/value pairs. On the other hand, the notion of well-typing that we consider here is simpler and is also better suited to applications in which the number of features containing information is sparse relative to the number of possible features that can be defined for any given feature structure. For instance, in the application to HPSG we provide below, feature structures occurring early in the search space will be quite sparse compared to their later instantiations.

4 Feature Logic

We can describe our feature structures with a variant of the feature logic introduced by Rounds and Kasper (1986). We present a simultaneous definition of both the *well-formed formulas* or *descriptions* and of *satisfaction* of a formula by a feature structure, which we write $F \models \phi$:

Definition 8 (Formulas and Satisfaction)

Formula	Satisfaction Condition
$F \models \sigma$	*the root node of* F *is assigned a type at least as specific as* σ
$F \models \pi : \phi$	*the value of* F *at* π *is defined and satisfies* ϕ
$F \models \pi \doteq \pi'$	*the paths* π *and* π' *lead to the same node in* F
$F \models \phi \wedge \psi$	$F \models \phi$ *and* $F \models \psi$.
$F \models \phi \vee \psi$	$F \models \phi$ *or* $F \models \psi$.

The behavior of this logic on the typed feature structures we present here can be given a complete equational axiomatization along the lines of Rounds and Kasper (1986) by adding in additional axioms for type unification (Pollard in press) and appropriateness (Pollard and Carpenter to appear). The primary result of Rounds and Kasper carries over to the present situation:

Theorem 9 (Minimal Satisfiers) *For every formula* ϕ *there is a finite set* $\{F_0, \ldots, F_{n-1}\}$ *of pairwise incomparable feature structures, unique up to alphabetic invariance, such that* $F \models \phi$ *if and only if* $F_i \sqsubseteq F$ *for some* $i < n$.

Proof: The proof of Rounds and Kasper (1986) can be easily adapted by applying the type inference procedure. The key result is that the set of minimal satisfiers of a conjunction is derived by the pairwise unification of the minimal satisfiers of the conjuncts. □

5 Constraint Systems and Solutions

Departing from Pollard and Sag (1987) and following Pollard and Moshier (1990), we attach constraints to types rather than allowing general implicative and negative constraints. The

constraints attached to types will be much more expressive than the easily decidable conditions arising from the inheritance and appropriateness conditions in the type scheme which are only intended to specify the class of well-typed feature structures over which the constraints range.

Definition 10 (Constraint System) *A* constraint system Φ *associates each type* τ *with a feature logic formula* Φ_τ.

We provide for the multiple inheritance of constraints, letting $\downarrow\Phi_\tau$ be the conjunction of the constraints associated with τ and all of its supertypes; formally $\downarrow\Phi_\tau = \bigwedge_{\sigma \sqsubseteq \tau} \Phi_\sigma$. Since the feature structures in Φ_σ may contain arbitrary types, the system Φ may be recursive. Pollard and Sag (1987) show how systems of constraints of this general form can be used to model not only language-specific grammars, but also entire linguistic theories (universal grammars).

In general, we will be interested in solving *queries* with respect to systems of constraints, where a query simply consists of an feature logic description. In applications to parsing, a query would represent the value of the phonology feature and a constraint on the syntactic category of the result; for generation, a query might represent instantiated semantic and pragmatic features. A solution is then a well-typed feature structure which satisfies both the query and all of the constraints expressed by the grammar.

Definition 11 (Solution) *A feature structure* F *is a solution to a query* ψ *with respect to a system* Φ *of constraints just in case* $F \models \psi$ *and the maximal substructure* F_q *rooted at each node* q *of* F *satisfies the inherited constraint on its type* $\theta(q)$, *so that* $F_q \models \downarrow\Phi_{\theta(q)}$.

We will provide a complete method for generating the solutions to queries with respect to constraint systems that is defined in terms of non-deterministic feature structure rewriting. Our method is inspired by the rewriting operation employed by Aït-Kaci (1984), which, to the best of our knowledge, was the first programming system based upon recursively defined constraints on feature structures; but our method is cleaner in that it provides a strong distinction between the logical language and its feature structure models and also more general in that it applies to cyclic feature structures. More importantly, our system is provably complete.

The basic operation of rewriting is to non-deterministically choose a node in the feature structure and then non-deterministically choose a minimal satisfier for the inherited constraint associated with the type attached to that node to be unified into the feature structure. This is analogous to SLD-resolution as applied to definite clauses, in which a subgoal is replaced by the body of a clause after unifying the head of the clause with the subgoal.

Let $\pi \cdot F$ be the feature structure consisting of the path π with F attached to its terminal node.

Definition 12 (Rewriting) *If* F *is a well-typed feature structure where the node at path* π *is assigned type* σ *and if* G *is a minimal satisfier of the inherited constraint* $\downarrow\Phi_\sigma$ *on* σ *then* rewriting *is defined so that* $F \stackrel{\pi}{\Longrightarrow} TypInf(F \sqcup \pi \cdot G)$.

Of course, as we mentioned earlier, type inference can be interleaved arbitrarily with this rewriting operation. We can also interleave rewriting along arbitrary paths, using the notation $F \Longrightarrow F'$ if $F \stackrel{\pi}{\Longrightarrow} F'$ for some path π.

Our next theorem shows that minimal solutions can be effectively generated by rewriting. In effect, it is the completeness theorem for our operational interpretation; it tells us that every solution can be found by rewriting. In particular, a breadth-first enumeration of the search space determined by the rewriting system will eventually uncover every solution.

Theorem 13 (Solution) *F is a* solution *to the query ψ with respect to the constraint system Φ if and only if $F \stackrel{\pi}{\Longrightarrow} F$ for every path π for which F is defined. F is a minimal solution if and only if there is a derivation of F by rewriting from a minimal satisfier of ψ.*

Proof: (Sketch) The conditions on a solution are just that every node satisfy the constraint on its type. This happens if and only if the unifying in of a minimal satisfier to the constraint does not add any new information.

The usual fixed-point style induction suffices to establish minimality. Suppose we fix a solution to the query ψ. In the base case, this solution must be more specific than a minimal satisfier of the query ψ. The inductive hypothesis is that at every stage during rewriting we are dealing with a feature structure which subsumes the solution.

During rewriting, we unify in constraints associated with more general types than in the solution since we have a feature structure which subsumes the solution. Since we inherit constraints, rewriting can be done so that it unifies in a minimal satisfier to a constraint which subsumes the minimal satisfier associated with the corresponding node in the solution. The rewriting process will eventually reach a solution after a finite number of steps or continue on indefinitely, because there are only a finite number of steps that can be taken without adding in more nodes due to the finite number of nodes in a feature structure and finite number of types in the inheritance hierarchy.

If rewriting reaches a solution, then by the inductive hypothesis, that solution must be at least as general as the given solution. Finite satisfiers which are not generated by rewriting from a minimal satisfier of the query can thus not be minimal. □

For the sake of brevity we have not discussed constraints which express n-ary relational dependencies between path values. An example of a relational dependency expression would be $append(\pi_1, \pi_2, \pi_3)$, where π_1, π_2, and π_3 are paths; this means that the value of the path π_3 must be the concatenation of the values of π_1 and π_2. Such relations can be given definite-clause-style recursive definitions, as in:

$$(2) \quad \mathbf{append}(\pi_1, \pi_2, \pi_3) \leftarrow (\pi_1 : \mathbf{nil} \wedge \pi_2 \doteq \pi_3) \vee (\pi_1 \cdot \textsc{first} \doteq \pi_3 \cdot \textsc{first} \wedge \mathbf{append}(\pi_1 \cdot \textsc{rest}, \pi_2, \pi_3 \cdot \textsc{rest}))$$

Adding definitions of this kind to our feature logic is somewhat analogous to augmenting an underlying constraint language with definite relations as proposed by Höhfeld and Smolka (1988). However, it should be borne in mind that the π_i in our definition schemata are path parameters, not genuine logical variables. Aït-Kaci (1984) showed how relations could be encoded as types with arguments specified by features and arbitrary constraints for definitions; each use of such a relation then requires a node in a feature structure at which to be anchored (usually as the value of a so-called *garbage feature*). Franz (1990) implemented relations directly, requiring their arguments to be typed; in the case of *append*, all of the arguments would be of type **list**, which has two subtypes: **nil** (empty list), which is not appropriate for any features, and **ne-list** (nonempty list), which is appropriate for the features FIRST with value restriction ⊥ and REST with value restriction **list**.

6 Implementation

The typed system described here has been implemented in both Lisp (Franz 1990) and Prolog. Emele and Zajac (personal communication) report that Franz's (1990) grammar has been

ported, with a 100-fold speedup, to their TFS system (1990) which was originally based on Aït-Kaci (1984, 1986). We anticipate that a number of the optimizations employed in TFS will carry over to the system described here. In Franz's system, compilation is first carried out on the type scheme and constraints to detect errors and compute minimal satisfiers. A serious processing bottleneck can be traced to the search incurred by disjunctive constraint solving. This naturally leads to the issue of which search strategy should be employed. The conclusion of Franz (1990) was that specialized search strategies would be needed for linguistic applications. Ideally, a general mechanism for specifying search preference would be provided.

The complexity of the basic operations of this system is very low. Subsumption can be computed in linear time by explicit construction of the mapping function. Similarly, efficient near-linear unification algorithms can be used (Jaffar 1984). On the other hand, disjunctive representations are very compact in that the number of minimal satisfiers for a formula is exponential in the size of the formula in the worst case and satisfiability of a formula is $\mathcal{NP}$-complete (Kasper and Rounds 1986). Relatively efficient practical algorithms for dealing with disjunctions have been developed by Kasper (1987) and Eisele and Dörre (1988). Another option that is being explored is the utilization of total typing as discussed above, for managing memory allocation and improving the speed of both unification and the unwinding of information upon backtracking. The features values themselves could then be retrieved automatically without searching through a collection of feature-value pairs. Hopefully, compilation and run-time optimization techniques employed for logic programs can also be directly incorporated, such as type indexing for rules and deterministic tree pruning.

Furthermore, the connections between constraint-based grammars and terminological knowledge representations based on inheritance networks such as KL-ONE (Brachman and Schmolze 1985) and especially its descendant LOOM (Mac Gregor 1988) has only begun to be explored (Kasper 1989, Nebel and Smolka 1989); there is a great deal of promise that insights from these systems can be employed to produce more powerful and efficient type inference and search techniques. Kasper and Pollard are currently exploring the possibility of a chart-parser analog for HPSG-style grammars that exploits the possible-worlds mechanism of LOOM for conceptually clean and space-efficient structure sharing within the chart.

There are many possible extensions that could be added to our constraint systems. In particular, Pollard and Moshier (1990) have provided a compatible account of set valued feature structures, Carpenter (1990) has added a notion of inequation analagous to the inequations of Prolog II (Colmerauer 1984), and a general notion of feature structure extensionality is discussed in Pollard and Carpenter (to appear).

One thing that this system shares with PATR-II and other general unification-based systems is that while the solutions to queries can be recursively enumerated, it is undecidable whether a query has a solution. While we do not present a proof here, the result follows from the fact that logic programs and queries can be reduced to the solution of a system of constraints (the trick is to include proof trees as a type and encode the notion of an acceptable proof tree with respect to a program as a constraint on its type). Of course, this does not render our system unusable any more than Prolog or PATR-II are rendered useless by their undecidability; it just means that the user must exercise due caution in constructing linguistically reasonable grammars, in order to ensure that all-paths parsing always terminates. In generation, of course, nontermination is to be expected; but in this case, fortunately, a single solution will suffice.

References

Aït-Kaci, H. (1984). *A Lattice-Theoretic Approach to Computation Based on a Calculus of Partially Ordered Types.* PhD thesis, University of Pennsylvania.

Aït-Kaci, H. (1986). An algebraic semantics approach to the effective resolution of type equations. *Theoretical Computer Science*, 45:293–351.

Aït-Kaci, H. and Nasr, R. (1986). Login: A logical programming language with built-in inheritance. *Journal of Logic Programming*, 3:187–215.

Brachman, R. J. and Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216.

Bresnan, J. W., editor (1982). *The Mental Representation of Grammatical Relations.* MIT Press, Cambridge, Massachusetts.

Carpenter, B. (1990). Typed feature structures: Inheritance, (in)equations and extensionality. In *Proceedings of the First International Workshop on Inheritance and Natural Language*, Tilburg, The Netherlands.

Chomsky, N. (1981). *Lectures on Government and Binding.* Foris, Dordrecht.

Colmerauer, A. (1984). Equations and inequations on finite and infinite trees. In *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo.

Eisele, A. and Dörre, J. (1988). Unification of disjunctive feature descriptions. In *Proceedings of the 26th Annual Conference of the Association for Computational Linguistics*, Buffalo, New York.

Emele, M. C. and Zajac, R. (1990). Typed unification grammars. In *Proceedings of the 13th International Conference on Computational Linguistics*, Helsinki, Finland.

Flickinger, D., Pollard, C. J., and Wasow, T. (1985). Structure-sharing in lexical representation. In *Proceedings of the 23rd Annual Conference of the Association for Computational Linguistics.*

Franz, A. (1990). A parser for HPSG. Technical Report LCL-90-3, Laboratory for Computational Linguistics, Carnegie Mellon University, Pittsburgh.

Gazdar, G., Klein, E., Pullum, G., and Sag, I. (1985). *Generalized Phrase Structure Grammar.* Basil Blackwell, Oxford.

Höhfeld, M., and Smolka, G. (1988) Definite Relations over Constraint Languages. LILOG-REPORT 53, IBM Deutschland GmbH, Stuttgart, FRG.

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

Johnson, M. (1988). *Attribute-Value Logic and the Theory of Grammar*, volume 14 of *Lecture Notes.* Center for the Study of Language and Information, Stanford, California.

Kasper, R. T. (1987). A unification method for disjunctive feature structures. In *Proceedings of the 25th Annual Conference of the Association for Computational Linguistics*, pages 235–242.

Kasper, R. T. (1989). Unification and classification: An experiment in information-based parsing. In *First International Workshop on Parsing Technologies*, pages 1–7, Pittsburgh.

Kasper, R. T. and Rounds, W. C. (1986). A logical semantics for feature structures. In *Proceedings of the 24th Annual Conference of the Association for Computational Linguistics*, pages 235–242.

Kay, M. (1985). Parsing in functional unification grammar. In Dowty, D. R., Karttunen, L., and Zwicky, A., editors, *Natural Language Parsing*, pages 206–250. Cambridge University Press, London.

King, P. (1989). *A Logical Formalism for Head-Driven Phrase Structure Grammar.* PhD thesis, University of Manchester, Manchester, England.

Lloyd, J. W. (1984). *Foundations of Logic Programming.* Springer-Verlag, West Berlin, FRG.

Mac Gregor, R. (1988). A deductive pattern matcher. In *Proceedings of the 1988 National Conference on Artificial Intelligence*, pages 403–408, St. Paul, Minnesota.

Meseguer, J., Goguen, J., and Smolka, G. (1987). Order-sorted unification. Technical Report CSLI-87-86, Center for the Study of Language and Information, Stanford University, Stanford, California.

Moshier, D. (1988). *Extensions to Unification Grammar for the Description of Programming Languages.* PhD thesis, University of Michigan, Ann Arbor.

Moshier, M. A. (1989). A careful look at the unification algorithm. Unpublished Manuscript, Department of Mathematics, University of California, Los Angeles.

Mycroft, A. and O'Keefe, R. A. (1984). A polymorphic type system for Prolog. *Artificial Intelligence*, 23:295–307.

Nebel, B. and Smolka, G. (1989). Representation and reasoning with attributive descriptions. IWBS Report 81, IBM – Deutschland GmbH, Stuttgart, FRG.

Pereira, F. C. (1987). Grammars and logics of partial information. In Lassez, J.-L., editor, *Proceedings of the Fourth International Symposium on Logic Programming*, pages 989–1013.

Pereira, F. C. N. and Shieber, S. M. (1984). The semantics of grammar formalisms seen as computer languages. In *Proceedings of the 10th International Conference on Computational Linguistics*, pages 123–129.

Pollard, C. J. (in press). Sorts in unification-based grammar and what they mean. In Pinkal, M. and Gregor, B., editors, *Unification in Natural Language Analysis.* MIT Press.

Pollard, C. J. and Carpenter, B. (to appear). Extensionality in Feature Structures and Feature Logic. Paper presented at the Workshop on Unification and Generation, Bad Teinach, Germany, November 1990. To appear in the Proceedings.

Pollard, C. J. and Moshier, M. D. (1990). Unifying partial descriptions of sets. In Hanson, P., editor, *Information, Language and Cognition*, volume 1 of *Vancouver Studies in Cognitive Science.* University of British Columbia Press, Vancouver.

Pollard, C. J. and Sag, I. A. (1987). *Information-Based Syntax and Semantics: Volume I – Fundamentals*, volume 13 of *CSLI Lecture Notes.* Chicago University Press, Chicago.

Rounds, W. C. and Kasper, R. T. (1986). A complete logical calculus for record structures representing linguistic information. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, Cambridge, Massachusetts.

Shieber, S. M. (1986). *An Introduction to Unification-Based Approaches to Grammar*, volume 4 of *CSLI Lecture Notes.* Chicago University Press, Chicago.

Shieber, S. M., Uszkoreit, H., Pereira, F. C. N., Robinson, J., and Tyson, M. (1983). The formalism and implementation of PATR-II. In *Research on Interactive Acquisition and Use of Knowledge*, volume 1894 of *SRI Final Report.* SRI International, Menlo Park, California.

Smolka, G. (1988). A feature logic with subsorts. LILOG-REPORT 33, IBM Deutschland GmbH, Stuttgart, FRG.

Walther, C. (1985). A mechanical solution of Schubert's Steamroller by many-sorted resolution. *Artificial Intelligence*, 26(2):217–224.

Walther, C. (1988). Many-sorted unification. *Journal of the ACM*, 35:1–17.