Parametric Types for Typed Attribute-Value Logic

Gerald Penn
Universität Tübingen
Kl. Wilhelmstr. 113
72074 Tuebingen
Germany
gpenn@sfs.nphil.uni-tuebingen.de

Abstract

Parametric polymorphism has been combined with inclusional polymorphism to provide natural type systems for Prolog (DH88), HiLog (YFS92), and constraint resolution languages (Smo89), and, in linguistics, by HPSG-like grammars to classify lists and sets of linguistic objects (PS94), and by phonologists in representations of hierarchical structure (Kle91). This paper summarizes the incorporation of parametric types into the typed attribute-value logic of (Car92), thus providing a natural extension to the type system for ALE (CP96). Following (Car92), the concern here is not with models of feature terms themselves, but with how to compute with parametric types, and what different kinds of information one can represent relative to a signature with parametric types, than relative to a signature without them. This enquiry has yielded a more flexible interpretation of parametric types with several specific properties necessary to conform to their current usage by linguists and implementors who work with feature-based formalisms.

1 Motivation

Linguists who avail themselves of attribute-value logic normally choose whether to encode information with subtypes or features on the aesthetic basis of what seems intuitively to capture their generalizations better. Linguists working in LFG typically use one implicit type for objects that bear features, and other types (atoms) for only featureless objects. In HPSG, the situation is less clear, both historically (semantic relations, for example, used to be values of a RELN attribute, and are now subtypes of a more general semantic type), and synchronically (verbs, for example, are identified as (un)inverted and (non-)auxiliaries by two boolean-valued features, AUX and INV, whereas their form, e.g., finite, infinitive, gerund, is identified by a subtype of a single *vform* type). That it makes, or at least should make, no difference from a formal or implementational point of view which encoding is used has been argued elsewhere (Mos96; Pen-f).

HPSG's type system also includes parametric types, e.g., Figure 1, from (PS94). In contrast

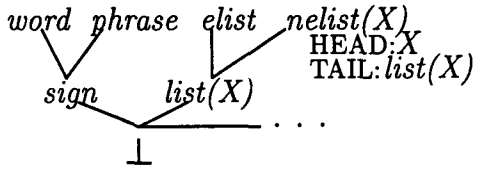

Figure 1: A fragment of the HPSG type signature.

to the relative expressive potential of normal typing and features, the expressive potential of parametric types is not at all understood. In fact, parametric types have never been formalized in a feature logic or in a manner general enough to capture their use in HPSG parsing so that a comparison could even be drawn. This paper summarizes such a formalization,[1] based on the typed attribute-value logic of (Car92). This logic is distinguished by its strong interpretation of *appropriateness*, a set of conditions that tell us which features an object of a given type can have, and which types a feature's value can have. Its interpretation, *total well-typedness*, says that every feature structure must have an appropriate value for all and only the appropriate features of its type. Previous approaches have required that every parameter of a subtype should be a parameter of all of its supertypes, and *vice versa*; thus, it would not be

[1]The full version of this paper presents a denotational semantics of the logic described here.

possible to encode Figure 1 because $\perp \sqsubseteq$ *list(X)*, and if $\perp$ were parametric, then all other types would be.[2] The present one eliminates this restriction (Section 2) by requiring the existence of a simple most general type (which (Car92)'s logic requires anyway), which is then used during type-checking and inferencing to interpret new parameters. All previous approaches deal only with fixed-arity terms; and none but one uses a feature logic, with the one, CUF (Dor92), being an implementation that permits parametric lists only as a special case. The present approach (Section 4) provides a generalization of appropriateness that permits both unrestricted parametricity and incremental feature introduction.

In contrast to the other encoding tradeoff, the use of parametric types in HPSG linguistics exhibits almost no variation. They are used almost exclusively for encoding lists (and, unconvincingly, sets), either with type arguments as they are posited in (PS94), or with general description-level arguments, e.g., *list*(LOCAL:CAT:HEAD:*verb*), the latter possibly arising out of the erroneous belief that parametric types are just "macro" descriptions for lists. Even in the former case, however, parametric types have as wide of a range of potential application to HPSG as simple types and features do; and there is no reason why they cannot be used as prolifically once they are understood. To use an earlier example, *auxiliary*, *inverted*, and *verb_form* could all be parameters of a parametric type, *verb*. In fact, parametrically typed encodings yield more compact specifications than simply typed encodings because they can encode products of information in their parameters, like features. Unlike features, however, they can lend their parameters to appropriateness restrictions, thus refining the feature structures generated by the signature to a closer approximation of what is actually required in the grammar theory itself.

It is possible, however, to regard parametric type *signatures*[3] as a shorthand for non-parametric *signatures*. The interpretation of parametric type hierarchies is introduced in Section 3 by way of establishing equivalent, infinite non-parametric counterparts. Section 5 considers whether there are any finite counterparts, i.e., whether in actual practice parametric signatures are only as expressive as non-parametric ones, and gives a qualified "yes."

In spite of this qualification, there is an easy way to compute with parametric types directly in an implementation, as described in Section 6. The two most common previous approaches have been to use the most general instance of a parametric type, e.g. *nelist*($\perp$) without its appropriateness, or manually to "unfold" a parametric type into a non-parametric sub-hierarchy that suffices for a fixed grammar (e.g. Figure 2). The former does not suffice even for fixed grammars because it simply disables type checking on feature values. The latter is error-prone, a nuisance, and subject to change with the grammar. As it happens, there is an automatic way to perform this unfolding.

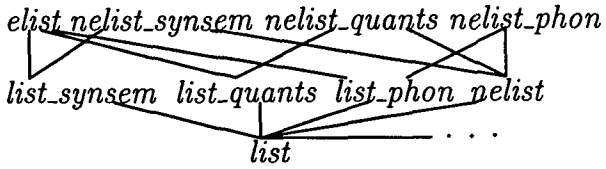

Figure 2: A manually unfolded sub-hierarchy.

2 Parametric Type Hierarchies

Parametric types are not types. They are functions that provide access or a means of reference to a set of types (their image) by means of argument types, or "parameters" (their domain). Figure 1 has only unary functions; but in general, parametric types can be n-ary functions over n-tuples of types.[4] This means that hier-

[2] In this paper, the most general type will be called $\perp$.

[3] By "signature," I refer to a partial order of types plus feature appropriateness declarations. The partial order itself, I shall refer to as a "type (inheritance) hierarchy."

[4] In this paper, "parametric type" will refer to such a function, written as the name of the function, followed by the appropriate number of "type variables," variables that range over some set of types, in parentheses, e.g. *list(X)*. "Type" will refer to both "simple types," such as $\perp$ or *elist*; and "ground instances" of parametric types, i.e. types in the image of a parametric type function, written as the name of the function followed by the appropriate number of actual type parameters in parentheses, such as *list*($\perp$), *set(psoa)* or *list(set*($\perp$)*)*. I will use letters t, u, and v to indicate types; capital letters to indicate type variables; capitalized words to indicate feature names; p, q, and r for names of parametric types; and g to indicate ground instances of parametric types,

archies that use parametric types are not "type" hierarchies, since they express a relationship between functions (we can regard simple types as nullary parametric types):

Definition 1: A *parametric (type) hierarchy* is a finite meet semilattice, $\langle P, \sqsubseteq_P \rangle$, plus a partial *argument assignment function*, $a_P : P \times P \times \mathsf{Nat} \rightarrow \mathsf{Nat} \cup \{0\}$, in which:

- P consists of (simple and) parametric types, (i.e. no ground instances of parametric types), including the simple most general type, $\bot$,
- For $p, q \in P$, $a_P(p, q, i)$, written $a_p^q(i)$, is defined iff $p \sqsubseteq_P q$ and $1 \leq i \leq arity(p)$, and
- $0 \leq a_p^q(i) \leq m$, when it exists.

Meet semilatticehood, a requirement of (Car92)'s logic as well, allows us to talk about unification, because we have a unique most-general unifier for every unifiable pair of types. The argument assignment function encodes the identification of parameters between a parametric type and its parametric subtype. The number, n, refers to the nth parameter of a parametric type, with 0 referring to a parameter that has been dropped. In practice, this is normally expressed by the names given to type variables. In the parametric type hierarchy of Figure 1, *list* and *nelist* share the same variable, X, because $a_{list}^{nelist}(1)$ is 1. If $a_{list}^{nelist}(1) = 0$, then *nelist* would use a different variable name. As a more complicated example, in Figure 3, $a_b^d(1) = 1$,

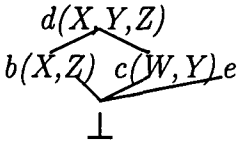

Figure 3: A subtype that inherits type variables from more than one supertype.

$a_b^d(2) = 3$, $a_c^d(2) = 2$, $a_c^d(1) = 0$, and $a_\bot$ and a_e are undefined ($\uparrow$) for any pair in $P \times \mathsf{Nat}$.

3 Induced Type Hierarchies

The relationship expressed between two functions by $\sqsubseteq_P$, informally, is one between their image sets under their domains,[5] while each image set internally preserves the subsumption ordering of its domain. It is, thus, possible to think of a parametric type hierarchy as "inducing" a non-parametric type hierarchy, populated with the ground instances of its parametric types, that obeys both of these relationships.

Definition 2: Given parametric type hierarchy, $\langle P, \sqsubseteq_P, a \rangle$, the *induced (type) hierarchy*, $\langle I(P), \sqsubseteq_I \rangle$, is defined such that:

- $I(P)$ is the smallest set, I, such that, for every parametric type, $p(X_1, \ldots, X_n) \in P$, and for every tuple, $\langle t_1 \ldots t_n \rangle \in I^n$, $p(t_1, \ldots, t_n) \in I$.
- $p(t_1, \ldots, t_n) \sqsubseteq_I q(u_1, \ldots, u_m)$ iff $p \sqsubseteq_P q$, and, for all $1 \leq i \leq n$, either $a_p^q(i) = 0$ or $t_i \sqsubseteq_I u_{a_p^q(i)}$.

It can easily be shown that $\langle I(P), \sqsubseteq_I \rangle$ is a partial order with a least element, namely $\bot$, the least element of P. Note that $I(P)$ also contains all of the simple types of P. In the case where $g_1 and g_2$ are simple, $g_1 \sqsubseteq_I g_2$ iff $g_1 \sqsubseteq_P g_2$.

Figure 4 shows a fragment of the type hierarchy induced by Figure 1. If *list* and *nelist* had

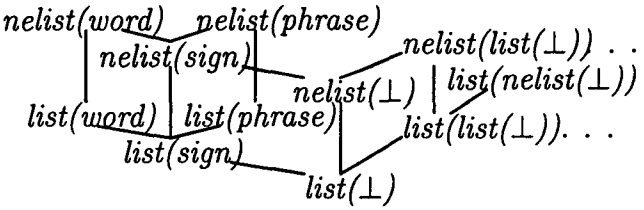

Figure 4: Fragment induced by Figure 1.

not shared the same type variable ($a_{list}^{nelist}(1) = 0$), then it would have induced the type hierarchy in Figure 5. In the hierarchy induced

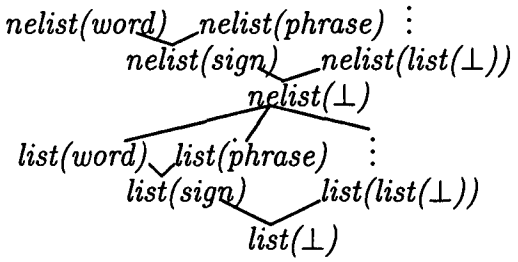

Figure 5: Another possible induced hierarchy.

where the arguments do not need to be expressed.

[5] One can restrict these domains with "parametric restrictions," a parallel to appropriateness restrictions on feature values. This abstract assumes that these domains are always the set of all types in the signature. This is the most expressive case of parametric types, and the worst case, computationally.

by Figure 3, *b(e,e)* subsumes types *d(e,Y,e)*, for any type Y, for example *d(e,c(e,e),e)*, or *d(e,b(⊥,e),e)*, but not *d(c(⊥,e),e,e)*, since $e \not\sqsubseteq_I c(\bot, e)$. Also, for any types, W, X, and Z, *c(W,e)* subsumes *d(X,e,Z)*.

The present approach permits parametric types in the type signature, but only ground instances in a grammar relative to that signature. If one must refer to "some list" or "every list" within a grammar, for instance, one may use $list(\bot)$, while still retaining groundedness. An alternative to this approach would be to attempt to cope with type variable parameters directly within descriptions. From a processing perspective, this is problematic when closing such descriptions under total well-typing, as observed in (Car92). The most general satisfier of the description, $list(X) \wedge$(HEAD:HEAD $\doteq$ TAIL:HEAD), for example, is an infinite feature structure of the infinitely parametric type, *nelist(nelist(...* because X must be bound to *nelist(X)*.

For which P does it make sense to talk about unification in $I(P)$, that is, when is $I(P)$ a meet semilattice? We can generalize the usual notion of *coherence* from programming languages, so that a subtype can add, and in certain cases drop, parameters with respect to a supertype:

Definition 3: $\langle P, \sqsubseteq_P, a_P \rangle$ is *semi-coherent* if, for all $p, q \in P$ such that $p \sqsubseteq_P q$, all $1 \leq i \leq arity(p)$, $1 \leq j \leq arity(q)$:

- $a_p^p(i) = i$,
- either $a_p^q(i) = 0$ or for every chain, $p = p_1 \sqsubseteq_P p_2 \sqsubseteq_P \ldots \sqsubseteq_P p_n = q$, $a_p^q(i) = a_{p_{n-1}}^{p_n}(a_{p_{n-2}}^{p_{n-1}}(\ldots a_{p_1}^{p_2}(i)\ldots))$, and
- If $p \sqcup_P q \downarrow$, then for all i and j for which there is a $k \geq 1$ such that $a_p^{p \sqcup_P q}(i) = a_q^{p \sqcup_P q}(j) = k$, the set, $\{r | p \sqcup_P q \sqsubseteq_P r$ and $(a_p^r(i) = 0$ or $a_q^r(j) = 0)\}$ is empty or has a least element (with respect to $\sqsubseteq_P$).

Theorem 1: If $\langle P, \sqsubseteq_P, a_P \rangle$ is semi-coherent, then $\langle I(P), \sqsubseteq_I \rangle$ is a meet semilattice. In particular, $p(t_1, \ldots, t_n) \sqcup_I q(u_1, \ldots, u_m) = r(v_1, \ldots, v_s)$, where $p \sqcup_P q = r$, and, for all $1 \leq k \leq s$,

$$v_k = \begin{cases} t_i \sqcup_I u_j & \text{if there exist } i \text{ and } j \text{ such that } a_p^r(i) = k \text{ and } a_q^r(j) = k \\ t_i & \text{if such an } i, \text{ but no such } j \\ u_j & \text{if such a } j, \text{ but no such } i \\ \bot & \text{if no such } i \text{ or } j. \end{cases}$$

So $p(t_1, \ldots, t_n) \sqcup_I q(u_1, \ldots, u_m)\uparrow$ if $p \sqcup_P q\uparrow$, or there exist i, j, and $k \geq 1$ such that $a_p^r(i) = a_q^r(j) = k$, but $t_i \sqcup_I u_j\uparrow$.[6]

In the induced hierarchy of Figure 3, for example, $b(e, \bot) \sqcup_I b(\bot, e) = b(e, e)$; $b(e, e) \sqcup_I c(\bot) = d(e, \bot, e)$; and $b(e, e)$ and $b(c(\bot), e)$ are not unifiable, as e and $c(\bot)$ are not unifiable. The first two conditions of semi-coherence ensure that a_P, taken as a relation between pairs of pairs of types and natural numbers, is an order induced by the order, $\sqsubseteq_P$, where it is not, taken as a function, zero. The third ensures that joins are preserved even when a parameter is dropped ($a_P = 0$). Note that joins in an induced hierarchy do not always correspond to joins in a parametric hierarchy. In those places where $a_P = 0$, types can unify without a corresponding unification in their parameters. Such is the case in Figure 5, where every instance of $list(X)$ ultimately subsumes $nelist(\bot)$. One may also note that induced hierarchies can have not only deep infinity, where there exist infinitely long subsumption chains, but broad infinity, where certain types can have infinite supertype (but never subtype) branching factors, as in the case of $nelist(\bot)$ or, in Figure 1, *elist*.

4 Appropriateness

So far, we have formally considered only type hierarchies, and no appropriateness. Appropriateness constitutes an integral part of a parametric type signature's expressive power, because the scope of its type variables extends to include it.

Definition 4: A *parametric (type) signature* is a parametric hierarchy, $\langle P, \sqsubseteq_P, a_P \rangle$, along with finite set of features, $Feat_P$, and a partial *(parametric) appropriateness function*, $Approp_P : Feat_P \times P \rightarrow Q$, where $Q = \bigcup_{n \in \mathsf{Nat}} Q_n$, and each Q_n is the smallest set satisfying the equation, $Q_n = \{1, \ldots, n\} \cup \{p(q_1, \ldots, q_k) | p \in P$ arity $k, q_i \in Q_n\}$, such that:

[6] The proofs of these theorems can be found in the full version of this paper.

1. (Feature Introduction) For every feature $f \in Feat_P$, there is a most general parametric type $Intro(f) \in P$ such that $Approp_P(f, Intro(f))$ is defined
2. (Upward Closure / Right Monotonicity) For any $p, q \in P$, if $Approp_P(f,p)$ is defined and $p \sqsubseteq_P q$, then $Approp_P(f,q)$ is also defined and $Approp_P(f,p) \sqsubseteq_Q Approp_P(f,q)$, where $\sqsubseteq_Q$ is defined as $\sqsubseteq_{I(P)}$ with natural numbers interpreted as universally quantified variables (e.g. $a(1) \sqsubseteq_Q b(1)$ iff $\forall x \in I(P).a(x) \sqsubseteq_{I(P)} b(x)$)
3. (Parameter Binding) For every $p \in P$ of arity n, for every $f \in Feat_P$, if $Approp_P(f,p)$ is defined, then $Approp_P(f,p) \in Q_n$.

$Approp_P$ maps a feature and the parametric type for which it is appropriate to its value restriction on that parametric type. The first two conditions are the usual conditions on (Car92)'s appropriateness. The third says that the natural numbers in its image refer, by position, to the parametric variables of the appropriate parametric type — we can use one of these parameters wherever we would normally use a type. Notice that ground instances of parametric types are permitted as value restrictions, as are instances of parametric types whose parameters are bound to these parametric variables, as are the parametric variables themselves. The first is used in HPSG for features such as SUBCAT, whose value must be $list(synsem)$; whereas the second and third are used in the appropriateness specification for $nelist(X)$ in Figure 1. The use of parameters in appropriateness restrictions is what conveys the impression that ground instances of lists or other parametric types are more related to their parameter types than just in name.

It is also what prevents us from treating instances of parametric types in descriptions as instantiations of macro descriptions. These putative "macros" would be, in many cases, equivalent only to infinite descriptions without such macros, and thus would extend the power of the description language beyond the limits of HPSG's own logic and model theory. Lists in HPSG would be one such case, moreover, as they place typing requirements on every element of lists of unbounded length. Ground instances of parametric types are also routinely used in appropriate value restrictions, whose extension to arbitrary descriptions would substantially extend the power of appropriateness as well. This alternative is considered further in the full version of this paper.

A parametric signature induces a type hierarchy as defined above, along with the appropriateness conditions on its ground instances, determined by the substitution of actual parameters for natural numbers. Thus:

Theorem 2: If $Approp_P$ satisfies properties (1)–(3) in Definition 4, then $Approp_{I(P)}$ satisfies properties (1) and (2).

5 Signature Subsumption

Now that parametric type signatures have been formalized, one can ask whether parametric types really add something to the expressive power of typed attribute-value logic. There are at least two ways in which to formalize that question:

Definition 5: Two type signatures, P and Q, are *equivalent* ($P \approx_S Q$) if there exists an order-isomorphism (w.r.t. subsumption) between the abstract totally well-typed feature structures of P and those of Q.

Abstract totally well-typed feature structures are the "information states" generated by signatures. Formally, as (Car92) shows, they can either be thought of as equivalence classes of feature structures modulo alphabetic variants, or as pairs of a type assignment function on feature paths and a path equivalence relation. In either case, they are effectively feature structures without their "nodes," which only bear information insofar as they have a type and serve as the focus of potential instances of *structure sharing* among feature path, where the traversal of two different paths from the same node leads to the same feature structure.

If, for every parametric signature P, there is a finite non-parametric N such that $P \approx_S N$, then parametric signatures add no expressive power at all — their feature structures are just those of some non-parametric signatures painted a different color. This is still an open question. There is, however, a weaker but still relevant reading:

Definition 6: Type signature, P, *subsumes* signature Q ($P \sqsubseteq_S Q$) if there exists an injection, f, from the abstract totally well-typed fea-

ture structures of P to those of Q, such that:

- if $F_1 \sqcup_{AT(P)} F_2\uparrow$, then $f(F_1) \sqcup_{AT(Q)} f(F_2)\uparrow$,
- otherwise, both exist and $f(F_1 \sqcup_{AT(P)} F_2) = f(F_1) \sqcup_{AT(Q)} f(F_2)$.

If for every parametric P, there is a finite non-parametric N such that $P \sqsubseteq_S N$, then it is possible to embed problems (specifically, unifications) that we wish to solve from P into N, solve them, and then map the answers back to P. In this reading, linguist users who want to think about their grammars with P must accept no non-parametric imitations because N may not have exactly the same structure of information states; but an implementor of a feature-based NLP system, for example, could secretly perform all of the work for those grammars in N, and no one would ever notice.

Under this reading, many parametrically typed encodings add no extra expressive power:
Definition 7: Parametric type hierarchy, $\langle P, \sqsubseteq_P, a_P \rangle$ is *persistent* if a_P never attains zero.
Theorem 3: For any persistent parametric signature, P, there is a finite non-parametric signature, N, such that $P \sqsubseteq_S N$.
If *elist* in Figure 1 retained *list(X)*'s parameter, then HPSG's type hierarchy (without sets) would be persistent. This is not an unreasonable change to make. The encoding, however, requires the use of *junk slots*, attributes with no empirical significance whose values serve as workspace to store intermediate results.

There are at least some non-persistent P, including the portion of HPSG's type hierarchy explicitly introduced in (PS94) (without sets), that subsume a finite non-parametric N; but the encodings are far worse. It can be proven, for example, that for any such P, some of its acyclic feature structures must be encoded by cyclic feature structures in N; and the encoding cannot be injective on the equivalence classes induced by the types of P, i.e. some type in N must encode the feature structures of more than one type from P. While parametric types may not be necessary for the grammar presented in (PS94) in the strict sense, their use in that grammar does roughly correspond to cases for which the alternative would be quite unappealing. Of course, parametric types are not the only extension that would ameliorate these encodings. The addition of relational expressions, functional uncertainty, or more powerful appropriateness restrictions can completely change the picture.

6 Finiteness

It would be ideal if, for the purposes of feature-based NLP, one could simply forget the encodings, unfold any parametric type signature into its induced signature at compile-time and then proceed as usual. This is not possible for systems that pre-compute all of their type operations, as the induced signature of any parametric signature with at least one non-simple type contains infinitely many types.[7] On the other hand, at least some pre-compilation of type information has proven to be an empirical necessity for efficient processing.[8] Given that one will only see finitely many ground instances of parametric types in any fixed theory, however, it is sufficient to perform some pre-compilation specific to those instances, which will involve some amount of unfolding. What is needed is a way of determining, given a signature and a grammar, what part of the induced hierarchy could be needed at run-time, so that type operations can be compiled only on that part.

One way to identify this part is to identify some set of ground instances (a *generator set*) that are necessary for computation, and close that set under $\sqcup_{I(P)}$:
Theorem 4: If $G \subseteq I(P)$, is finite, then the sub-algebra of $I(P)$ generated by G, $I(G)$, is finite.
$|I(G)|$ is exponential in $|G|$ in the worst case; but if the maximum *parametric depth* of G can be bounded (thus bounding $|G|$), then it is polynomial in $|P|$, although still exponential in the maximum arity of P:
Definition 8: Given a parametric hierarchy, P, the *parametric depth* of a type, $t = p(t_1, \ldots, t_n) \in I(P)$, $\delta(t)$, is 0 if $n = 0$, and $1 + \max_{1 \leq i \leq n} \delta(t_i)$ if $n > 0$.
So, for example, $\delta(\text{list}(\text{list}(\text{list}(\bot))))= 3$.
In practice, the maximum parametric depth should be quite low,[9] as should the maximum

[7] With parametric restrictions (fn. 5), this is not necessarily the case.

[8] Even in LFG, a sensible implementation will use *de facto* feature co-occurrence constraints to achieve much of the same effect.

[9] With lists, so far as I am aware, the potential demand has only reached $\delta = 2$ (MSI98) in the HPSG

arity. A standard closure algorithm can be used, although it should account for the commutativity and associativity of unification. One could also perform the closure lazily during processing to avoid a potentially exponential delay at compile-time. All of the work, however, can be performed at compile-time. One can easily construct a generator set: simply collect all ground instances of types attested in the grammar, or collect them and add all of the simple types, or add the simple types along with some extra set of types distinguished by the user at compile-time. The partial unfoldings like Figure 2 are essentially manual computations of $I(G)$.

Some alternatives to this approach are discussed in the full version of this paper. The benefit of this one is that, by definition, $I(G)$ is always closed under $\sqcup_{I(P)}$. In fact, $I(G)$ is the least set of types that is adequate for unification-based processing with the given grammar. Clearly, this method of sub-signature extraction can be used even in the absence of parametric types, and is a useful, general tool for large-scale grammar design and grammar reuse.

7 Conclusion

This paper presents a formal definition of parametric type hierarchies and signatures, extending (Car92)'s logic to the parametric case through equivalent induced non-parametric signatures. It also extends appropriateness to the common practice of giving the binding of parametric type variables scope over appropriate value restrictions.

Two formalizations of the notion of expressive equivalence for typed feature structures are also provided. While the question of $\approx_S$-equivalence remains to be solved, a weaker notion can be used to establish a practical result for understanding what parametric types actually contribute to the case of HPSG's type signature. A general method for generating sub-signatures is outlined, which, in the case of parametric type signatures, can be used to process with signatures that even have infinite equivalent induced signatures, avoiding equivalent encoding problems altogether.

Parametric type compilation is currently being implemented for ALE using the method given in Section 6.

literature to date.

References

(Car92) Carpenter, B., 1992. *The Logic of Typed Feature Structures.* Cambridge University Press.

(CP96) Carpenter, B., and Penn, G., 1996. Efficient Parsing of Compiled Typed Attribute Value Logic Grammars. In H. Bunt and M. Tomita, eds., *Recent Advances in Parsing Technology*, pp. 145–168. Kluwer.

(DH88) Dietrich, R. and Hagl, F., 1988. A Polymorphic Type System with Subtypes for Prolog. *Proceedings of the 2nd European Symposium on Programming*, pp. 79–93. Springer LNCS 300.

(Dor92) Dorna, M., 1992. *Erweiterung der Constraint-Logiksprache CUF um ein Typsystem.* Diplomarbeit, Universität Stuttgart.

(Kle91) Klein, E., 1991. Phonological Data Types. In S. Bird, ed., *Declarative Perspectives on Phonology*, pp. 127–138. Edinburgh Working Papers in Cognitive Science, 7.

(MSI98) Manning, C., Sag, I., and Iida, M., 1998. The Lexical Integrity of Japanese Causatives. To appear in G. Green and R. Levine eds., *Studies in Contemporary Phrase Structure Grammar.* Cambridge.

(Mos96) Moshier, M. A., 1995. Featureless HPSG. In P. Blackburn and M. de Rijke, eds., *Specifying Syntactic Structures.* CSLI Publications.

(Pen-f) Penn, G., forthcoming. Ph.D. Dissertation, Carnegie Mellon University.

(PS94) Pollard, C. and Sag, I., 1994. *Head-Driven Phrase Structure Grammar.* University of Chicago Press.

(Smo89) Smolka, G., 1989. Logic Programming over Polymorphically Order-Sorted Types. Ph.D. Dissertation, Universität Kaiserslautern.

(YFS92) Yardeni, E., Früwirth, T. and Shapiro, E., 1992. Polymorphically Typed Logic Programs. In F. Pfenning, ed., *Types in Logic Programming*, pp. 63–90. MIT Press.