Partially Specified Signatures: a Vehicle for Grammar Modularity

Yael Cohen-Sygal
Dept. of Computer Science
University of Haifa
yaelc@cs.haifa.ac.il

Shuly Wintner
Dept. of Computer Science
University of Haifa
shuly@cs.haifa.ac.il

Abstract

This work provides the essential foundations for modular construction of (typed) unification grammars for natural languages. Much of the information in such grammars is encoded in the signature, and hence the key is facilitating a modularized development of type signatures. We introduce a definition of signature modules and show how two modules combine. Our definitions are motivated by the actual needs of grammar developers obtained through a careful examination of large scale grammars. We show that our definitions meet these needs by conforming to a detailed set of desiderata.

1 Introduction

Development of large scale grammars for natural languages is an active area of research in human language technology. Such grammars are developed not only for purposes of theoretical linguistic research, but also for natural language applications such as machine translation, speech generation, etc. Wide-coverage grammars are being developed for various languages (Oepen et al., 2002; Hinrichs et al., 2004; Bender et al., 2005; King et al., 2005) in several theoretical frameworks, e.g., LFG (Dalrymple, 2001) and HPSG (Pollard and Sag, 1994).

Grammar development is a complex enterprise: it is not unusual for a single grammar to be developed by a team including several linguists, computational linguists and computer scientists. The scale of grammars is overwhelming: for example, the English resource grammar (Copestake and Flickinger, 2000) includes thousands of types. This raises problems reminiscent of those encountered in large-scale software development. Yet while software engineering provides adequate solutions for the programmer, no grammar development environment supports even the most basic needs, such as grammar modularization, combination of sub-grammars, separate compilation and automatic linkage of grammars, information encapsulation, etc.

This work provides the essential foundations for modular construction of signatures in typed unification grammars. After a review of some basic notions and a survey of related work we list a set of desiderata in section 4, which leads to a definition of signature modules in section 5. In section 6 we show how two modules are combined, outlining the mathematical properties of the combination (proofs are suppressed for lack of space). Extending the resulting module to a stand-alone type signature is the topic of section 7. We conclude with suggestions for future research.

2 Type signatures

We assume familiarity with theories of (typed) unification grammars, as formulated by, e.g., Carpenter (1992) and Penn (2000). The definitions in this section set the notation and recall basic notions. For a partial function F, '$F(x)\downarrow$' means that F is defined for the value x.

Definition 1 *Given a partially ordered set $\langle P, \leq \rangle$, the set of* **upper bounds** *of a subset $S \subseteq P$ is the set $S^u = \{y \in P \mid \forall x \in S \;\; x \leq y\}$.*

For a given partially ordered set $\langle P, \leq \rangle$, if $S \subseteq P$ has a least element then it is unique.

Definition 2 *A partially ordered set $\langle P, \leq \rangle$ is a* **bounded complete partial order** *(BCPO) if for every $S \subseteq P$ such that $S^u \neq \emptyset$, S^u has a least element, called a* **least upper bound (lub)**.

Definition 3 *A* **type signature** *is a structure $\langle \textsc{Type}, \sqsubseteq, \textsc{Feat}, Approp \rangle$, where:*

1. *$\langle \textsc{Type}, \sqsubseteq \rangle$ is a finite bounded complete partial order (the* **type hierarchy***)*

2. FEAT *is a finite set, disjoint from* TYPE.

3. $Approp$: TYPE×FEAT → TYPE *(the* ***appropriateness specification****) is a partial function such that for every* $F \in$ FEAT*:*

   (a) ***(Feature Introduction)*** *there exists a type* $Intro(F) \in$ TYPE *such that* $Approp(Intro(F), F)\downarrow$*, and for every* $t \in$ TYPE*, if* $Approp(t, F)\downarrow$*, then* $Intro(F) \sqsubseteq t$*;*

   (b) ***(Upward Closure)*** *if* $Approp(s, F)\downarrow$ *and* $s \sqsubseteq t$*, then* $Approp(t, F)\downarrow$ *and* $Approp(s, F) \sqsubseteq Approp(t, F)$*.*

Notice that every signature has a least type, since the subset $S = \emptyset$ of TYPE has the non-empty set of upper bounds, $S^u =$ TYPE, which must have a least element due to bounded completeness.

Definition 4 *Let* $\langle$TYPE$, \sqsubseteq\rangle$ *be a type hierarchy and let* $x, y \in$ TYPE*. If* $x \sqsubseteq y$*, then* x *is a* **supertype** *of* y *and* y *is a* **subtype** *of* x*. If* $x \sqsubseteq y$*,* $x \neq y$ *and there is no* z *such that* $x \sqsubseteq z \sqsubseteq y$ *and* $z \neq x, y$ *then* x *is an* **immediate supertype** *of* y *and* y *is an* **immediate subtype** *of* x*.*

3 Related Work

Several authors address the issue of grammar modularization in unification formalisms. Moshier (1997) views HPSG , and in particular its signature, as a collection of constraints over maps between sets. This allows the grammar writer to specify any partial information about the signature, and provides the needed mathematical and computational capabilities to integrate the information with the rest of the signature. However, this work does not define modules or module interaction. It does not address several basic issues such as bounded completeness of the partial order and the feature introduction and upward closure conditions of the appropriateness specification. Furthermore, Moshier (1997) shows how signatures are distributed into components, but not the conditions they are required to obey in order to assure the well-definedness of the combination.

Keselj (2001) presents a modular HPSG, where each module is an ordinary type signature, but each of the sets FEAT and TYPE is divided into two disjoint sets of private and public elements. In this solution, modules do not support specification of partial information; module combination is not associative; and the only channel of interaction between modules is the names of types.

Kaplan et al. (2002) introduce a system designed for building a grammar by both extending and restricting another grammar. An LFG grammar is presented to the system in a priority-ordered sequence of files where the grammar can include only one definition of an item of a given type (e.g., rule) with a particular name. Items in a higher priority file override lower priority items of the same type with the same name. The override convention makes it possible to add, delete or modify rules. However, a basis grammar is needed and when modifying a rule, the entire rule has to be rewritten even if the modifications are minor. The only interaction among files in this approach is overriding of information.

King et al. (2005) augment LFG with a makeshift signature to allow modular development of *untyped* unification grammars. In addition, they suggest that any development team should agree in advance on the feature space. This work emphasizes the observation that the modularization of the signature is the key for modular development of grammars. However, the proposed solution is adhoc and cannot be taken seriously as a concept of modularization. In particular, the suggestion for an agreement on the feature space undermines the essence of modular design.

Several works address the problem of modularity in other, related, formalisms. Candito (1996) introduces a description language for the trees of LTAG. Combining two descriptions is done by conjunction. To constrain undesired combinations, Candito (1996) uses a finite set of names where each node of a tree description is associated with a name. The only channel of interaction between two descriptions is the names of the nodes, which can be used only to allow identification but not to prevent it. To overcome these shortcomings, Crabbé and Duchier (2004) suggest to replace node naming by colors. Then, when unifying two trees, the colors can prevent or force the identification of nodes. Adapting this solution to type signatures would yield undesired order-dependence (see below).

4 Desiderata

To better understand the needs of grammar developers we carefully explored two existing grammars: the LINGO grammar matrix (Bender et al., 2002), which is a basis grammar for the rapid development of cross-linguistically consistent grammars; and a grammar of a fragment of Modern Hebrew, focusing on inverted constructions (Melnik, 2006). These grammars were chosen since they are comprehensive enough to reflect the kind of data large scale grammar encode, but are not too large to encumber this process. Motivated by these two grammars, we experimented with ways to divide the signatures of grammars into modules and with different methods of module interaction. This process resulted in the following desiderata for a beneficial solution for signature modularization:

1. The grammar designer should be provided with as much flexibility as possible. Modules should not be unnecessarily constrained.
2. Signature modules should provide means for specifying *partial* information about the components of a grammar.
3. A good solution should enable one module to refer to types defined in another. Moreover, it should enable the designer of module M_i to use a type defined in M_j without specifying the type explicitly. Rather, some of the attributes of the type can be (partially) specified, e.g., its immediate subtypes or its appropriateness conditions.
4. While modules can specify partial information, it must be possible to deterministically extend a module (which can be the result of the combination of several modules) into a full type signature.
5. Signature combination must be associative and commutative: the order in which modules are combined must not affect the result.

The solution we propose below satisfies these requirements.[1]

5 Partially specified signatures

We define *partially specified signatures (PSSs)*, also referred to as *modules* below, which are structures containing partial information about a signature: part of the subsumption relation and part of the appropriateness specification. We assume enumerable, disjoint sets TYPE of types and FEAT of features, over which signatures are defined. We begin, however, by defining *partially labeled graphs*, of which PSSs are a special case.

[1]The examples in the paper are inspired by actual grammars but are obviously much simplified.

Definition 5 *A **partially labeled graph (PLG)** over* TYPE *and* FEAT *is a finite, directed labeled graph $S = \langle Q, T, \preceq, Ap\rangle$, where:*

1. *Q is a finite, nonempty set of nodes, disjoint from* TYPE *and* FEAT*.*
2. *$T : Q \to$ TYPE is a partial function, marking some of the nodes with types.*
3. *$\preceq \subseteq Q \times Q$ is a relation specifying (immediate) subsumption.*
4. *$Ap \subseteq Q \times$ FEAT $\times Q$ is a relation specifying appropriateness.*

Definition 6 *A **partially specified signature (PSS)** over* TYPE *and* FEAT *is a PLG $S = \langle Q, T, \preceq, Ap\rangle$, where:*

1. *T is one to one.*
2. *'$\preceq$' is antireflexive; its reflexive-transitive closure, denoted '$\overset{*}{\preceq}$', is antisymmetric.*
3. (a) *(Relaxed Upward Closure) for all $q_1, q_1', q_2 \in Q$ and $F \in$ FEAT, if $(q_1, F, q_2) \in Ap$ and $q_1 \overset{*}{\preceq} q_1'$, then there exists $q_2' \in Q$ such that $q_2 \overset{*}{\preceq} q_2'$ and $(q_1', F, q_2') \in Ap$; and*
   (b) *(Maximality) for all $q_1, q_2 \in Q$ and $F \in$ FEAT, if $(q_1, F, q_2) \in Ap$ then for all $q_2' \in Q$ such that $q_2' \overset{*}{\preceq} q_2$ and $q_2 \neq q_2'$, $(q_1, F, q_2') \notin Ap$.*

A PSS is a finite directed graph whose nodes denote types and whose edges denote the subsumption and appropriateness relations. Nodes can be *marked* by types through the function T, but can also be *anonymous* (unmarked). Anonymous nodes facilitate reference, in one module, to types that are defined in another module. T is one-to-one since we assume that two marked nodes denote different types.

The '$\preceq$' relation specifies an immediate subsumption order over the nodes, with the intention that this order hold later for the types denoted by nodes. This is why '$\overset{*}{\preceq}$' is required to be a partial order. The type hierarchy of a type signature is a BCPO, but current approaches (Copestake, 2002) relax this requirement to allow more flexibility in grammar design. PSS subsumption is also a partial order but not necessarily a bounded complete one. After all modules are combined, the resulting subsumption relation will be extended to a BCPO (see section 7), but any intermediate result can be a general partial order. Relaxing the BCPO requirement also helps guaranteeing the associativity of module combination.

Consider now the appropriateness relation. In contrast to type signatures, Ap is not required to be a function. Rather, it is a relation which may specify *several* appropriate nodes for the values of a feature F at a node q. The intention is that the eventual value of $Approp(T(q), F)$ be the *lub* of the types of all those nodes q' such that $Ap(q, F, q')$. This relaxation allows more ways for modules to interact. We do restrict the Ap relation, however. Condition 3a enforces a relaxed version of upward closure. Condition 3b disallows redundant appropriateness arcs: if two nodes are appropriate for the same node and feature, then they should not be related by subsumption. The feature introduction condition of type signatures is not enforced by PSSs. This, again, results in more flexibility for the grammar designer; the condition is restored after all modules combine, see section 7.

Example 1 *A simple PSS S_1 is depicted in Figure 1, where solid arrows represent the '$\preceq$' (subsumption) relation and dashed arrows, labeled by features, the Ap relation. S_1 stipulates two subtypes of cat, n and v, with a common subtype, $gerund$. The feature* AGR *is appropriate for all three categories, with distinct (but anonymous) values for $Approp(n, \text{AGR})$ and $Approp(v, \text{AGR})$. $Approp(gerund, \text{AGR})$ will eventually be the lub of $Approp(n, \text{AGR})$ and $Approp(v, \text{AGR})$, hence the multiple outgoing* AGR *arcs from $gerund$.*

Observe that in S_1, '$\preceq$' is not a BCPO, Ap is not a function and the feature introduction condition does not hold.

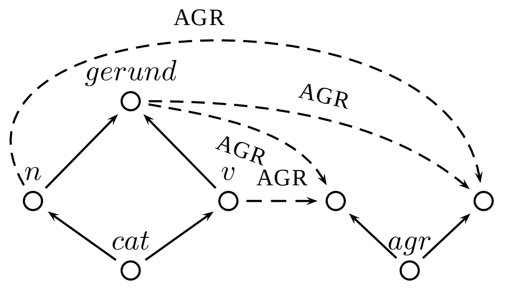

Figure 1: A partially specified signature, S_1

We impose an additional restriction on PSSs: a PSS is *well-formed* if any two different anonymous nodes are *distinguishable*, i.e., if each node is unique with respect to the information it encodes. If two nodes are indistinguishable then one of them can be removed without affecting the information encoded by the PSS. The existence of indistinguishable nodes in a PSS unnecessarily increases its size, resulting in inefficient processing.

Given a PSS S, it can be *compacted* into a PSS, $compact(S)$, by unifying all the indistinguishable nodes in S. $compact(S)$ encodes the same information as S but does not include indistinguishable nodes. Two nodes, only one of which is anonymous, can still be otherwise indistinguishable. Such nodes will, eventually, be coalesced, but only after all modules are combined (to ensure the associativity of module combination). The detailed computation of the compacted PSS is suppressed for lack of space.

Example 2 *Let S_2 be the PSS of Figure 2. S_2 includes two pairs of indistinguishable nodes: q_2, q_4 and q_6, q_7. The compacted PSS of S_2 is depicted in Figure 3. All nodes in $compact(S_2)$ are pairwise distinguishable.*

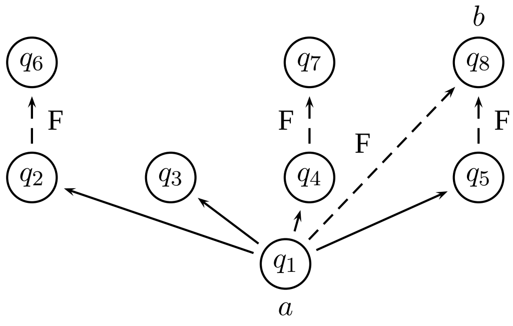

Figure 2: A partially specified signature with indistinguishable nodes, S_2

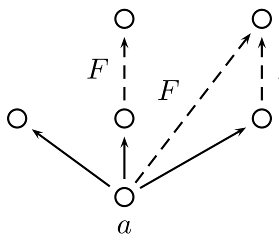

Figure 3: The compacted partially specified signature of S_2

Proposition 1 *If S is a PSS then $compact(S)$ is a well formed PSS.*

6 Module combination

We now describe how to combine modules, an operation we call *merge* bellow. When two modules are combined, nodes that are marked by the same type are coalesced along with their attributes. Nodes that are marked by different types cannot be coalesced and must denote different types. The main complication is caused when two *anonymous* nodes are considered: such nodes are coalesced only if they are indistinguishable.

The merge of two modules is performed in several stages: First, the two graphs are unioned (this is a simple pointwise union of the coordinates of the graph, see definition 7). Then the resulting graph is compacted, coalescing nodes marked by the same type as well as indistinguishable anonymous nodes. However, the resulting graph does not necessarily maintain the relaxed upward closure and maximality conditions, and therefore some modifications are needed. This is done by *Ap-Closure*, see definition 8. Finally, the addition of appropriateness arcs may turn two anonymous distinguishable nodes into indistinguishable ones and therefore another compactness operation is needed (definition 9).

Definition 7 *Let* $S_1 = \langle Q_1, T_1, \preceq_1, Ap_1 \rangle$, $S_2 = \langle Q_2, T_2, \preceq_2, Ap_2 \rangle$ *be two PLGssuch that* $Q_1 \cap Q_2 = \emptyset$. *The* **union** *of* S_1 *and* S_2*, denoted* $S_1 \cup S_2$*, is the PLG* $S = \langle Q_1 \cup Q_2, T_1 \cup T_2, \preceq_1 \cup \preceq_2, Ap_1 \cup Ap_2 \rangle$.

Definition 8 *Let* $S = \langle Q, T, \preceq, Ap \rangle$ *be a PLG. The* **Ap-Closure** *of* S*, denoted* $ApCl(S)$*, is the PLG* $\langle Q, T, \preceq, Ap'' \rangle$ *where:*

- $Ap' = \{(q_1, F, q_2) \mid q_1, q_2 \in Q$ *and there exists* $q_1' \in Q$ *such that* $q_1' \overset{*}{\preceq} q_1$ *and* $(q_1', F, q_2) \in Ap\}$
- $Ap'' = \{(q_1, F, q_2) \in Ap' \mid$ *for all* $q_2' \in Q$*, such that* $q_2 \overset{*}{\preceq} q_2'$ *and* $q_2 \neq q_2', (q_1, F, q_2') \notin Ap'\}$

Ap-Closure adds to a PLG the arcs required for it to maintain the relaxed upward closure and maximality conditions. First, arcs are added (Ap') to maintain upward closure (to create the relations between elements separated between the two modules and related by mutual elements). Then, redundant arcs are removed to maintain the maximality condition (the removed arcs may be added by Ap' but may also exist in Ap). Notice that $Ap \subseteq Ap'$ since for all $(q_1, F, q_2) \in Ap$, by choosing $q_1' = q_1$ it follows that $q_1' = q_1 \overset{*}{\preceq} q_1$ and $(q_1', F, q_2) = (q_1, F, q_2) \in Ap$ and hence $(q_1', F, q_2) = (q_1, F, q_2) \in Ap'$.

Two PSSs can be merged only if the resulting subsumption relation is indeed a partial order, where the only obstacle can be the antisymmetry of the resulting relation. The combination of the appropriateness relations, in contrast, cannot cause the merge operation to fail because any violation of the appropriateness conditions in PSSs can be deterministically resolved.

Definition 9 *Let* $S_1 = \langle Q_1, T_1, \preceq_1, Ap_1 \rangle$, $S_2 = \langle Q_2, T_2, \preceq_2, Ap_2 \rangle$ *be two PSSs such that* $Q_1 \cap Q_2 = \emptyset$. S_1, S_2 *are* ***mergeable*** *if there are no* $q_1, q_2 \in Q_1$ *and* $q_3, q_4 \in Q_2$ *such that the following hold:*

1. $T_1(q_1)\downarrow, T_1(q_2)\downarrow, T_2(q_3)\downarrow$ *and* $T_2(q_4)\downarrow$
2. $T_1(q_1) = T_2(q_4)$ *and* $T_1(q_2) = T_2(q_3)$
3. $q_1 \overset{*}{\preceq}_1 q_2$ *and* $q_3 \overset{*}{\preceq}_2 q_4$

If S_1 *and* S_2 *are mergeable, then their* ***merge****, denoted* $S_1 \Cup S_2$*, is* $compact(ApCl(compact(S_1 \cup S_2)))$.

In the merged module, pairs of nodes marked by the same type and pairs of indistinguishable anonymous nodes are coalesced. An anonymous node cannot be coalesced with a typed node, even if they are otherwise indistinguishable, since that will result in an unassociative combination operation. Anonymous nodes are assigned types only after all modules combine, see section 7.1.

If a node has multiple outgoing Ap-arcs labeled with the same feature, these arcs are not replaced by a single arc, even if the *lub* of the target nodes exists in the resulting PSS. Again, this is done to guarantee the associativity of the merge operation.

Example 3 *Figure 4 depicts a naïve agreement module,* S_5. *Combined with* S_1 *of Figure 1,* $S_1 \Cup S_5 = S_5 \Cup S_1 = S_6$*, where* S_6 *is depicted in Figure 5. All dashed arrows are labeled* AGR*, but these labels are suppressed for readability.*

Example 4 *Let* S_7 *and* S_8 *be the PSSs depicted in Figures 6 and 7, respectively. Then* $S_7 \Cup S_8 = S_8 \Cup S_7 = S_9$*, where* S_9 *is depicted in Figure 8. By standard convention,* Ap *arcs that can be inferred by upward closure are not depicted.*

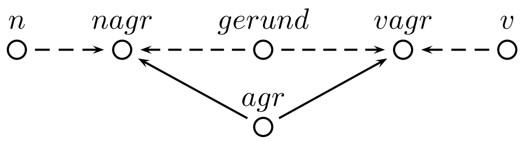

Figure 4: Naïve agreement module, S_5

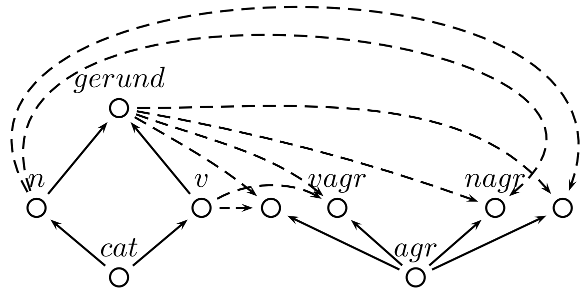

Figure 5: $S_6 = S_1 \Cup S_5$

Proposition 2 *Given two mergeable PSSs* S_1, S_2, $S_1 \Cup S_2$ *is a well formed PSS.*

Proposition 3 *PSS merge is commutative: for any two PSSs,* S_1, S_2, $S_1 \Cup S_2 = S_2 \Cup S_1$*. In particular, either both are defined or both are undefined.*

Proposition 4 *PSS merge is associative: for all* S_1, S_2, S_3, $(S_1 \Cup S_2) \Cup S_3 = S_1 \Cup (S_2 \Cup S_3)$.

7 Extending PSSs to type signatures

When developing large scale grammars, the signature can be distributed among several modules. A PSS encodes only partial information and therefore is not required to conform with all the constraints imposed on ordinary signatures. After all the modules are combined, however, the PSS must be extended into a signature. This process is done in 4 stages, each dealing with one property: 1. Name resolution: assigning types to anonymous nodes (section 7.1); 2. Determinizing Ap, converting it from a relation to a function (section 7.2); 3. Extending '$\preceq$' to a BCPO. This is done using the algorithm of Penn (2000); 4. Extending Ap to a full appropriateness specification by enforcing the feature introduction condition: Again, we use the algorithm of Penn (2000).

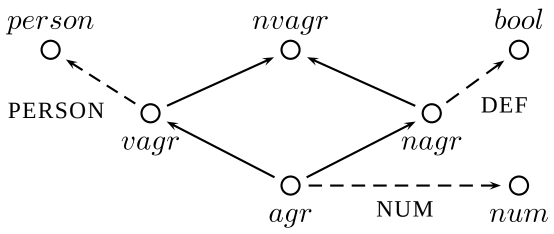

Figure 6: An agreement module, S_7

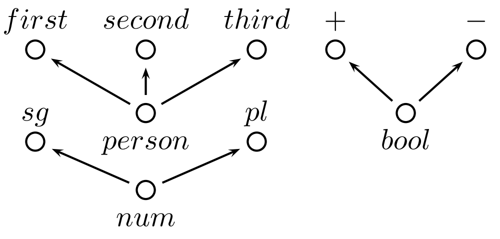

Figure 7: A partially specified signature, S_8

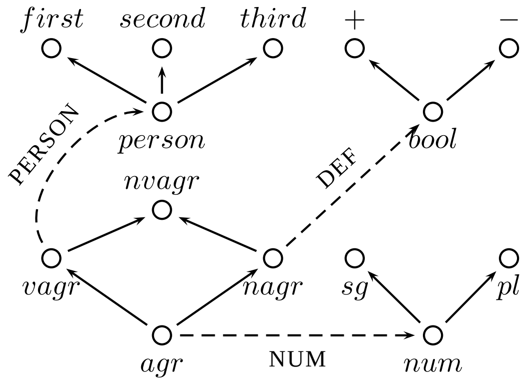

Figure 8: $S_9 = S_7 \Cup S_8$

7.1 Name resolution

By the definition of a well-formed PSS, each anonymous node is unique with respect to the information it encodes among the anonymous nodes, but there may exist a *marked* node encoding the same information. The goal of the name resolution procedure is to assign a type to every anonymous node, by coalescing it with a similar marked node, if one exists. If no such node exists, or if there is more than one such node, the anonymous node is given an arbitrary type.

The name resolution algorithm iterates as long as there are nodes to coalesce. In each iteration, for each anonymous node the set of its similar typed nodes is computed. Then, using this computation, anonymous nodes are coalesced with their paired similar typed node, if such a node uniquely exists. After coalescing all such pairs, the resulting PSS may be non well-formed and therefore the PSS is compacted. Compactness can trigger more pairs that need to be coalesced, and therefore the above procedure is repeated. When no pairs that need to be coalesced are left, the remaining anonymous nodes are assigned arbitrary names and the algorithm halts. The detailed algorithm is suppressed for lack of space.

Example 5 *Let S_6 be the PSS depicted in Figure 5. Executing the name resolution algorithm on this module results in the PSS of Figure 9 (*AGR*-labels are suppressed for readability.) The two anonymous nodes in S_6 are coalesced with the nodes marked $nagr$ and $vagr$, as per their attributes. Cf. Figure 1, in particular how two anonymous nodes in S_1 are assigned types from S_5 (Figure 4).*

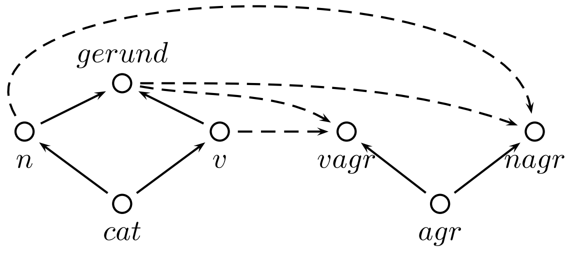

Figure 9: Name resolution result for S_6

7.2 Appropriateness consolidation

For each node q, the set of outgoing appropriateness arcs with the same label F, $\{(q, F, q')\}$, is replaced by the single arc (q, F, q_l), where q_l is marked by the *lub* of the types of all q'. If no *lub* exists, a new node is added and is marked by the *lub*. The result is that the appropriateness relation is a function, and upward closure is preserved; feature introduction is dealt with separately.

The input to the following procedure is a PSS whose typing function, T, is total; its output is a PSS whose typing function, T, is total and whose appropriateness relation is a function. Let $S = \langle Q, T, \preceq, Ap\rangle$ be a PSS. For each $q \in Q$ and $F \in$ FEAT, let

$$\begin{aligned} target(q, F) &= \{q' \mid (q, F, q') \in Ap\} \\ sup(q) &= \{q' \in Q \mid q' \preceq q\} \\ sub(q) &= \{q' \in Q \mid q \preceq q'\} \\ out(q) &= \{(F, q') \mid (q, F, q') \in Ap \end{aligned}$$

Algorithm 1 Appropriateness consolidation ($S = \langle Q, T, \preceq, Ap\rangle$)

```
1. Find a node $q$ and a feature $F$ for which $|target(q, F)| > 1$ and for all $q' \in Q$ such that $q' \overset{*}{\preceq} q$, $|target(q', F)| \leq 1$. If no such pair exists, halt.

2. If $target(q, F)$ has a lub, $p$, then:
    (a) for all $q' \in target(q, F)$, remove the arc $(q, F, q')$ from $Ap$.
    (b) add the arc $(q, F, p)$ to $Ap$.
    (c) for all $q' \in Q$ such that $q \overset{*}{\preceq} q'$, if $(q', F, p) \notin Ap$ then add $(q', F, p)$ to $Ap$.
    (d) go to (1).

3. (a) Add a new node, $p$, to $Q$ with:
        • $sup(p) = target(q, F)$
        • $sub(p) = (target(q, F))^u$
        • $out(p) = \bigcup_{q' \in target(q,F)} out(q')$
    (b) Mark $p$ with a fresh type from NAMES.
    (c) For all $q' \in Q$ such that $q \overset{*}{\preceq} q'$, add $(q', F, p)$ to $Ap$.
    (d) For all $q' \in target(q, F)$, remove the arc $(q, F, q')$ from $Ap$.
    (e) Add $(q, F, p)$ to $Ap$.
    (f) go to (1).
```

The order in which nodes are selected in step 1 of the algorithm is from supertypes to subtypes. This is done to preserve upward closure. In addition, when replacing a set of outgoing appropriateness arcs with the same label F, $\{(q, F, q')\}$, by a single arc (q, F, q_l), q_l is added as an appropriate value for F and all the subtypes of q. Again, this is done to preserve upward closure. If a new node is added (stage 3), then its appropriate features and values are inherited from its immediate supertypes. During the iterations of the algorithm, condition 3b (maximality) of the definition of a PSS may be violated but the resulting graph is guaranteed to be a PSS.

Example 6 *Consider the PSS depicted in Figure 9. Executing the appropriateness consolidation algorithm on this module results in the module depicted in Figure 10.* AGR*-labels are suppressed.*

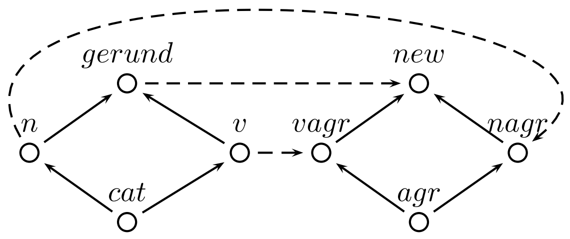

Figure 10: Appropriateness consolidation result

8 Conclusions

We advocate the use of PSSs as the correct concept of signature modules, supporting interaction among grammar modules. Unlike existing approaches, our solution is formally defined, mathematically proven and can be easily and efficiently implemented. Module combination is a commutative and associative operation which meets all the desiderata listed in section 4.

There is an obvious trade-off between flexibility and strong typedeness, and our definitions can be finely tuned to fit various points along this spectrum. In this paper we prefer flexibility, following Melnik (2005), but future work will investigate other options.

There are various other directions for future research. First, grammar *rules* can be distributed among modules in addition to the signature. The definition of modules can then be extended to include also parts of the grammar. Then, various combination operators can be defined for grammar modules (cf. Wintner (2002)). We are actively pursuing this line of research.

Finally, while this work is mainly theoretical, it has important practical implications. We would like to integrate our solutions in an existing environment for grammar development. An environment that supports modular construction of large scale grammars will greatly contribute to grammar development and will have a significant impact on practical implementations of grammatical formalisms.

9 Acknowledgments

We are grateful to Gerald Penn and Nissim Francez for their comments on an earlier version of this paper. This research was supported by The Israel Science Foundation (grant no. 136/01).

References

Emily M. Bender, Dan Flickinger, and Stephan Oepen. 2002. The grammar matrix: An open-source starter-kit for the rapid development of cross-linguistically consistent broad-coverage precision grammars. In *Proceedings of ACL Workshop on Grammar Engineering. Taipei, Taiwan*, pages 8–14.

Emily M. Bender, Dan Flickinger, Fredrik Fouvry, and Melanie Siegel. 2005. Shared representation in multilingual grammar engineering. *Research on Language and Computation*, 3:131–138.

Marie-Hélène Candito. 1996. A principle-based hierarchical representation of LTAGs. In *COLING-96*, pages 194–199, Copenhagen, Denemark.

Bob Carpenter. 1992. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Ann Copestake and Dan Flickinger. 2000. An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of LREC*, Athens, Greece.

Ann Copestake. 2002. *Implementing typed feature structures grammars*. CSLI publications, Stanford.

Benoit Crabbé and Denys Duchier. 2004. Metagrammar redux. In *CSLP*, Copenhagen, Denemark.

Mary Dalrymple. 2001. *Lexical Functional Grammar*, volume 34 of *Syntax and Semantics*. Academic Press.

Erhard W. Hinrichs, W. Detmar Meurers, and Shuly Wintner. 2004. Linguistic theory and grammar implementation. *Research on Language and Computation*, 2:155–163.

Ronald M. Kaplan, Tracy Holloway King, and John T. Maxwell. 2002. Adapting existing grammars: the XLE experience. In *COLING-02 workshop on Grammar engineering and evaluation*, pages 1–7, Morristown, NJ, USA.

Vlado Keselj. 2001. Modular HPSG. Technical Report CS-2001-05, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada.

Tracy Holloway King, Martin Forst, Jonas Kuhn, and Miriam Butt. 2005. The feature space in parallel grammar writing. *Research on Language and Computation*, 3:139–163.

Nurit Melnik. 2005. From "hand-written" to implemented HPSG theories. In *Proceedings of HPSG-2005, Lisbon, Portugal*.

Nurit Melnik. 2006. A constructional approach to verb-initial constructions in Modern Hebrew. *Cognitive Linguistics*, 17(2). To appear.

Andrew M. Moshier. 1997. Is HPSG featureless or unprincipled? *Linguistics and Philosophy*, 20(6):669–695.

Stephan Oepen, Daniel Flickinger, J. Tsujii, and Hans Uszkoreit, editors. 2002. *Collaborative Language Engineering: A Case Study in Efficient Grammar-Based Processing*. CSLI Publications, Stanford.

Gerald B. Penn. 2000. *The algebraic structure of attributed type signatures*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications.

Shuly Wintner. 2002. Modular context-free grammars. *Grammars*, 5(1):41–63.