A Context-Free Approximation of Head-Driven Phrase Structure Grammar

Bernd Kiefer and Hans-Ulrich Krieger
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
{kiefer,krieger}@dfki.de

Abstract

We present a context-free approximation of unification-based grammars, such as HPSG or PATR-II. The theoretical underpinning is established through a least fixpoint construction over a certain monotonic function. In order to reach a finite fixpoint, the concrete implementation can be parameterized in several ways, either by specifying a finite iteration depth, by using different restrictors, or by making the symbols of the CFG more complex adding annotations à la GPSG. We also present several methods that speed up the approximation process and help to limit the size of the resulting CF grammar.

1 Introduction

This paper presents a context-free approximation of unification-based grammars, such as HPSG or PATR-II. The basic idea of our approach is to generalize in a first step the set of all lexicon entries. The resulting structures form equivalence classes, since they abstract from word-specific information, such as FORM or STEM. The abstraction is specified by means of a restrictor (Shieber 1985), the so-called *lexicon restrictor*. The grammar rules are then instantiated via unification, using the abstracted lexicon entries and resulting in derivation trees of depth 1. We apply the *rule restrictor* to each resulting feature structure, which removes all information contained only in the daughters of the rule. Additionally, the restriction gets rid of information that will either lead to infinite growth of the feature structures during derivation or that does not restrict the search space. Why this is crucial is explained in section 4. The restricted feature structures then serve as the basis for the next instantiation step. Again, this gives us feature structures encoding a derivation, and again we are applying the rule restrictor. We proceed with the iteration, until we reach a fixpoint. In parallel, we also generate annotated context-free rules as have been made popular in linguistics by Harman 1963 and are a cornerstone in GPSG (Gazdar et al. 1985). The final result of this process is the set of CF productions.

Although several approaches have been proposed to limit the size of grammatical descriptions or to delay their applicability (cf. Shieber 1985, Kasper and Krieger 1996, Kiefer et al. 1999), it is practically exciting to extract grammars in a weaker formalism that either preserve the generated strings (weak equivalence) or that approximate them through a superset. Kasper et al. 1995 achieve a compilation of a large fragment of HPSG into lexicalized feature-based TAG. This paper even moves to a weaker formalism, viz., context-free grammars.

Approximating an HPSG through a CFG is interesting for the following reason. Assuming that we have a CFG that comes close to an HPSG, we can use the CFG as a cheap filter (running time

complexity is $O(n^3)$). The main idea is to use the CFG first and then let the HPSG deterministically replay the derivations of the CFG. The important point here is that one can correlate every CF production with the corresponding HPSG rule. Kasper et al. 1996 describe such an approach for word graph parsing which employs only the relatively unspecific CF backbone of an HPSG-like grammar. Diagne et al. 1995 replaces the CF backbone through a restriction of the original HPSG. This grammar, however, is still an unification-based grammar, since it employs coreference constraints.

The structure of the paper is as follows. In the next section, we recapitulate some basic HPSG terminology. We argue why it is hard to extract a restrictive CFG from a given HPSG due to lexicalization. We also introduce other concepts which we will need later: restrictors and root nodes. After that, section 3 present the theoretical foundation of our method: approximation as fixpoint construction of a certain monotonic function T. Section 4 presents the basic algorithm which exactly implements T. In order to overcome some deficiencies of the basic algorithm, both in terms of the running time and the resulting CFG, we describe useful optimizations in section 5. In section 6, we apply our method to four grammars and discuss the outcome of the approximation. We conclude this article by indicating our next steps.

2 Basic Inventory

Unification-based theories of grammar allow for an elegant integration of different levels of linguistic descriptions in the common framework of typed feature structures (see Carpenter 1992 for an introduction). In Head-Driven Phrase Structure Grammar this assumption is embodied in the fundamental concept of a *sign*; see Pollard and Sag 1994. A sign is a structure incorporating information from all levels of linguistic analysis, such as phonology, syntax, and semantics. This structure specifies interactions between these levels by means of coreferences, indicating the sharing of information. It also describes how the levels constrain each other mutually.

Such a concept of linguistic description is attractive for several reasons: 1. it supports the use of common formalisms and data structures on all linguistic levels, 2. it provides declarative and reversible interface specifications between these levels, 3. all information is simultaneously available, and 4. no procedural interaction between linguistic modules needs to be set up.

Similar approaches, especially for the syntax-semantics interface, have been suggested for all major kinds of unification-based theories, such as LFG or CUG. A crucial property of HPSG is that it is *lexicalized*, meaning that the rules (called rule schemata) are very general (and only a few of them exist) and that the lexicon entries (actually the lexical types) are relatively specific. HPSG shares this feature with other approaches, such as TAG and CG.

Let us now clarify some terminology which is used throughout this paper. In this paper, we interchange the terms *rules*, *rule schemata*, and *ID schemata* in HPSG. The set of all *rules* is depicted by $\mathcal{R}$. An example of a rule is the head-complement schema. When we say *lexicon entry*, we mean a feature structure that is subsumed by *word*. The collection of all *lexicon entries* constitutes the lexicon $\mathcal{L}$. $ar(r)$ denotes the number of daughters of a given rule $r \in \mathcal{R}$ and is called the *arity* of r.

We also need the notion of a *restrictor*, as originally introduced by Shieber 1985. A restrictor is an automaton describing the paths in a feature structure that will remain after *restriction* (the deletion operation). Since HPSG requires all relevant information to be contained in the `SYNSEM` feature of the mother structure, the unnecessary daughters only increase the size of the overall structure without constraining the search space. Due to the Locality Principle of HPSG, they can therefore be legally

removed in fully instantiated items (see Kiefer et al. 1999). Other portions of a feature structure will also be deleted during the iteration process. In section 4, we will describe why this is necessary. We employ two different restrictors in our approach: the *lexicon restrictor* L and the *rule restrictor* R.

Finally, we introduce the term *root node*. These are constraints specifying well-formedness conditions for other feature structures to be legal utterances/phrases (e.g., empty `SUBCAT` list). Root nodes are also feature structures and checking well-formedness is achieved via unification. The set of all root nodes is depicted by $\mathcal{N}$.

As we said in the introduction, we not merely produce CF rules employing flat (non-)terminal symbols, but instead enrich them with other information as is known from GPSG. This information directly comes from the feature structure and a user is requested to state this information in terms of an ordered set of paths $\mathcal{P}$, the so-called *relevant paths*. The values under these paths are exactly the annotations associated with the (non-)terminal symbols. Usually, one select those paths which restrict HPSG derivations the most, hence a resulting CF grammar becomes more restrictive. Kiefer et al. 1999 describe how such paths can be obtained with respect to a training corpus. This set of relevant paths is used both for the mother as well as for every daughter of an instantiated rule.

In the following, we make some minor simplifications to the original HPSG framework. (1) We do not distinguish between rules and principles. Instead, we assume that each rule has already inherited the principles which can be applied to it via unification. Such a strategy can often be found in implemented systems. (2) We do not pay attention to LP constraints as they are used in HPSG. The idea here is to hard-wire them in the rules, using the specialized attribute `ARGS` (see section 6 for a description). This technique is commonly used in many systems. (3) We do not take into account relational or set constraints, such as set membership or set difference. This information is clearly missing in the CFG. We note here that crucial relational constraints such as *append* as used in the subcategorization principle of HPSG must be expressed differently; in case of *append* either by employing the well-known *difference list* representation (good!) or by using a recursive type constraint à la Aït-Kaci. Again, many systems such as $\mathcal{TDL}$ or LKB use exactly the diff list representation. For instance, the English Verb*mobil* grammar (see section 6.4) as well as CSLI's LINGO grammar circumvent *append* in favor of diff lists.

3 Approximation as Fixpoint Construction

The basic inventory we have introduced in the last section now comes into play in order to formalize our idea as stated in the introduction. Let $\mathcal{L}$ be the set of all lexicon entries, $\mathcal{R}$ the set of all rules (rule schemata), L the lexicon restrictor, and R the rule restrictor. We can view a rule $r \in \mathcal{R}$, with $ar(r) = n$ as an n-ary function, mapping n-tuples of feature structures into a single feature structure:

$$r : \mathcal{FS}^n \longmapsto \mathcal{FS} \tag{1}$$

By instantiating only a single daughter, we obtain again a function (*currying*), i.e., $r(fs) = r'$ such that $ar(r') = n - 1$ ($fs \in \mathcal{FS}$). Similarly, we can view the lexicon and the rule restrictor as a (unary) function:

$$L/R : \mathcal{FS} \longmapsto \mathcal{FS} \tag{2}$$

During each iteration step i, we (inductively) obtain the following set T_i of feature structures:

$$T_0 := \bigcup_{l \in \mathcal{L}} L(l) \tag{3}$$

$$T_{i+1} := \bigcup_{n=0}^{i} \bigcup_{r \in \mathcal{R}} \bigcup_{t \in T_n^{ar(r)}} R(r(t)) \tag{4}$$

(3) corresponds to the generalization process of the lexicon, whereas (4) exactly describes the $i+1$ step of the iteration.

More formally in terms of the (upward) ordinal power of a monotonic mapping T (see Lloyd 1987), we define

$$T(S) := S \cup \bigcup_{r \in \mathcal{R}} \bigcup_{s \in S^{ar(r)}} R(r(s)) \tag{5}$$

Mathematically speaking, T itself operates on the power set of our domain of feature structures $\mathcal{FS}$, mapping approximations into finer approximations, i.e.,

$$T : \wp(\mathcal{FS}) \longmapsto \wp(\mathcal{FS}) \tag{6}$$

The idea here is that in the limit, T will exactly enumerate those feature structures which are instantiations of underspecified rule schemata and which have undergone the application of the rule restrictor R. Thus we are interested in sets of feature structures $S \in \wp(\mathcal{FS})$, where

$$T(S) = S \tag{7}$$

since in this case, S has been saturated, and so we can stop our iteration process via T. In this case, we call S a *fixpoint* of T.

Clearly, T is *monotonic*, since for all $S, S' \in \wp(\mathcal{FS})$ with $S \subseteq S'$, we have $T(S) \subseteq T(S')$. In order to show $T(S) \subseteq T(S')$, we use the definition of T:

$$S \cup \bigcup_{r \in \mathcal{R}} \bigcup_{s \in S^{ar(r)}} R(r(s)) \subseteq S' \cup \bigcup_{r \in \mathcal{R}} \bigcup_{s' \in S'^{ar(r)}} R(r(s'))$$

Since $S \subseteq S'$ (assumption), we have

$$\bigcup_{r \in \mathcal{R}} \bigcup_{s \in S^{ar(r)}} R(r(s)) \subseteq \bigcup_{r \in \mathcal{R}} \bigcup_{s' \in S'^{ar(r)}} R(r(s'))$$

Because $S \subseteq S'$, there must exist an S'' s.t. $S \cup S'' = S'$, hence we finally have for every $r \in \mathcal{R}$

$$\bigcup_{s \in S^{ar(r)}} R(r(s)) \subseteq \bigcup_{s \in S^{ar(r)}} R(r(s)) \cup \bigcup_{s'' \in S''^{ar(r)}} R(r(s'')) \subseteq \bigcup_{t \in (S \cup S'')^{ar(r)}} R(r(t)) = \bigcup_{s' \in S'^{ar(r)}} R(r(s'))$$

The approximated context-free grammar $\mathcal{G}$ then is defined as the *least fixpoint* of T:

$$\mathcal{G} := lfp(T) \tag{8}$$

Alternatively, one can define a fixpoint over the sets of annotated CF productions $\langle P_i \rangle_{i \geq 0}$ which are created in parallel with the construction of $\langle T_i \rangle_{i \geq 0}$. Since elements from T_i are much more specific than from P_i, (7) should of course guarantee a much more specific CF grammar. From a practical

point of view, however, terminating the iteration process when $P_i = P_{i+1}$ might suffice in most cases, due to the following observation. When we chose a wrong restrictor R during our experiments, we often experimented a dramatic increase of T_i, but a standstill of P_i. The reason for this were growing feature values in elements from T_i which however do not occur as annotations in rules from P_i.

It should now be clear that a finite fixpoint can only be reached if such growing values are eliminated. Furthermore, the more features are deleted by the restrictor, the more general the CFG will be and the sooner the fixpoint will be reached. Since the approximated CFG is defined as the least fixpoint of T, a pre-specified finite iteration depth usually does neither lead to a subset nor a superset of the original HPSG. But even a pre-specified finite iteration depth together with a restrictor that only deletes the daughters makes perfect sense, since in this case the CFG should generate the same sentences as the HPSG up to a certain number of embeddings.

4 The Basic Algorithm

We will now describe the naïve approximation algorithm which can be seen as a direct implementation of the definitional equations (3) and (4), together with the fixpoint termination criterion given by (7). The construction of (new) annotated context-free rules from successful instantiations is done in parallel with the computation of T_i during iteration step i. Several optimizations are discussed later in section 5 that improve the efficiency of the algorithm, but are uninteresting at this stage of presentation.

We start with the description of the top-level function *HPSG2CFG* which initiates the approximation process. In addition to the parameters already presented, we also use a global variable n, the iteration depth, and a local variable P that accumulates annotated context-free rules.

We begin the approximation by first abstracting from the lexicon entries $\mathcal{L}$ with the help of the lexicon restrictor L (line 6 of the algorithm). This constitutes our initial set T_0 (line 7). We note here that an abstracted lexicon entry is not merely added using set union but instead through a more complex operation, depicted by $\cup_{\sqsubseteq}$. We will describe this operation in section 5.

In case that an abstracted entry l is compatible with one of the root nodes from $\mathcal{N}$ (line 8), a start production is generated from l (employing the relevant paths $\mathcal{P}$) and added to the set of productions P (line 9). Again, adding a production does not reduce to a simple set union, but to a more sophisticated operation. We will describe $\cup_{\rightarrow}$ in section 5. Finally, we start the true iteration calling *Iterate* with the necessary parameters.

1 *HPSG2CFG*$(\mathcal{R}, \mathcal{L}, \mathcal{N}, \mathcal{P}, R, L, n) :\Longleftrightarrow$
2 **local** T_0, P;
3 $T_0 := \emptyset$;
4 $P := \emptyset$;
5 **for each** $l \in \mathcal{L}$
6 $l := L(l)$;
7 $T_0 := T_0 \cup_{\sqsubseteq} \{l\}$;
8 **when** *Is-Saturated*$(l, \mathcal{N})$
9 $P := P \cup_{\rightarrow} \{S \rightarrow$ *Make-Symbol*$(l, \mathcal{P})\}$;
10 *Iterate*$(\mathcal{R}, \mathcal{P}, \mathcal{N}, R, T_0, P, n)$.

Iterate first checks whether we have reached a pre-defined iteration depth n specified originally in *HPSG2CFG* (line 13 of the algorithm). If this is the case, we immediately return the annotated CF

rules computed so far (line 14). Otherwise, we decrease the depth, since we are going to start a new iteration (line 15). Usually the iteration depth is ∞, hence line 13 will never become true.

After that the instantiation of the rule schemata with rule/lexicon-restricted elements from the previous iteration T_i begins (line 17–23). The instantiation is performed by *Fill-Daughters* which takes into account a single rule r and T_i, returning successful instantiations (line 18). It is explained in section 5.

For each successful instantiation t which represents a feature structure tree of depth 1, we create an annotated CF rule using *Make-Production*. As was the case for *Make-Symbol*, *Make-Production* uses the relevant paths $\mathcal{P}$ for extracting the right annotations (line 19). It also inspects the daughters of t to compute the proper right-hand side of the new CF rule. Again, we are using the special set union operation $\cup_{\rightarrow}$ here.

After this, we apply the rule restrictor to t, since at this point we are allowed to delete the daughters (line 20). Furthermore, the rule restrictor should be carefully tuned (if possible) to get rid of paths whose values would otherwise grow infinitely through the approximation. Critical areas include the semantics (roughly speaking: information under feature `CONTENT` in HPSG) which is collected through a derivation but does not shrink as is usually the case for valence information. Now to guarantee a *finite* fixpoint (and we are interested in this), such information must be deleted in order to enforce a moderate growth of T_{i+1}. We then add the restricted t to T_{i+1} (line 21) and check whether it is saturated as specified by one of the root nodes from $\mathcal{N}$; if so, we generate a start production for t (lines 22–23).

We finally come to the point where we compare the (restricted) feature structures from the previous iteration T_i with the new ones from T_{i+1}. If both sets contain the same elements, we clearly have reached a fixpoint. In this case we immediately terminate with the set of annotated CF rules P; otherwise, we proceed with the iteration (lines 24–26).

11 $\mathit{Iterate}(\mathcal{R},\mathcal{P},\mathcal{N},R,T_i,P,n) :\Longleftrightarrow$
12 **local** T_{i+1};
13 **when** $n = 0$
14 **return** P;
15 $n := n - 1$;
16 $T_{i+1} := T_i$;
17 **for each** $r \in \mathcal{R}$
18 **for each** $t \in \mathit{Fill\text{-}Daughters}(r, T_i)$ **do**
19 $P := P \cup_{\rightarrow} \{\mathit{Make\text{-}Production}(t,\mathcal{P})\}$;
20 $t := R(t)$;
21 $T_{i+1} := T_{i+1} \cup_{\sqsubseteq} \{t\}$;
22 **when** $\mathit{Is\text{-}Saturated}(t,\mathcal{N})$
23 $P := P \cup_{\rightarrow} \{S \rightarrow \mathit{Make\text{-}Symbol}(t,\mathcal{P})\}$;
24 **if** $T_i = T_{i+1}$
25 **then return** P
26 **else** $\mathit{Iterate}(\mathcal{R},\mathcal{P},\mathcal{N},R,T_{i+1},P,n)$.

5 Implementation Issues and Optimizations

In this section we describe several techniques that speed up the iteration and that help to reduce the size of the resulting annotated context-free grammar.

5.1 Speeding Up the Algorithm

In order to make the basic algorithm work for large-size grammars, we modified it in several ways. The most obvious optimization applies to the function *Fill-Daughters*, where the number of unifications is reduced by avoiding re-computation of combinations of daughters and rules that already have been checked. To do this in a simple way, we split T_i into $T_i \setminus T_{i-1}$ and T_{i-1} and fill a rule with only those permutations of daughters which contain at least one element of $T_i \setminus T_{i-1}$. This guarantees checking of only those configurations which were enabled by the last iteration.

To further reduce the number of unifications, the algorithm could keep partly-filled rules which may be re-used in later iterations. Whether this pays off depends on the number of these items and the cost for copying and memory management vs. the cost for unification.

We also use techniques originally developed to speed up parsing, namely the so-called *rule filter* and the *quick-check method* (see Kiefer et al. 1999 for a detailed description of these techniques). The rule filter precomputes the applicability of rules into each other and thus is able to predict a failing unification using simple and fast table lookup. The quick-check method exploits the fact that unification fails more often at certain points in feature structures than at others. In an off-line stage, we parse a test corpus using a special unifier that records all failures instead of bailing out after the first in order to determine the most prominent failure points. These points constitute the quick-check vector. When executing a unification during approximation, those points are efficiently accessed and checked prior to the rest of the structure.

As it was mentioned in section 4, instead of using set union we use the more elaborate operation $\cup_{\sqsubseteq}$ when adding new feature structure nodes to T_{i+1}. In fact, we add a new node only if it is not subsumed by some node already in the set. To do this efficiently, the quick-check vectors described above are employed here: before performing full feature structure subsumption, we pairwise check the elements of the vectors using type subsumption and only if this succeeds do a full subsumption test. Extending feature structure subsumption by quick-check subsumption definitely pays of: between 95–99% of all failing subsumptions can be detected early. If we add a new node, we also remove all those nodes in T_{i+1} that are subsumed by the new node in order to keep the set small. This does not change the language of the resulting CF grammar because a more general node can be put into at least those daughter positions which can be filled by the more specific one. Consequently, for each production that employs the more specific node, there will be a (possibly) more general production employing the more general node in the same daughter positions.

A further, although orthogonal method to speed up the subsumption check is to compute a hash key from a quick-check vector that preselects sets of feature structures which are likely to be subsumed. I.e., the quick-check vector here serves as a means to partition feature structures from T_{i+1}. The collection of all vectors form a hierarchy (DAG) which is incrementally build up during the iteration process.

The last two methods accelerate different aspects of the subsumption check: the former technique quickly rejects failing subsumptions, whereas the latter determines likely-subsuming structures, i.e., it is worth to have them both.

5.2 Reducing CF Grammar Size

Subsumption can be extended to annotated symbols and CF productions in an obvious way, namely as componentwise subsumption on types and (non-)terminal symbols. We can then remove all those productions which are subsumed by another one from the set of productions P. We indicated this in the algorithm in section 4 by using the special operation $\cup_{\rightarrow}$. Since applicability of the more general production implies the applicability of the specialized one, the language induced by the reduced grammar remains the same.

Rule folding is another method that can drastically decrease the number of CF productions. Assume that we find a set of annotated CF rules that only differ in one slot. Assume further that the set of values for the slot is equal to the set of all possible values for this slot. Hence we can replace these rules by a single rule in which the slot is assigned the most general value. Rule folding also leads to a grammar inducing the same language. Clearly, the analogue to rule folding can also be implemented in the feature structure domain, but is much more complicated.

Since the annotated CF grammar is still a CFG, we can furthermore remove those symbols which will never contribute to a reading. These symbols and the productions using them are called *useless* and there exist fast decidable algorithms that given a grammar produce a weakly equivalent one which does not contain useless productions (e.g., Hopcroft and Ullman 1979, pp. 87).

5.3 Computing the Productions Afterwards

Although the algorithm from section 4 produces annotated CF productions in parallel with instantiated rule schemata (lines 9, 19, and 23), it might be more efficient to generate them afterwards when reaching the fixpoint (or a predefined iteration depth). Since (5) tells us that (more general) elements from iteration step i are always present in step j, $j \geq i$ (remember, T is monotonic), it suffices to generate CF productions simply by instantiating the rule schemata a final time with elements from the last iteration step . Such a strategy can be more efficient than the parallel generation, due to the fact that we do not have to check for rule subsumption and rule folding (recall that elements in T_i are pairwise incomparable).

6 Examples and Results

In our system, grammar rules are encoded as pure feature structures using a special list-valued feature `ARGS` to define the right-hand side, while the other top-level features belong to the left-hand side. The rules given in the following figures use this encoding too. During the iteration process, this `ARGS` feature is the only feature that we remove with the rule restrictor in the next three examples. Otherwise, feature structures would grow in every iteration, due to the rising depth of rule applications and consequently the derivation structure. The fourth example describes the transformation of the large English HPSG that is used in Verb*mobil*.

6.1 A Grammar for $a^n b^n$

The first grammar licenses the language $\{a^n b^n \mid n > 0\}$ and is shown together with the final result of the approximation algorithm (reached after four iterations) in figure 1. The set of relevant paths $\mathcal{P}$ is simply given by the `CAT` feature.

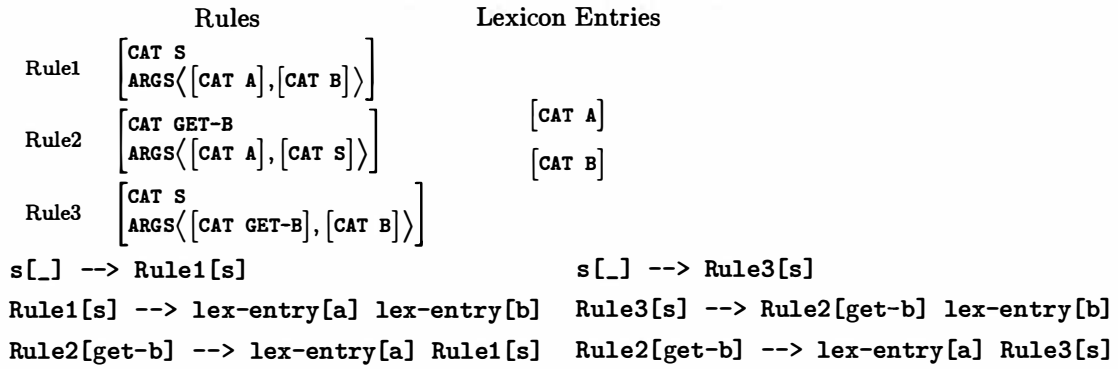

Figure 1: Example grammar 1. `s[_]` denotes the start symbol.

6.2 A Grammar With Coreferences

Our second example employs coreferences to show the variation of the `CAT` feature in the annotated CF rules between the mother and the two daughters. After three iterations, the algorithm terminates with six feature structure nodes and 18 context-free productions (see figure 2).

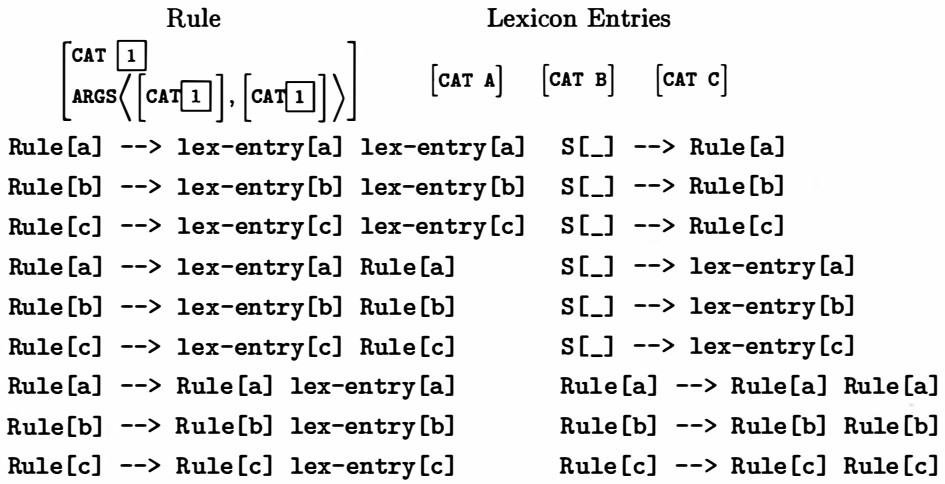

Figure 2: Coreference variation in grammar 2.

6.3 Shieber's Toy Grammar

The third example is the feature structure equivalent of Shieber's second sample grammar Shieber 1986, pp. 71–76. This grammar uses two underspecified rules for verb phrase construction and is therefore a test case much more in the direction of HPSG than the first two examples (see figure 3). Overall, the grammar consists of three rules, together with 13 lexicon entries. Two of the lexicon entries can be collapsed into a single category.

We made two tests with different path sets, one containing only agreement, the other containing also subcategorization information. After four iterations, both processes stopped with 31 feature structure nodes ($= T_5$). Without subcategorization information, we got 20 context-free productions. However by using also the type and agreement information of subcategorized elements, we obtained 33 productions. It should be noted that only the second set of productions correctly distinguishes

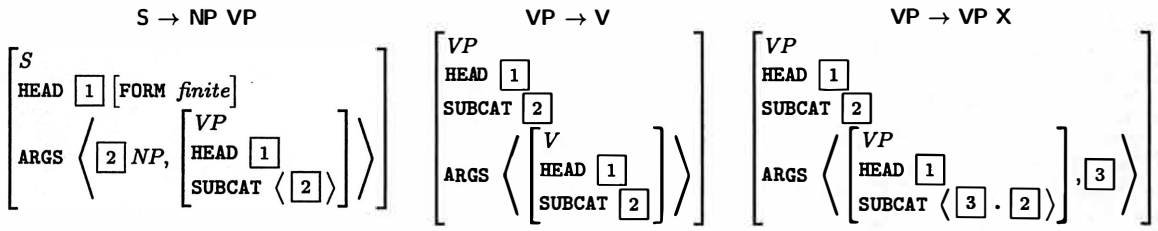

Figure 3: Underspecified rules in Shieber's grammar.

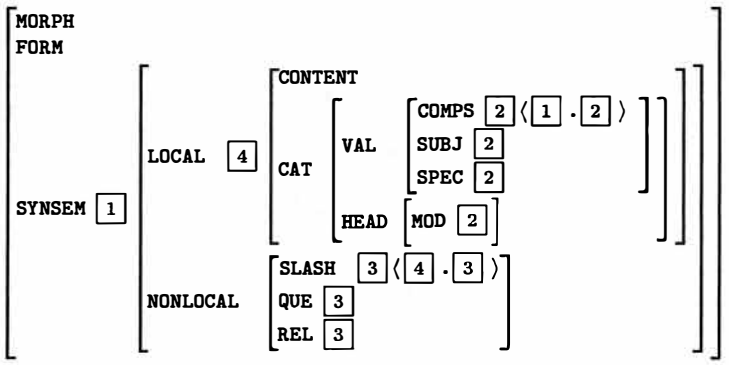

Figure 4: The (slightly simplified) lexicon restrictor used in our experiments.

VP nonterminals subcategorizing for NPs with specific agreement values. Clearly, a more specific CF grammar results in more productions plus more nonterminal symbols.

6.4 CSLI's English Verb*mobil* Grammar

Our final example employs the English Verb*mobil* grammar, developed at CSLI, Stanford. The grammar consists of 42 rule schemata, 7,473 types, and a lexicon of 4,919 stems from which 10,967 full forms are derived. The lexicon restrictor for the English grammar as shown in figure 4 maps these entries onto 422 lexical abstractions. This restrictor tells us which parts of a feature structure have to be deleted—this is the kind of restrictor which we are usually going to use. We call this a *negative* restrictor, contrary to the *positive* restrictors used in the PATR-II system that specify those parts of a feature structure which will survive after restricting it. Since a restrictor could have reentrance points, one can even define a *recursive* (or cyclic) restrictor to foresee recursive embeddings as is the case in HPSG. The rule restrictor looks quite similar, cutting off additional feature-value pairs, such as DTRS/ARGS.

We made four experiments, using 5, 10, 29, and finally the full 42 rule schemata of the English grammar. Finite fixpoints were reached after at most 12 iteration steps; see figure 5. As we discussed in the last section, it might be the case that more specific feature structures were removed in favor of more general ones. Exactly this happened during the iterations (figure 5). For example, during the approximation of the full grammar, we obtained 8,388 feature structures after iteration step 3, but ended up with 5,522 in step 12. Since the full grammar licensed more constructions than the other three, we guessed that a more general grammar could perhaps result in even less feature structures, due to the generalization process. Indeed, this has happened when comparing the full grammar (5,522) with the grammar consisting of only 29 rules (5,664).

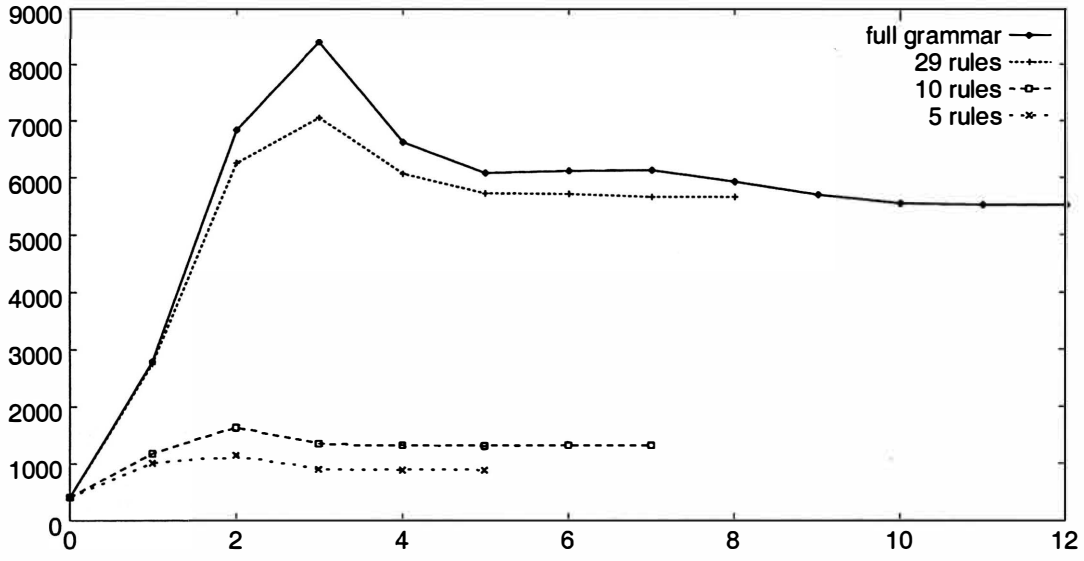

Figure 5: Number of feature structures computed for the four grammars, using the restrictor from figure 4. Finite fixpoints were reached after 5, 7, 8, and 12 iteration steps, resp.

The computation of the fixpoint for the full grammar took about 45 CPU hours on a 300MHz SUN Ultrasparc 2 with Solaris 2.5/Franz Allegro Common Lisp. During the iteration, 22,834,257 full top-level unifications and 71,666,481 subsumptions were performed. The quick-check method, used both during unification and subsumption, leads to a filter rate of 96.1% and 98.8%, resp. This translates into an enormous number of 341,906,443 and 5,018,258,054 quickly failing unification and subsumption operations.

The full grammar consists of 42 rules (14 unary and 28 binary rules). As we indicated in section 5.3, it is possible to compute the CF productions after reaching the fixpoint, viewing full feature structures from the fixpoint as complex CF symbols (one can assign a unique integer to every feature structure). We did this and obtained the following results (a full discussion, including an evaluation of the CF grammar can be found in Kiefer and Krieger 2000). Given the 5,522 feature structures from the fixpoint which are exactly the (non-)terminal symbols of the CF grammar, the 42 rules might lead to $14 \times 5,522 + 28 \times 5,522 \times 5,522 = 853,866,860$ CF productions in the worst case. Our method produces 2,157,445 productions, only 0.25% of all possible ones.

7 Summary

We have presented a context-free approximation of unification-based grammars, such as HPSG or PATR-II. The application of our method to the large-scale English Verb*mobil* grammar has been finished with promising results. We plan to evaluate our method on the German and Japanese HPSGs used in Verb*mobil*. The idea also seems to work with minor modifications for approximating HPSG by Tree Substitution Grammars, TAGs, or other kinds of lexicalized grammars. The substantial difference here comes from the interpretation of the restrictor and the function *Fill-Daughters*. This is currently under evaluation. We also plan to compare the outcome of our approach with the results reported in Torisawa et al. 2000 who also achieved a conversion of the LINGO grammar into a CFG. In section 5.1, we outlined two orthogonal techniques that speed up different aspects of feature

structure subsumption, viz., quickly rejecting failing subsumptions and quickly determining likely-subsuming feature structures. These techniques might be worthwhile in other cases where subsumption is a time-critical operation (e.g., during packing of feature structures).

Acknowledgments

We would like to thank our colleagues Tilman Becker, Dan Flickinger, Günter Neumann, and Stephan Oepen for fruitful discussions and two of our reviewers for moral support. Especially Stephan has helped us a lot with the English grammar. The paper has also benefited from a former collaboration with John Carroll and Rob Malouf. Thanks guys. This research was supported by the German Federal Ministry for Education, Science, Research and Technology under grant no. 01 IV 701 V0.

References

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

Diagne, A. K., W. Kasper, and H.-U. Krieger. 1995. Distributed Parsing With HPSG Grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies, IWPT-95*, 79–86.

Gazdar, G., E. Klein, G. Pullum, and I. Sag. 1985. *Generalized Phrase Structure Grammar.* Cambridge, MA: Harvard University Press.

Harman, G. 1963. Generative Grammars Without Transformation Rules: A Defence of Phrase Structure. *Language* 39:597–616.

Hopcroft, J. E., and J. D. Ullman. 1979. *Introduction to Automata Theory, Languages, and Computation.* Reading, MA: Addison-Wesley.

Kasper, R. T., B. Kiefer, K. Netter, and K. Vijay-Shanker. 1995. Compilation of HPSG to TAG. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, ACL-95*, 92–99.

Kasper, W., and H.-U. Krieger. 1996. Modularizing Codescriptive Grammars for Efficient Parsing. In *Proceedings of the 16th International Conference on Computational Linguistics, COLING-96*, 628–633.

Kasper, W., H.-U. Krieger, J. Spilker, and H. Weber. 1996. From Word Hypotheses to Logical Form: An Efficient Interleaved Approach. In *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, ed. D. Gibbon, 77–88. Berlin: Mouton de Gruyter.

Kiefer, B., and H.-U. Krieger. 2000. Practical Experiments with an HPSG-to-CFG Approximation. Research report.

Kiefer, B., H.-U. Krieger, J. Carroll, and R. Malouf. 1999. A Bag of Useful Techniques for Efficient and Robust Parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, ACL-99*, 473–480.

Lloyd, J. 1987. *Foundations of Logic Programming.* Springer. 2nd edition.

Pollard, C., and I. A. Sag. 1994. *Head-Driven Phrase Structure Grammar.* Studies in Contemporary Linguistics. Chicago: University of Chicago Press.

Shieber, S. M. 1985. Using Restriction to Extend Parsing Algorithms for Complex-Feature-Based Formalisms. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics, ACL-85*, 145–152.

Shieber, S. M. 1986. *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes, Number 4. Stanford: Center for the Study of Language and Information.

Torisawa, K., K. Nishida, Y. Miyao, and J. Tsujii. 2000. An HPSG Parser with CFG Filtering. *Natural Language Engineering.*