Grammar Approximation as a Speed-Up Technique for Unification-Based Parsing

Hans-Ulrich Krieger
German Research Center for Artificial Intelligence (DFKI)
Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
e-mail: `krieger@dfki.de`

ABSTRACT

In this talk, I will present three techniques for obtaining fast grammars from a given unification-based grammar (UBG). The research is motivated by the idea that we can exploit (large-scale), hand-written UBGs not only for the purpose of describing natural language and obtaining a syntactic structure (and perhaps a semantic form), but also to address several other very practical topics. Firstly, to speed up deep parsing by having a cheap recognition pre-filter. Secondly, to obtain an indirect stochastic parsing model for the UBG. Thirdly, to generate domain-specific subgrammars for real applications. And finally, to compile context-free language models for a speech recognizer.

1. Introduction

Developing effective approximation and transformation methods that convert unification-based grammars into more tractable formalisms (e.g., CFGs, TAGs) is a demanding task and is motivated by several practical reasons.

1.1. Approximation

In this talk, I use the term *approximation* for methods that convert a given source grammar $\mathcal{G}$ in some formalism X into a new grammar $\mathcal{G}'$ in another formalism Y (X might be equal to Y), so that $\mathcal{G}'$ is in some sense "simpler" than $\mathcal{G}$. Simpler here means that a sentence that lies in $\mathcal{L}(\mathcal{G}) \cap \mathcal{L}(\mathcal{G}')$ is faster recognized by $\mathcal{G}'$ than by $\mathcal{G}$. Furthermore, when we say *sound* approximation, the following condition should also hold for the generated languages: $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}')$. Thus a sound approximation of the context-free language $\{a^n b^n \mid n \geq 0\}$ is, for instance, the regular language $(a+b)^\star$. During this talk, I will present one unsound and two sound approximation methods that take unification grammars as their input. Two of the methods yield CFGs.

1.2. Motivation

Two-Step Parsing. An approximated CFG can be seen as a cheap recognition pre-filter during natural language parsing with a worst-case running time of $O(n^3)$ for a sentence of length n, whereas unification-based grammar (UBG) parsing usually comes up with an exponential worst-case complexity.[1] Given the existence of such a pre-filter, a unification-based parser thus only

[1]In general, the recognition problem for UBGs is undecidable, but certain side condition, such as the *offline parsability constraint* [12], guarantee decidability. Nevertheless, as shown by [1], popular grammar frameworks are in *theory* intractable, e.g., LFG [13] (NP-hard) or GPSG [11] (ExpPoly-hard).

needs to *deterministically* replay the CF parse trees afterwards, i.e., must not fully explore its search space during all-path parsing [17]. UBG tree reconstruction is easily established by a total many-to-one mapping (a surjective function) from CFG rules onto UBG rules.

Disambiguation of UBGs. By moving from an approximated CFG and a training corpus to a PCFG that predicts probabilities for CF parse trees, we also obtain an indirect and efficient stochastic parsing model for the UBG, even though the UBG might encode an infinite number of categories. I.e., we shift the responsibility for predicting the 'right' derivation trees to the context-free side, where mathematically clean methods exist. Using the probabilities, we can dynamically rank the CF parse trees at runtime, and again, let the UBG parser deterministically replay these trees. This establishes an order on the UBG trees that can be employed for UBG disambiguation [22].

Domain-Specific Subgrammars. The extraction of useful domain-specific context-free subgrammars from general-purpose unification-based grammars is an important topic in information extraction and question answering. The automated construction of such domain-specific grammars may obviate a laboriously manual construction. As we will see later, one of our methods neither produces a superset nor a subset of the language generated by the original UBG. However, this is a desired effect, since a domain-specific subgrammar is not intended to generate a true superset: it should cover the relevant constructions of the domain and only these constructions (in the best case). And because it is also not a true subset of the UBG, it can be robust against ungrammatical input w.r.t. the domain we are interested in.

Context-Free Language Models. Given an UBG, it would be nice to have an automated compilation method that yields a CFG which in turn serves as a symbolic, word-based (instead of phoneme-based) context-free language model, guiding a speech recognizer [32]. This strategy obviates the *sparse data problem* in (commercial) recognizers, since we can directly operate on the high-level grammar without collecting and constructing large amounts of annotated speech training material (Wizard of Oz). The automatic compilation of a CFG from an UBG also makes a tedious hand-coded formulation of a CFG (or a regular grammar) superfluous [9]. Findings in [33] suggest that agreement constraints in context-free language models improve the performance of a recognizer in terms of both word error rate and semantic error rate. Keeping the agreement constraints of the UBG in the CFG is easy in our approximation method. Finally, CF-based models clearly benefit from their greater expressive power when compared to regular models. It is worth noting that within the last four years, the commercial speech community has focused primarily on the grammar-based approach (e.g., VoiceXML or Nuance); see also [33].

2. Parsing in a Unification-Based Framework

This section serves as a brief introduction to the objects (typed feature structures, TFSs) that are built during unification-based parsing and that are manipulated later by our algorithms. From an implementation point of view, TFSs are not that different from records/structures in imperative programming languages (e.g., C) or classes in OO languages such as JAVA. They can also be seen as a generalization of unnamed tuples and fixed-arity terms (as, e.g., in PROLOG). We often say that a TFS is a collection of so-called *feature-value pairs*. Given a TFS, a *feature* expresses a functional property (i.e., having exactly one value) and its value might again be a highly structured TFS. This allows the construction of deeply nested, arbitrary complex objects. TFS formalisms come up with an important descriptive device, viz., *coreferences* or reentrancies (logic variables). This helps to state the fact that the values under several features within a TFS are identical (and not merely structural equal). In this sense, coreference can also serve as a means for information transport. TFSs are *typed* and thus must be interpreted against a multiple inheritance hierarchy of types, for short, a *type hierarchy* (a decidable partial order).

A more thorough investigation considering typed feature formalisms can be found in, e.g., [35], [4], and [24]. In the following, we assume a finite set of *features* $\mathcal{F}$ (often called attributes) and a finite set of *types* $\mathcal{T}$ (often called sorts), and let $\mathcal{FS}$ denote the countably infinite set of *typed feature structures* (TFSs). We often use the term *path* in order to refer to a sequence $f_1 \dots f_n$ of features, helping us to access information from deeper levels of a TFS. One specific path stands out, viz., the empty path ϵ, referring to the TFS itself.

We will not go into the details of unification-based parsing here, but simply assume that it is an extension of standard CF techniques [23, 10] in that the context-free backbone is usually equipped with equality constraints which specify additional well-formedness conditions that must be satisfied and that go beyond the power of ordinary CF string concatenation (see [36, 37] for a more broader view on this topic). As in CFG, we assume a set of (unification) rules $\mathcal{R}$ and a set of lexicon entries $\mathcal{L}$, similar to CF terminal symbols. Modern natural language parsers today are usually (agenda-based) chart parsers [16, 38], for instance, the PET system [3] which we have employed during our experiments below (see [29] for a general introduction to tabular parsing).

UBG formalisms usually provide a monotonic and declarative framework, hence UBGs are in principle reversible, meaning that they are neutral w.r.t. parsing and generation. The term *unification-based* stems from the fact that *unification* is the main operation that checks for valid rule instantiations and that builds up output structure. The examples and algorithms that will follow use the terminology of a modern, state-of-the-art grammar theory, viz, Head-Driven Phrase Structure Grammar [30, 31], actively developed and in use at several sites in the world. It is worth noting that the algorithms can be easily adapted to other formalisms and theories, such as LFG [13], GPSG [11], or PATR-II [35].

2.1. An Example

HPSG is distinguished from older theories and that it is lexicalized, meaning that rules (or rule schemata, to borrow the broader HPSG term) are very general and subsume a large, potentially infinite number of traditional CF rules. Take, for instance, the head-complement rule from HPSG-I, extended by the head-feature principle and the subcategorization principle (boxed items indicate coreference information):

$$\mathrm{HCR} \wedge \mathrm{HFP} \wedge \mathrm{SCP} \equiv \begin{bmatrix} \mathrm{SYN|LOC} & \begin{bmatrix} \mathrm{HEAD} & \boxed{1} \\ \mathrm{SUBCAT} & \langle\,\rangle \end{bmatrix} \\ \mathrm{DTRS} & \begin{bmatrix} \mathrm{HEAD\text{-}DTR|SYN|LOC} & \begin{bmatrix} \mathrm{HEAD} & \boxed{1} \\ \mathrm{SUBCAT} & \langle \boxed{2} \rangle \\ \mathrm{LEX} & - \end{bmatrix} \\ \mathrm{COMP\text{-}DTRS} & \langle \boxed{2} \rangle \end{bmatrix} \end{bmatrix}$$

Most of the linguistic information is kept in the lexicon. Let us look at the (simplified) TFS for the English word *man*:

$$\mathrm{man} \equiv \begin{bmatrix} \mathrm{PHON} & \langle \text{"man"} \rangle \\ \mathrm{SYN|LOC} & \begin{bmatrix} \mathrm{HEAD} & \begin{bmatrix} \mathrm{MAJ} & n \\ \mathrm{NFORM} & \mathit{norm} \end{bmatrix} \\ \mathrm{SUBCAT} & \langle\, \mathit{det} \,\rangle \\ \mathrm{LEX} & + \end{bmatrix} \\ \mathrm{SEM} & \begin{bmatrix} \mathrm{CONT|IND} & \begin{bmatrix} \mathrm{VAR} & \boxed{1} \begin{bmatrix} \mathrm{PER} & \mathit{3rd} \\ \mathrm{NUM} & \mathit{sg} \end{bmatrix} \\ \mathrm{RESTR} & \begin{bmatrix} \mathrm{RELN} & \mathit{man} \\ \mathrm{INST} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \dots & \end{bmatrix} \end{bmatrix}$$

Given these two items and a lexicon entry for the determiner *the*, the head-complement rule licenses the following well-formed phrase:

$$\begin{bmatrix} \textit{hc-rule} \\ \text{PHON}\ \langle \text{"the", "man"} \rangle \\ \text{SYN|LOC|HEAD}\ \boxed{1} \\ \text{DTRS}\ \begin{bmatrix} \text{HEAD-DTR}\ \boxed{2} \\ \text{COMP-DTRS}\ \langle \boxed{3} \rangle \end{bmatrix} \end{bmatrix}$$

$$/ \qquad\qquad\qquad \backslash$$

$$\boxed{3} \begin{bmatrix} \textit{det} \\ \text{PHON}\ \langle \text{"the"} \rangle \\ \text{SYN|LOC|HEAD|MAJ}\ \ \text{D} \end{bmatrix} \qquad \boxed{2} \begin{bmatrix} \textit{noun} \\ \text{PHON}\ \langle \text{"man"} \rangle \\ \text{SYN|LOC|HEAD}\ \boxed{1} \begin{bmatrix} \text{MAJ}\ \ \text{N} \end{bmatrix} \end{bmatrix}$$

2.2. What this Talk is Not About

This talk will *not* cover speedup techniques that operate on the level of the formalism. Especially, I will *not* address topics such as *parsing strategies*, *lazy-copying unification*, *rule/quick-check filtering*, *ambiguity packing/fast subsumption checking*, or *precompiling/caching type unification*; see, e.g., [20, 21, 6, 27] for more information.

3. Speedup Techniques, Part 1: Grammar Modularization

This section focuses on an operation that directly manipulates the original unification grammar in that it deletes information and removes (parts of) that information to a new grammar. The work was originally presented in [8, 14, 15].

3.1. Restriction

The foundation to grammar modularization is an operation called *restriction* [34] that removes unnecessary information from the mother structure of an instantiated unification rule. Take for instance the local tree from section 2.1. Due to the *Locality Principle* of HPSG [30, p. 145], the unnecessary daughters (value under the DTRS feature) in the head-complement structure for *the man* can be legally removed in fully instantiated items, since they only increase the size of the overall TFS, but do not restrict the search space of the parser. In the Verbmobil project, we have found that even restriction alone gave us a speedup by a factor of two for the large German and English grammars [21]. This is due to the fact that speed and memory consumption of the unification operation is (quasi-)linearly related to the size of a TFS and unification itself is by far the most expensive component in UBG parsing (sometimes more than 99% of the overall time).

3.2. Basic Idea

The fundamental observation to grammar modularization is that a UBG simultaneously specifies different kinds of constraints. *Genuine constraints* are directly related to the grammaticality (wellformedness) of the input. Typically, these are the *syntactic* constraints. *Spurious constraints* basically build up representational structures. These constraints are less concerned with wellformedness of the input, but rather with output for other components in an overall system. Much of *semantic* descriptions are of this kind.

If a parser treats all constraints on a par, it cannot distinguish between the structure-building and the filtering constraints. The basic idea here is to distribute the labour of evaluating the constraints among several parsers. In the ASL part of the Verbmobil project, a speech parser (the so-called SYN-parser) used the syntactic constraints of the original grammar to process the input word lattice, whereas an HPSG parser (the SEM-parser) deterministically replayed hypotheses sent by the speech parser. Both parsers ran in parallel and communicated through a specialized protocol in two ways. *Bottom-up hypotheses* are emitted by the SYN-parser and sent to the SEM-parser. A bottom-up hypothesis describes a record on how a subtree was completed. Basically, it tells for each passive edge in the chart which other edges are spanned. *Top-Down hypotheses* result from activities of the SEM-parser, trying to verify bottom-up-hypotheses. To keep the communication efforts low, only failures are reported back to the SYN-parser by sending simply the hypothesis' identifier. Reporting a failure leads to a revision of the SYN-parser's chart. Note that such a protocol only works since both the SYN- and the SEM-parser operate on a subgrammar of the original grammar and rules in SYN have a direct counterpart in SEM, and vice versa.

Modularizing a grammar $\mathcal{G}$ into, say, two subgrammars $\mathcal{G}_{syn}$ and $\mathcal{G}_{sem}$ is realized by employing the above *restriction* operation. Interestingly, the generated languages of $\mathcal{L}(\mathcal{G})$ and $\mathcal{L}(\mathcal{G}_{syn}) \cap \mathcal{L}(\mathcal{G}_{sem})$ are *not* the same, although unification is a monotonic operation. To be more precise, $\mathcal{L}(\mathcal{G}) \subseteq \mathcal{L}(\mathcal{G}_{syn}) \cap \mathcal{L}(\mathcal{G}_{sem})$. This is due to the fact that restriction eliminates coreferences between syntax and semantics that were originally present in $\mathcal{G}$. In Verbmobil, grammar modularization resulted in a speed improvement of the overall parsing process by a factor of two.

4. Speedup Techniques, Part 2: Corpus-Driven Approximation

We now come to an approximation technique that takes a textual representation of the UBG chart and produces annotated context-free grammars. The work was originally presented in [25, 26].

4.1. Basic Idea

Since unification-based parsers usually rely on a context-free backbone of unification rules, it should not be that difficult to extract a context-free grammar. In fact, relatively specific unification-based rules (e.g., ANLT [5] or LFG [13]) should result in approximated CFGs of good quality [7]. However, lexicalized grammar theories such as HPSG, or even categorial grammar frameworks, like CUG [39] or UCG [40], are of a different kind: rule schemata in these frameworks are usually so general that the resulting CFGs are worthless, meaning that they accept nearly everything [2]. As we have seen above, proper recognition of utterances in lexicalized theories is realized by shifting the great amount of information into the lexicon and by applying a specific descriptive means in rule schemata: coreferences.

Our attempt thus does *not* operate on the rules of a unification-based grammar (as we do in the next section during a sound UBG approximation), but instead on valid rule *instantiations* of a special kind, viz., passive edges of the unification chart, resulting from parsing a corpus. In order to access such passive edges, we have defined an external exchange format for representing a chart (similar to the format of bottom-up hypotheses in the last section) and store each chart in a separate file. Since passive edges directly encode their immediate daughters, a passive edge can be seen as a tree of depth 1. From such a tree and with the help of the feature structure directly associated with each passive edge, it is possible to create an annotated context-free rule of arbitrary specificity. Terminal and nonterminal symbols in our framework are equipped with information from the related feature structure of the passive edge, similar to annotated symbols in the GPSG framework [11]. When taking deeper nested daughters into account, we can even escape the flat domain of context-free rules, resulting in CF tree grammars.

```
1  HPSG2CFG(D, Q) :⟺
2    local result = ∅;
3    for each file ∈ D do
4      local edges = makeEdges(file);
5      for i = 1 to |edges| do
6        local e = edges[i];
7        local lhs = ⟨ getName(e),getAnnotation(e, Q) ⟩;
8        local dtrs = getImmediateDaughters(e);
9        local rhs = makeVector(|dtrs|);
10       for j = 1 to |dtrs| do
11         local d = dtrs[j];
12         rhs[j] = ⟨ getName(d),getAnnotation(d, Q) ⟩;
13       end for;
14       result = result ∪≡ {lhs → rhs};
15     end for;
16   end for;
17   return result;
```

Figure 1: The overall structure of the extraction algorithm.

4.2. Algorithm

The idea behind the corpus-driven approximation of a CFG is relatively straightforward and is given in pseudo code in figure 1. As we already said, the HPSG parser produces for each input sentence an output file that contains an external representation of the passive edges of the chart for this sentence. The extraction *HPSG2CFG* then works as follows. Given a directory D and a vector of annotation paths Q that specifies which information should be carried over to the CF symbol, we iterate over the files in D (line 3). For each file, we then construct a vector edges of internal edges for the set of external passive edges stored in this file, using *makeEdges* (line 4). This includes the in-memory reconstruction of the TFSs for the mother structures (the LHSs). For each vector position, i.e., for each edge e, we build up a LHS symbol, consisting of a name field (via *getName*) and an annotation vector (via *getAnnotation*), given the annotation paths Q (line 7). The same is done for every RHS symbol, but since we usually have more than one RHS symbol, we collect them in a vector rhs of length equal to the number of the immediate daughters dtrs of the passive chart edge (lines 8–13). For every passive edge, we finally generate a context-free rule, given the LHS and the RHS (line 14). The new CF rules are adjoined to the result set (line 14). After we have processed all files in directory D, the result set is returned at last (line 17).

The subscript $\equiv$ of the union operator in line 14 of the algorithm should indicate that new rules are added to the result set using rule equivalence.[2] I.e., a new rule only contributes to the final CFG if no structural equivalent rule has already been introduced earlier during the extraction process. Even for a small corpus, a large number of structural equivalent rules are generated, resulting either from reappearing words or from reappearing linguistic constructions. The non-astonishing observation is that the smaller the annotation gets, the larger the number of equivalent rules becomes. Clearly, by taking more annotation paths into account, we obtain more specific CF grammars, consisting of more rules. The interaction between corpus size and the size of the set of annotation paths is given in figure 2.

Measurements have shown that the approximated CFG grammars can be fruitfully employed

[2] An alternative strategy would be to use rule subsumption here.

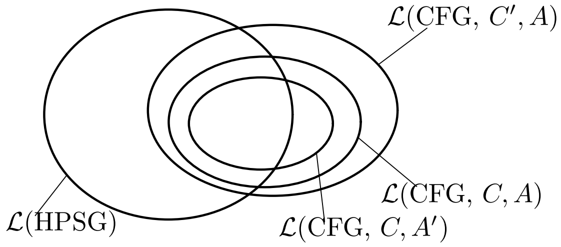

Figure 2: The correlation between the language accepted by the HPSG $\mathcal{L}$(HPSG) and the approximated context-free language $\mathcal{L}$(CFG, C, A), given a corpus C and a set of annotation paths A. The approximated language $\mathcal{L}$(CFG, C', A) monotonically grows when given a larger corpus $C' \supseteq C$. The language $\mathcal{L}$(CFG, C, A') usually shrinks when given a larger set of annotation paths $A' \supseteq A$ (but the *number* of rules grows).

in a two-stage parsing approach as a pre-filter (see section 1). Overall, parsing speed of a two-stage system can be accelerated by up to a factor of 100.

5. Speedup Techniques, Part 3: Grammar-Driven Approximation

Our last approximation technique is sound in that the languages generated by the approximated CFGs are supersets of the language generated by the original UBG. The work was originally presented in [17, 18, 22, 19].

5.1. Basic Idea

In this section, I present a simple and intuitive sound approximation method for turning unification-based grammars, such as HPSG [31] or PATR-II [34], into context-free grammars. The approximation method can be seen as the construction of the least fixpoint of a certain monotonic function and shares similarities with the instantiation of rules in a bottom-up passive chart parser or with partial evaluation in logic programming.

As we have already said in section 4, rule schemata in lexicalized grammar theories (such as HPSG) are too general to be directly translated into a single context-free production (remember the head-complement rule from section 2.1). The basic idea of our approach now is as follows. In a first step, we generalize 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, the so-called *lexicon restrictor*. After that, the grammar rules are instantiated by unification, using the abstracted lexicon entries and resulting in derivation trees of depth 1. We apply the *rule restrictor* to each resulting feature structure, removing all information contained only in the daughters of a rule. Additionally, the restriction gets rid of information that will either lead to infinite growth of the feature structures or that does not constrain the search space. The restricted feature structures (together with older ones) 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, meaning that further iteration steps will not add (or remove) new (or old) feature structures. Given the feature structures from the fixpoint, it is then easy to generate context-free productions, using the full feature structures as symbols of the CFG.

5.2. Approximation as Fixpoint Construction

Let us now introduce the basic inventory that is needed to describe our algorithm. Let $\mathcal{R}$ be the set of all *rules* in our (HPSG) grammar. $ar(r)$ denotes the number of daughters of a given rule $r \in \mathcal{R}$ and is called the *arity* of r. When we say *lexicon entry*, we mean a feature structure that is subsumed by the type *word*. The collection of all *lexicon entries* constitutes the lexicon $\mathcal{L}$. We also need the notion of a *restrictor*, as already introduced in section 3.1. A restrictor can be seen as an automaton or alternatively here as a function, describing the paths in a feature structure that will remain after *restriction*, that is, after applying the deletion operation. The result of this operation is again a feature structure that always subsumes (is more general than) the input structure. We employ two different restrictors in our approach: the *lexicon restrictor* L and the *rule restrictor* R. We abbreviate the countable infinite set of TFSs by $\mathcal{FS}$.

We can now 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:

$$r : \mathcal{FS}^n \longmapsto \mathcal{FS} \tag{5.1}$$

Similarly, we can view the lexicon restrictor and the rule restrictor as a (unary) function:

$$L, R : \mathcal{FS} \longmapsto \mathcal{FS} \tag{5.2}$$

During each iteration step i, we obtain the following sets T_i of feature structures:

$$T_0 := \bigcup_{l \in \mathcal{L}} L(l) \tag{5.3}$$

$$T_{i+1} := \bigcup_{k=0}^{i} \bigcup_{r \in \mathcal{R}} \bigcup_{t \in (T_k)^{ar(r)}} R(r(t)) \tag{5.4}$$

(5.3) corresponds to the generalization process of the lexicon, whereas (5.4) exactly describes the $i + 1$ step of the iteration, taking into account feature structures from the previous steps $0, \ldots, i$.[3]

More formally in terms of the (upward) ordinal power of a monotonic mapping T [28, p. 26–31], we define

$$T(S) := S \cup \bigcup_{r \in \mathcal{R}} \bigcup_{s \in S^{ar(r)}} R(r(s)) \tag{5.5}$$

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

$$T : 2^{\mathcal{FS}} \longmapsto 2^{\mathcal{FS}} \tag{5.6}$$

The idea behind T is that it will exactly enumerate those feature structures which are legal instantiations of underspecified rule schemata and which have undergone the application of the rule restrictor R. In the end, we are interested in sets of feature structures $S \in 2^{\mathcal{FS}}$, where

$$T(S) = S \tag{5.7}$$

since in this case, S has been saturated, hence we can stop the iteration process. If so, we call S a *fixpoint* of T. We say that S is the *least* fixpoint of T iff S is a fixpoint and for all other fixpoints S' of T, we have $S \subseteq S'$. Clearly, T is *monotonic*, since for all $S, S' \in 2^{\mathcal{FS}}$ with $S \subseteq S'$, we have $T(S) \subseteq T(S')$. This is due to the fact that set union employed in the definition of T is

[3] $(T_k)^{ar(r)}$ represents the $ar(r)$-times Cartesian product of T_k, i.e., vectors (or tuples) of length $ar(r)$ with elements from T_k (i.e., from previous iterations).

a monotonic operation (and does not delete any other information). Furthermore, the argument S of T occurs on the right side of definition (5.5), thus $S \subseteq T(S)$ is always true.

We then define the approximated context-free grammar $\mathcal{G}$ as the *least fixpoint* of T which is well-defined, since T is monotonic and operates on a complete lattice, viz., $2^{\mathcal{FS}}$ (the famous Knaster-Tarski fixpoint theorem):

$$\mathcal{G} := \mathit{lfp}(T) \tag{5.8}$$

It should now be clear that a *finite* fixpoint can only be reached if features, leading to an infinite growth of the overall structure, are deleted by the restrictor. Clearly, the more features are removed, 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 predefined finite iteration depth usually neither leads to a subset nor a superset of the original HPSG. But even a finite iteration depth (say 6 or 10 iterations) 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.

5.3. Algorithm

We will now describe the naïve approximation algorithm which can be seen as a direct implementation of the definitional equations (5.3) and (5.4), together with the fixpoint termination criterion given by (5.7).

We start with the description of the top-level function *UBG2CFG* which initiates the approximation process. We begin the approximation by first abstracting from the lexicon entries $\mathcal{L}$ with the help of the lexicon restrictor L (line 5 of the algorithm). This constitutes our initial set T_0 (line 6). We note here that an abstracted lexicon entry is not merely added using set union but instead through a more complex operation (viz., TFS subsumption), depicted by $\cup_{\sqsubseteq}$. At this stage of presentation, it suffices to assume that $\cup_{\sqsubseteq}$ simply denotes set union. Finally, we start the true iteration calling *Iterate* with the necessary parameters.

1 $\mathit{UBG2CFG}(\mathcal{R}, \mathcal{L}, R, L) :\Longleftrightarrow$
2   **var** T_0;
3   $T_0 := \emptyset$;
4   **for each** $l \in \mathcal{L}$ **do**
5     $l := L(l)$;
6     $T_0 := T_0 \cup_{\sqsubseteq} \{l\}$;
7   $\mathit{Iterate}(\mathcal{R}, R, T_0)$.

After that, the instantiation of the rule schemata with rule-/lexicon-restricted elements from the previous iteration T_i begins (lines 10–13). The instantiation is performed in *Iterate* by *Fill-Daughters* which takes into account a single rule r and T_i, returning successfully instantiated feature structures t to which we apply the rule restrictor. We then add the restricted t to T_{i+1}, employing $\cup_{\sqsubseteq}$ (line 13).

It should be noted that the rule restrictor should be carefully tuned to get rid of those paths whose values would otherwise grow infinitely during the approximation. Now, to guarantee a fixpoint that can be reached in *finitely* many steps (and we are clearly interested in this), such information must be deleted in order to enforce a convergence of $\langle T_i \rangle_{i \geq 0}$.

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 T_i; otherwise, we proceed with the iteration (lines 14–16).

$\mathit{Iterate}(\mathcal{R}, R, T_i) :\Longleftrightarrow$
  var $T_{i+1} := T_i$;
  for each $r \in \mathcal{R}$ **do**
    for each $t \in \mathit{Fill\text{-}Daughters}(r, T_i)$ **do**
      $t := R(t)$;
      $T_{i+1} := T_{i+1} \cup_{\sqsubseteq} \{t\}$;
  if $T_i = T_{i+1}$
    then return T_i
    else $\mathit{Iterate}(\mathcal{R}, R, T_{i+1})$.

5.4. Outlook

Due to space limitations, certain points are missing here which, however, will be presented in my talk. Firstly, the above algorithm is intractable for large-scale UBGs and several optimizations have been developed to make it feasible. Secondly, the output of the algorithm is *not* a set of CF rules, but instead a set of TFSs, representing those rules. As we will see, the theoretical basis for computing the rules from the TFS set comes from the properties of the fixpoint. Thirdly, since TFSs are added under TFS subsumption, the mathematical apparatus in section 5.2 is not correct and only works for TFS equivalence. The overall idea, however, can be saved by moving form the powerset of TFSs to the complete restricted powerset of $\mathcal{FS}$ under the so-called *Hoare-ordering*.

Acknowledgments

Parts of the work presented here have benefited from discussions or has been joint work with the following fine people: John Carroll, Abdel Kader Diagne, Dan Flickinger, Walter Kasper, Bernd Kiefer, Rob Malouf, Mark-Jan Nederhof, Stephan Oepen, and Detlef Prescher. This work was supported by the German BMBF and the EU in the projects ASL, Verbmobil, Airforce, and QUETAL.

References

[1] G. Edward Barton, Robert C. Berwick, and Eric Sven Ristad. *Computational Complexity and Natural Language*. MIT Press, Cambridge, MA, 1987.

[2] Ted Briscoe and John Carroll. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59, 1993.

[3] Ulrich Callmeier. PET—a platform for experimentation with efficient HPSG processing. *Natural Language Engineering*, 6(1):99–107, 2000.

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

[5] John Carroll, Ted Briscoe, and Claire Grover. A development environment for large natural language grammars. Technical Report 233, Computer Laboratory, Cambridge University, UK, 1991.

[6] John Carroll and Stephan Oepen. Ambiguity packing in HPSG—Practical results. In *Proceedings of the 1st Conference of the North American Chapter of the ACL, NAACL2000*, 2000.

[7] John Andrew Carroll. *Practical Unification-based Parsing of Natural Language*. PhD thesis, University of Cambridge, Computer Laboratory, Cambridge, England, September 1993.

[8] Abdel Kader Diagne, Walter Kasper, and Hans-Ulrich Krieger. Distributed parsing with HPSG grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies, IWPT'95*, pages 79–86, 1995. Also available as DFKI Research Report RR-95-19.

[9] John Dowding, Beth Ann Hockey, Jean Mark Gawron, and Christopher Culy. Practical issues in compiling typed unification grammars for speech recognition. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL-2001*, pages 164–171, 2001.

[10] Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, February 1970.

[11] Gerald Gazdar, Ewan Klein, Geoffrey Pullum, and Ivan Sag. *Generalized Phrase Structure Grammar.* Harvard University Press, Cambridge, MA, 1985.

[12] Mark Johnson. *Attribute Value Logic and the Theory of Grammar.* CSLI Lecture Notes, Number 16. Center for the Study of Language and Information, Stanford, 1988.

[13] Ronald Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, Cambridge, Mass, 1982.

[14] Walter Kasper and Hans-Ulrich Krieger. Modularizing codescriptive grammars for efficient parsing. In *Proceedings of the 16th International Conference on Computational Linguistics, COLING-96*, pages 628–633, 1996.

[15] Walter Kasper, Hans-Ulrich Krieger, Jörg Spilker, and Hans Weber. From word hypotheses to logical form: An efficient interleaved approach. In D. Gibbon, editor, *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, pages 77–88. Mouton de Gruyter, Berlin, 1996.

[16] Martin Kay. Algorithm schemata and data structures in syntactic processing. Technical report, Xerox Palo Alto Research Center, Palo Alto, California, 1980.

[17] Bernd Kiefer and Hans-Ulrich Krieger. A context-free approximation of Head-Driven Phrase Structure Grammar. In *Proceedings of the 6th International Workshop on Parsing Technologies, IWPT2000*, pages 135–146, 2000.

[18] Bernd Kiefer and Hans-Ulrich Krieger. A context-free approximation of Head-Driven Phrase Structure Grammar. In S. Oepen, D. Flickinger, J. Tsuji, and H. Uszkoreit, editors, *Collaborative Language Engineering. A Case Study in Efficient Grammar-based Processing*, pages 49–76. CSLI Publications, 2002.

[19] Bernd Kiefer and Hans-Ulrich Krieger. A context-free superset approximation of unification-based grammars. In H. Bunt, J. Carroll, and G. Satta, editors, *New Developments in Parsing Technology*, pages 229–250. Kluwer Academic Publishers, 2004.

[20] Bernd Kiefer, Hans-Ulrich Krieger, John Carroll, and Rob Malouf. 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*, pages 473–480, 1999.

[21] Bernd Kiefer, Hans-Ulrich Krieger, and Mark-Jan Nederhof. Efficient and robust parsing of word hypotheses graphs. In W. Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, pages 280–295. Springer, Berlin, 2000.

[22] Bernd Kiefer, Hans-Ulrich Krieger, and Detlef Prescher. A novel disambiguation method for unification-based grammars using probabilistic context-free approximations. In *Proceedings of the 19th International Conference on Computational Linguistics, COLING2002*, 2002.

[23] Donald E. Knuth. On the translation of languages from left to right. *Information and Control*, 8(6):607–639, 1965.

[24] Hans-Ulrich Krieger. *TDL—A Type Description Language for Constraint-Based Grammars. Foundations, Implementation, and Applications.* PhD thesis, Universität des Saarlandes, Department of Computer Science, September 1995.

[25] Hans-Ulrich Krieger. A corpus-driven context-free approximation of head-driven phrase structure grammar. In Georgios Paliouras and Yasubumi Sakakibara, editors, *Proceedings of the 7th International Colloquium on Grammatical Inference, ICGI-2004*, number 3264 in Lecture Notes in Artificial Intelligence, pages 199–210. Springer, 2004.

[26] Hans-Ulrich Krieger. From UBGs to CFGs—a practical corpus-driven approach. *Natural Language Engineering*, 2005. Kluwer, forthcoming.

[27] Hans-Ulrich Krieger, Witold Drożdżyński, Jakub Piskorski, Ulrich Schäfer, and Feiyu Xu. A bag of useful techniques for unification-based finite-state transducers. In *Proceedings of KONVENS 2004*, pages 105–112, 2004.

[28] J.W. Lloyd. *Foundations of Logic Programming*. Springer, 2nd edition, 1987.

[29] Mark-Jan Nederhof and Giorgio Satta. Tabular parsing. In C. Martin-Vide, V. Mitrana, and G. Paun, editors, *Formal Languages and Applications*, pages 529–549. Springer, Berlin, 2004. Studies in Fuzziness and Soft Computing 148.

[30] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics. Vol. I: Fundamentals*. CSLI Lecture Notes, Number 13. Center for the Study of Language and Information, Stanford, 1987.

[31] Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. University of Chicago Press, Chicago, 1994.

[32] Manny Rayner, John Dowding, and Beth Ann Hockey. A baseline method for compiling typed unification grammars into context free language models. In *Proceedings of EUROSPEECH*, 2001.

[33] Manny Rayner, Genevieve Gorrell, Beth Ann Hockey, John Dowding, and Johan Boye. Do CFG-based language models need agreement constraints. In *Proceedings of the 2nd Conference of the North American Chapter of the ACL, NAACL2001*, 2001.

[34] Stuart M. Shieber. 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*, pages 145–152, 1985.

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

[36] Stuart M. Shieber. Parsing and type inference for natural and computer languages. Technical Note 460, SRI International, Menlo Park, MA, March 1988. Also available as *Constraint-Based Grammar Formalisms*, MIT Press, 1992.

[37] Stuart M. Shieber, Yves Shabes, and Fernando C.N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1–2):3–36, 1995.

[38] Henry S. Thompson. Chart parsing and rule schemata in GPSG. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 167–172, 1981.

[39] Hans Uszkoreit. Categorial unification grammars. In *Proceedings of the 11th International Conference on Computational Linguistics*, pages 187–194, 1986.

[40] Henk Zeevat, Ewan Klein, and Jo Calder. Unification categorial grammar. In N. Haddock, E. Klein, and G. Morrill, editors, *Edinburgh Working Papers in Cognitive Science, 1: Categorial Grammar, Unification Grammar, and Parsing*, pages 195–222. Centre for Cognitive Science, Edinburgh University, UK, 1987.