Natural Language Engineering **13** (4): 317–351. © 2006 Cambridge University Press
doi:10.1017/S1351324906004128 First published online 26 April 2006 Printed in the United Kingdom

From UBGs to CFGs
A practical corpus-driven approach

HANS-ULRICH KRIEGER

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

(*Received 11 November 2004; revised 6 July 2005*)

Abstract

We present a simple and intuitive unsound corpus-driven approximation method for turning unification-based grammars, such as HPSG, CLE, or PATR-II into context-free grammars (CFGs). Our research is motivated by the idea that we can exploit (large-scale), hand-written unification grammars 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 (the approximated CFG). Secondly, to obtain an indirect stochastic parsing model for the unification grammar through a trained PCFG, obtained from the approximated CFG. This gives us an efficient disambiguation model for the unification-based grammar. Thirdly, to generate domain-specific subgrammars for application areas such as information extraction or question answering. And finally, to compile context-free language models which assist the acoustic model of a speech recognizer. The approximation method is unsound in that it does not generate a CFG whose language is a true superset of the language accepted by the original unification-based grammar. It is a corpus-driven method in that it relies on a corpus of parsed sentences and generates broader CFGs when given more input samples. Our open approach can be fine-tuned in different directions, allowing us to monotonically come close to the original parse trees by shifting more information into the context-free symbols. The approach has been fully implemented in JAVA.

1 Introduction

We present a simple and intuitive unsound corpus-driven approximation method for turning unification-based grammars (UBGs), such as HPSG (Pollard and Sag 1994), CLE (Alshawi 1992), or PATR-II (Shieber, Uszkoreit, Pereira, Robinson and Tyson 1983), into context-free grammars (CFGs). The method is *unsound* (Pereira and Wright 1991) in that it does *not* generate a CFG whose language is a true superset of the language accepted by the original unification-based grammar. It is a corpus-driven method in that it relies on a corpus of parsed sentences and generates broader CFGs when given more input samples. Our open approach can be fine-tuned in different directions, allowing us to monotonically come close to the original parse trees by shifting more information into the context-free symbols. The approach has been fully implemented in JAVA. This article updates and extends the

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

paper presented at the International Colloquium on Grammatical Inference ICGI 2004 (Krieger 2004) and presents further measurements.

1.1 Motivation

The motivation for developing this corpus-based approximation method was manifold.

Two-Step Parsing. An approximated CFG can be seen as a cheap recognition pre-filter during NL parsing with a worst-case running time of $O(n^3)$ for a sentence of length n, whereas UBG parsing usually comes up with an exponential worst-case complexity. Given the existence of such a pre-filter, a unification-based parser thus only needs to *deterministically* replay the CF parse trees afterwards, i.e., must not fully explore its search space during all-path parsing (Kiefer and Krieger 2002; Kiefer and Krieger 2004). 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 the 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 (Kiefer, Krieger and Prescher 2002).

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 (Neumann 2003). Our approach thus may obviate the laboriously manual construction of such domain-specific grammars. As we will see later, our method 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 (Rayner, Dowding and Hockey 2001a). 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 (Dowding, Hockey, Gawron and Culy 2001).

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

Findings in Rayner, Gorrell, Hockey, Dowding and Boye (2001b) 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 or so, the commercial speech community has focused primarily on the grammar-based approach (VoiceXML, W3C, Nuance, Speech Works, etc.); see also Rayner *et al* (2001a).

Open Approach. The adjusting parameters of our approach make it easy to approximate CFGs of different size and quality, e.g., by varying the annotations of context-free symbols, by collecting the approximated CF rules either under rule equivalence or under rule subsumption, or by taking a larger domain of locality into account, by using the feature structures of the UBG, resulting in various forms of context-free tree grammars. Even though these tree grammars are still context-free, they clearly have a larger event horizon, establishing a restricted form of look-ahead.

1.2 General Idea

Since unification-based parsers usually rely on a context-free backbone of unification rules (or rule schemata, to borrow the broader HPSG term), it should not be that difficult to extract a context-free grammar. In fact, relatively specific unification-based rules, for instance, in ANLT (Carroll, Briscoe and Grover (1991) or LFG (Kaplan and Bresnan 1982)), should result in approximated CFGs of good quality (cf. Carroll (1993) and section 4 for a discussion of other approaches). However, lexicalized grammar theories such as HPSG, or even categorial grammar frameworks, like CUG (Uszkoreit 1986) or UCG (Zeevat, Klein and Calder 1987), 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 (Briscoe and Carroll 1993). 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 or reentrancies.

Our attempt thus does *not* operate on the rules of a unification-based grammar – as do (Kiefer and Krieger 2002) in their sound HPSG 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 (see section 3.1). 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 (section 3). 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 (Gazdar, Klein, Pullum and Sag 1985) (see section 2.1). When taking deeper nested daughters into

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

account, we can even escape the flat domain of context-free rules, resulting in CF tree grammars.

1.3 Structure of the article

The structure of this article is as follows. In the next section, we first discuss the objects which are constructed and manipulated during the extraction of the CFG (symbols, rules, edges). After that, section 3 presents the interface to the chart of the HPSG parser and describes the basic extraction algorithm, together with a variation which produces smaller, although more general CFGs. The section also has a few words on the quick-check paths of the UBG which serve as the starting point for finding the proper annotations of the context-free symbols. Section 4 then comes with a discussion of other approaches which aim at extracting a CFG from an UBG, most of them unsound. In section 5, we elaborate on further aspects of the extraction methods, discussing several orthogonal adjusting parameters which result in different CFGs, given a UBG. We also discuss additional postprocessing steps and motivate that the extracted CFGs can be tuned to deliver a meaningful semantic output as well. After that, section 6 applies our method to the large English Resource Grammar, developed at CSLI, Stanford, including a small evaluation w.r.t. to a 10% hold-out sample set. Finally, section 7 indicates our next steps.

2 Objects of Interest

The goal of this section is a description of the implemented objects which are built and manipulated by the extraction algorithm (section 3). The section furthermore defines certain important relations between symbols and rules. We assume a basic familarity with typed feature structures (TFSs) here (Carpenter 1992), and refer to the set of all types by $\mathscr{T}$. By type subsumption we mean the decidable partial order $\preceq \subseteq \mathscr{T} \times \mathscr{T}$.

2.1 Symbols

Terminal and nonterminal symbols of the context-free grammar are represented as instances of the JAVA class `Symbol`.[1] Symbols bear a `name` field of type `String` and an `annotation` field of type `int[]` (an integer array). The `name` field of a terminal refers to the full surface string of this terminal word and its `annotation` field is empty (refers to the `null` value). A nonterminal also has a `name` and encodes the HPSG rule name (e.g., hcomp or measure_np). The `annotation` of a nonterminal symbol groups several type identifiers which originate from (possibly deep-nested) values under pre-specified paths (the so-called annotation paths) within the instantiated rule TFS for this nonterminal symbol. Thus an annotation is quite similar to a feature specification in GPSG (Gazdar et al. 1985) or a quick-check vector (Kiefer, Krieger, Spilker and Weber 1996; Malouf, Carroll and Copestake 2000).

[1] Note that during the more technical aspects of this article, I will adopt the JAVA notation and its specific language use, e.g., by referring to an instance variable as a *field* (Flanagan 2002).

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

During the more formal parts of this article, we need two fundamental concepts: symbol and annotation.

Definition 1 (SYMBOL)
We refer to a symbol s in the abstract syntax by a pair $\langle n, a \rangle$, consisting of name n and an annotation a. We write $N(s)$ to depict the first projection of s (the name) and $A(s)$ to access the annotation part of s. When using concrete syntax, we write a symbol in the more GPSGish notation `n[a]`.

Definition 2 (ANNOTATION)
An annotation $a = \langle t_1, \ldots, t_n \rangle$ is a n-tuple of type names or type IDs $t_i \in \mathscr{T}$, $i \in \{1, \ldots, n\}$. We write $\pi_i(a)$ to denote value t_i of the i-th projection of a. Using the GPSG-like concrete syntax, we write $[t_1, \ldots, t_n]$ to depict annotation a.

Given two symbols, we define a subsumption relation $\preceq$ which turns out to be useful in a moment.

Definition 3 (SYMBOL SUBSUMPTION)
Let s_1 and s_2 be two annotated context-free symbols. s_1 is said to be subsumed by s_2 (written as $s_1 \preceq s_2$) iff $N(s_1) = N(s_2)$ and $\pi_i(A(s_1)) \preceq \pi_i(A(s_2))$, for all $i \in \{1, \ldots, n\}$. Alternatively, we say that s_2 subsumes (or is more general than) s_1. Assuming that s_1 and s_2 are terminal symbols, their annotation must be empty ($A(s_1) = A(s_2) = \langle\ \rangle$), thus the second condition above is trivially satisfied.[2]

Let us give an example. Assume we have the four CF symbols (concrete syntax) `N[sg,fem]`, `N[pl,fem]`, `N[num,fem]`, `D[num,fem]` and assume that `sg` $\preceq$ `num` and `pl` $\preceq$ `num` is the case. We can then infer that `N[sg,fem]` $\preceq$ `N[num,fem]` and `N[pl,fem]` $\preceq$ `N[num,fem]` holds. However, `N[sg,fem]` and `N[pl,fem]` are not related by the subsumption relation. Consequently, we say that these symbols are incomparable and use the $\bowtie$ sign to indicate this: `N[sg,fem]` $\bowtie$ `N[pl,fem]`. Furthermore, `D[num,fem]` is incomparable with every other symbol, due to its name: `D` $\neq$ `N`.

2.2 Rules

We represent context-free grammar rules by the class `Rule` whose instances have a left-hand side (`lhs`) and a right-hand side (`rhs`) field. The left-hand side is (from a CFG point of view) a nonterminal symbol, which we represent as an instance of the class `Symbol`. The right hand-side is an array of `Symbol` objects (JAVA notation: `Symbol[]`). The length of `rhs` is encoded in an additional field `length` of type `int`. A `Rule` object also possesses a frequency field, telling us how often that rule has been applied during parsing of a given corpus. It is worth noting that the frequency counter might gain importance when we move from the extracted CFG to a PCFG which will predicts probabilities for CFG trees.

Next, we need the formal notation of a grammar rule.

[2] Note that we have overloaded (and in the following will further overload) the $\preceq$ relation, originally used to depict type subsumption.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

Definition 4 (CONTEXT-FREE GRAMMAR RULE)
Let $l, r_1, \ldots, r_n$ be (annotated) context-free symbols and let furthermore l be a nonterminal symbol. We then call the expression $l \rightarrow r_1 \ldots r_n$ a context-free grammar rule. l is usually referred to as the left-hand side (LHS) of the rule, whereas the sequence $r_1 \ldots r_n$ is called the right-hand side (RHS). Given a CF rule α, the projection L yields the LHS of α, i.e., $L(\alpha) = l$. R delivers the RHS: $R(\alpha) = r_1 \ldots r_n$.

Given the above apparatus, defining a subsumption relation $\preceq$ on rules is relatively straightforward.

Definition 5 (RULE SUBSUMPTION)
Let $\alpha = (l_\alpha \rightarrow r_{1\alpha} \ldots r_{n\alpha})$ and $\beta = (l_\beta \rightarrow r_{1\beta} \ldots r_{n\beta})$ be two context-free grammar rules. We say that α is subsumed by β and write $\alpha \preceq \beta$ iff $l_\alpha \preceq l_\beta$, $n_\alpha = n_\beta$, and $r_{i\alpha} \preceq r_{i\beta}$, for all $1 \leq i \leq n_\alpha$. Alternatively, we say that β subsumes (or is more general than) α.

We say that two CF rules α and β are equivalent iff they both subsume each other: $\alpha \equiv \beta :\Longleftrightarrow \alpha \preceq \beta$ and $\beta \preceq \alpha$.

Let $\alpha \not\preceq \beta$ abbreviate $\neg(\alpha \preceq \beta)$. Two rules are said to be incomparable iff they are not related by rule subsumption: $\alpha \bowtie \beta :\Longleftrightarrow \alpha \not\preceq \beta$ and $\beta \not\preceq \alpha$.

An example. Given the symbols from the example at the end of subsection 2.1, we define a subsumption order over the following three rules (we use again the concrete syntax in which the implemented algorithm delivers the rules). Let α = (`NP[sg,fem] -> N[sg,fem]`), β = (`NP[pl,fem] -> N[pl,fem]`), and γ = (`NP[num,fem] -> N[num,fem]`). Then $\alpha \preceq \gamma$ and $\beta \preceq \gamma$, however $\alpha \bowtie \beta$.

2.3 Edges

The edges which are transmitted by the HPSG parser in plain text (see figure 1) are reconstructed in main memory within the JAVA virtual machine. At the moment, we are using a modification of the freely available PET parser (Callmeier 2000). The set of all passive edges for a single parsed sentence are grouped in a text file (see next section).

Edges are represented as instances of the JAVA class `Edge`, consisting of the instance fields `id`, `ruleName`, `immDtrs`, `noOfDtrs`, and `annotation`. The `id` of an edge is a handle to the edge object and allows other edges to refer to this edge in their immediate daughters array `immDtrs`. `id` is of type `int`, thus `immDtrs` must be of type `int[]`. The string in the `ruleName` field of the edge leads to the primary category symbol of the LHS of the context-free rule later. As already described, annotations are represented as `int` vectors. The extraction algorithm in the next section produces for each given edge exactly one context-free rule. Due to the `ruleName` field, we know the name of the UBG rule from which the edge has been derived. Exactly this information is utilized during the deterministic second phase of two-stage parsing.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

3 Extracting a Context-Free Grammar

This section centers around the offline extraction of a context-free grammar from a given corpus, originally parsed by the deep HPSG parser of the PET system (Callmeier 2000). We first describe the textual interface between our extraction component and PET. After this, we motivate that the annotation values of context-free symbols for a *recognition* grammar are related to the quick-check paths, originally introduced within the context of deep HPSG parsing (Kiefer et al. 1999). Given this background, we then describe the basic extraction algorithm in pseudo code. Finally, we argue for an extension of the original algorithm, which helps to compute smaller grammars and describe how the frequencies for the context-free rules are obtained.

3.1 Interface to HPSG

The interface to HPSG is established via the creation of text files: for every input sentence of the corpus, a new file is created that contains exactly the passive edges produced by PET. Although not every deep passive edge contributes to a deep reading, we have decided to take all passive edges into account (one can think of other options as well; see section 5). Since the passive edges of the deep parser are objects in main memory and since our extraction runs in a separate thread, we have defined an ASCII-based exchange format for chart edges that is given by the EBNF in figure 1. Figure 2 then displays the stripped-down chart for the sentence *Kim loves Sandy*. Due to the fact that features and types are represented as integers in PET, it is important that both PET and the extraction process operate on the same $\mathcal{TDL}$ (Krieger and Schäfer 1994) grammar.

Because HPSG requires all relevant information to be contained in the `synsem` feature of the mother structure, the unnecessary daughters (which are part of the TFS) only increase the size of the overall feature structure without constraining the search space. Due to the *Locality Principle* of HPSG (Pollard and Sag 1987), they can therefore be legally removed in fully instantiated items, i.e., passive edges which are delivered by the PET parser. To be independent from a certain grammatical theory or implementation, we use *restrictors* similar to Shieber (1985) as a flexible and easy-to-use specification to perform this deletion. In case we are trying to work with a larger tree context and not limiting ourselves to context-free rules (= trees of depth 1), the restrictor is the right means to accomplish this (see section 5).

3.2 Quick-check paths

Quick-check paths (QC) are used during unification-based parsing to quickly and correctly filter out failing unifications without applying the more costly unification operation (Kiefer et al. 1999; Malouf et al. 2000). Such a filter is extremely important since most of the unifications usually fail (95–99% of all unifications). The quick-check filter exploits the fact that unification fails more often at certain points in feature structures than at others. In order to determine the most prominent failure points, we parse a large test corpus in an offline stage (to be precise, we use a part of the corpus from which we extract the CFG), using a special unification engine that

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

```
feat      ::= integer
type      ::= integer
coref     ::= # integer
fvpair    ::= feat { {coref} tfs | coref }
tfs       ::= [ type fvpair* ]
start     ::= integer
end       ::= integer
id        ::= integer
edgename  ::= id | string
rulename  ::= string
imdtrs    ::= ( edgename+ )
edge      ::= id start end rulename imdtrs tfs <CR>
chart     ::= start end edge+
```

Fig. 1. The external exchange format of a chart as delivered by a modified version of the PET system. The meta characters { and } express optionality, | enforces the choice of exactly one alternative, and * and + refer to Kleene star and Kleene plus, resp. *integer* denotes the set of all integers, i.e., sequences of numeric decimal characters. *string* denotes the set of all strings, i.e., sequences of plain text characters, enclosed by double quotes. Note that each chart edge must be separated by the newline character <CR> at the end. At the moment, we do not benefit from the start and end position of an edge. *rulename* serves as the main category symbol. *imdtrs* refers to the immediate daughters of an edge. Since the delivered passive edges are topologically sorted, it is not necessary to reencode an edge that has already been introduced earlier. Thus the immediate daughters *imdtrs* of an edge are referred to by integer numbers *id*. The only exception are terminal symbols (the surface form) which are written as pure strings.

records all failures instead of exiting after the first failing unification.[3] These failure points, more exactly, the types of the feature structures at these points, constitute the quick-check (QC) vector. When executing unification during parsing, those points are efficiently accessed and checked using type unification prior to the rest of the structure. QC filtering heavily relies on type unification being very fast, which in fact is the case since it can be cached or even precompiled (Kiefer *et al.* 1999; Kiefer, Krieger and Nederhof 2000). Figure 3 displays the ten most prominent failure paths for a specific corpus we have used in our experiments (cf. section 6).

As already said in section 2.1, the annotation of a context-free symbol bears a close resemblance to a QC vector – an annotation is a subvector of a QC vector., i.e., we might not take all vector positions into account. The reason for using (parts of) the QC vector as an annotation is due to the fact that we are interested in fast and modestly overgenerating context-free *recognition* grammars. Exactly the failure points in a QC vector are of this property, viz., heavily contributing to failures which rule out parts of the search space during unification-based parsing.

[3] It does not make a big difference which part of the corpus is chosen, provided that at least 100–200 sentences are used for training. Assuming that the test suite covers the most common phenomena (and grammar writers have such suites), the most prominent paths always come up first, and only rare paths in the 'tail' of the set of all QC paths are rated differently. This observation is shared by other grammar writers (p.c. Berthold Crysmann). Clearly, when manipulating the type hierarchy (feature topology, grammar rules, lexicon entries), a different set of QC paths is usually obtained.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

```
0 3
1 0 1 Kim[] ( "kim" )  [ 28398 76 ... ]
11 2 3 Sandy[] ( "sandy" )  [ 28398 76 ... ]
12 0 1 sing_noun_infl_rule ( 1 )  [ 27973 77  ... ]
13 2 3 sing_noun_infl_rule ( 11 )  [ 27973 77  ... ]
3 1 2 love_v2[plur_noun_infl_rule] ( "loves" )  [ 27896 76  ... ]
6 1 2 love_v2[third_sg_fin_verb_infl_rule] ( "loves" )  [ 27896 76  ... ]
9 1 2 love_v2[possessed_word_lr] ( "loves" )  [ 27896 76  ... ]
15 1 2 third_sg_fin_verb_infl_rule ( 6 )  [ 27973 77  ... ]
2 1 2 love_v1[plur_noun_infl_rule] ( "loves" )  [ 27838 76  ... ]
4 1 2 love_v3[plur_noun_infl_rule] ( "loves" )  [ 27890 76  ... ]
5 1 2 love_v1[third_sg_fin_verb_infl_rule] ( "loves" )  [ 27838 76  ... ]
7 1 2 love_v3[third_sg_fin_verb_infl_rule] ( "loves" )  [ 27890 76  ... ]
8 1 2 love_v1[possessed_word_lr] ( "loves" )  [ 27838 76  ... ]
10 1 2 love_v3[possessed_word_lr] ( "loves" )  [ 27890 76  ... ]
17 0 1 proper_np ( 12 )  [ 2265 76  ... ]
18 2 3 proper_np ( 13 )  [ 2265 76  ... ]
19 1 2 third_sg_fin_verb_infl_rule ( 5 )  [ 27973 77  ... ]
21 1 3 hcomp ( 19 18 )  [ 2238 78
22 1 2 third_sg_fin_verb_infl_rule ( 7 )  [ 27973 77  ... ]
26 0 3 subjh ( 17 21 )  [ 2237 76  ... ]
28 1 3 extradj_t ( 21 )  [ 2261 78  ... ]
30 0 3 subjh ( 17 28 )  [ 2237 76  ... ]
31 1 3 extradj_i_subj ( 21 )  [ 2260 78  ... ]
33 0 3 subjh ( 17 31 )  [ 2237 76  ... ]
34 0 3 extradj_t ( 26 )  [ 2261 78  ... ]
35 1 2 extracomp ( 19 )  [ 2244 74  ... ]
37 0 2 subjh ( 17 35 )  [ 2237 76  ... ]
38 1 3 extrasubj_f ( 21 )  [ 2245 74  ... ]
40 0 3 fin_non_wh_rel ( 30 )  [ 2262 74  ... ]
41 0 2 fin_non_wh_rel ( 37 )  [ 2262 74  ... ]
42 0 3 fin_non_wh_rel ( 34 )  [ 2262 74  ... ]
44 0 1 nocop_id_vp ( 17 )  [ 2303 74  ... ]
45 0 1 nocop_s ( 44 )  [ 2304 78  ... ]
47 2 3 nocop_id_vp ( 18 )  [ 2303 74  ... ]
49 2 3 nocop_s ( 47 )  [ 2304 78  ... ]
```

Fig. 2. Parts of the chart for the sentence *Kim loves Sandy*. Note that the chart is topologically sorted, i.e., it is guaranteed that edge ids are introduced before they are referred to in the immediate daughters list. For instance, edge 13 refers to edge 11 (i.e., originating from a unary rule), edge 30 to edges 17 and 28 (a binary rule), etc. The edges 1–11 can be thought of as realizing the lexicon lookup. Notice that *loves* is assigned nine readings (different senses & morphosyntactic variations). The unary rules 12, 13, 15, 19, and 22 are morpholexical rules. The unary rule proper_np (17, 18) raises a noun to an NP. Some passive edges (40–42) result in useless CF rules for this sentence, since they represent dead branches in the search space.

```
SYNSEM|LOCAL|CAT|HEAD
SYNSEM|LOCAL|CAT|MC
SYNSEM|NON-LOCAL|QUE
SYNSEM|LOCAL|CONJ
SYNSEM|LOCAL|CAT|HEAD|MOD
SYNSEM|LOCAL|KEYS|KEY
SYNSEM|LOCAL|CAT|VAL|COMPS|FIRST|OPT
SYNSEM|NON-LOCAL|SLASH|LIST
SYNSEM|LOCAL|CAT|HEAD|MOD|FIRST|LOCAL|CAT|HEAD
SYNSEM|LOCAL|CAT|HEAD|VFORM
```

Fig. 3. The ten most prominent failure points for the English HPSG grammar (June 2002) in decreasing order.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

```
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;
```

Fig. 4. The overall structure of the extraction algorithm.

We note here that not every QC path has to be present in every feature structure, since only certain features are appropriate for certain TFSs. Let us give an example. Given the above set of QC paths, it turns out that the original head-complement rule `hcomp` is undefined for the ninth QC path. To account for this problem, we introduce a type U to express such undefinedness. U clearly only unifies with the most general type $\top$, and of course with itself. Using concrete syntax, we write `*undef*`. Consequently, we obtained instantiations of the binary head-complement rule schema `hcomp`, schema 1 in Pollard and Sag (1994), such as

```
hcomp[verb*, na, 0-dlist, ... , *undef*, bse] →
  bse_verb_infl_rule[ ... ] bare_np[ ... ]
```

To circumvent such undef values, it might be a good idea to work with different annotation vectors for each primary category symbol. This means that we have to partition the set of rule names $R = R_1 \cup \ldots R_n$, such that each R_i $(1 \leq i \leq n)$ is associated with a set of defined quick-check paths. We have not tried this strategy yet.

3.3 Algorithm

The idea behind the context-free extraction is relatively straightforward and is given in pseudo code in Figure 4.

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, encoded in the format given by Figure 1. The extraction *HPSG2CFG* then works as follows. Given a directory D and a vector of quick check paths Q, we iterate over the files in D (line 3). For each file, we then construct a vector edges of internal edges (i.e., JAVA objects) for the set of external passive edges stored in this file, using *makeEdges* (line 4). This includes the in-memory reconstruction of

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

the TFSs for the mother structures (the LHSs). For each vector position, i.e., for each edge e, we build up a LHS symbol (i.e., a pair, see section 2.1), consisting of a name field (via *getName*) and an annotation vector (via *getAnnotation*), given the quick-check 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 object (see section 2.2), 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. That is, 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 quick-check paths into account, we obtain more specific CF grammars, consisting of more rules. In the next subsection, we will slightly modify line 14, replacing the rule equivalence test by rule subsumption.

3.4 A variation

Rule subsumption, defined in section 2.2, now comes into play to potentially scale down generated grammars. We apply this operation online during the extraction process in that we replace line 14 of the algorithm by

14 $\quad \mathsf{result} = \mathsf{result} \cup_{\preceq} \{\mathsf{lhs} \rightarrow \mathsf{rhs}\};$

The intention behind $\cup_{\preceq}$ is that a new rule is only added to the result set iff it is more general than at least one rule already in the set. If so, the old rule and perhaps further other rules are removed. If it is more specific, the new rule is clearly not added. The application of that operation guarantees that the rules from the result set result are pairwise incomparable, i.e.,

$$\forall \alpha, \beta \in \mathsf{result}.\ \alpha \bowtie \beta, \text{ for all } \alpha \neq \beta$$

$\cup_{\preceq}$ is somewhat related to the specialized union operation $\cup_{\sqsubseteq}$ in Kiefer and Krieger (2002). Whereas $\cup_{\sqsubseteq}$ operates over typed feature structures representing context-free symbols, our operation is directly applied to annotated CF rules.

Of course, $\cup_{\preceq}$ does change the context-free language $\mathscr{L}_{\preceq}$ when compared to the language $\mathscr{L}_{\equiv}$, resulting from the application of $\cup_{\equiv}$:

$$\mathscr{L}_{\equiv} \subseteq \mathscr{L}_{\preceq}$$

Given the number of rules for the two grammars $\mathscr{G}_{\preceq}$ and $\mathscr{G}_{\equiv}$, we have

$$| \mathscr{G}_{\preceq} | \leq | \mathscr{G}_{\equiv} |$$

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

A simple example clearly shows this. Assume that the temporary result set contains the following three CF rules

$$\text{result} = \left\{ \begin{array}{l} \alpha = A \rightarrow \ldots B_1 \ldots \\ \beta_1 = B_1 \rightarrow \ldots \\ \beta_2 = B_2 \rightarrow \ldots \\ \ldots \end{array} \right\}$$

and assume that the new rule

$$\beta = B \rightarrow \ldots$$

subsumes both β_1 and β_2. $\cup_{\leq}$ would thus delete β_1 and β_2 from the result set and will add β to it. Furthermore, since B_1 and B_2 are no longer valid (β_1 and β_2 have been deleted!), we must replace every occurrence of B_1 and B_2 by the new nonterminal B, introduced in production β. This, however, has the effect that at least the modified rule ($\alpha' = A \rightarrow \ldots B \ldots$), derived from α, overgenerates.

Our approach keeps track of such rule deletions by implementing a symbol subsumption maintenance graph. In the above example, we establish two associations between B_1, B_2 and B: $(B_1 \mapsto B)$, $(B_2 \mapsto B)$. Given the example, it is possible that the new rule β might even be deleted by a newer, more general rule $B' \rightarrow \ldots$ later. In this case, we have to further specify a new association: $(B \mapsto B')$.

In the end, such substitution chains will be dereferenced, so that we can immediately substitute a dead RHS symbol by its correct and existing LHS counterpart. In the example, for instance, we must then know that B_1 should not be substituted by B, but instead by B'.

We also have to make associations for the converse case – if a new special rule γ is not added due to an existing more general rule γ', we must record this fact by creating the association $(L(\gamma) \mapsto L(\gamma'))$, since $L(\gamma)$ might occur on the RHS of final CF rules.

We note here that the substitutions are of course not restricted to the LHS symbols only. Consider the following three toy CF rules which we might have acquired so far

$$\text{result} = \left\{ \begin{array}{l} \alpha = \texttt{N[sg,fem]} \rightarrow \texttt{"Mary"} \\ \beta = \texttt{NP[sg,fem]} \rightarrow \texttt{N[sg,fem]} \\ \gamma = \texttt{S[...]} \rightarrow \texttt{NP[sg,fem] VP[...]} \\ \ldots \end{array} \right\}$$

and assume that the new, more general rule β'

$$\beta' = \texttt{NP[num,fem]} \rightarrow \texttt{N[num,fem]}$$

now comes in, substituting β. The result set then changes to

$$\text{result} = \left\{ \begin{array}{l} \alpha' = \texttt{N[num,fem]} \rightarrow \texttt{"Mary"} \\ \beta' = \texttt{NP[num,fem]} \rightarrow \texttt{N[num,fem]} \\ \gamma' = \texttt{S[...]} \rightarrow \texttt{NP[num,fem] VP[...]} \\ \ldots \end{array} \right\}$$

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

Since we substitute the dead symbols at the very end of the approximation, the resulting CFG is clearly not optimal, i.e., not minimal. A proper treatment here would require that we have to update the symbol substitution graph (and potentially perform substitutions) each time a new passive edge is checked against the temporary CF rule set. Since we might process several millions of edges during the approximation of a grammar, we do not apply this technique at the moment.

At this point of our investigation, we already note here that favoring rule subsumption does not have any significant advantage over rule equivalence (about 10% less rules only), but mostly disadvantages (complex handling of dead symbols and rule frequencies, worst running time and space consumption of the approximation, overgeneration). Our measurements in section 6 show that even for the largest English HPSG and for two different huge training corpora, the number of approximated CF rules under rule equivalence can still be handled by a properly implemented CF parser (less than 60,000 rules). It is worth mentioning that the closely related subsumption operation $\cup_{\sqsubseteq}$ in Kiefer and Krieger (2002) was ultimately necessary to made their purely grammar-driven approximation tractable. Kiefer and Krieger noticed that the use of equivalence had led to a massive explosion of CF rules in their setting. We were pretty sure to obtain similar results, but luckily rule equivalence gave us the best results and is nevertheless tractable.[4] We still hope that practical UBGs exist which will take advantage of rule subsumption. One can easily imagine a synthetic unification grammar which result in a CFG under rule subsumption that has exponentially-less rules than another CFG, obtained under equivalence from the same UBG.

3.5 Computing start productions

One point in the algorithm is still missing, viz., the generation of start productions. We have decided to employ only a single synthetic start symbol s in our grammars. This symbol has to be fresh, i.e., for all symbols $\langle n, a\rangle$ of the extracted grammar, we demand that $N(\mathsf{s}) \neq n$. In the implementation, the user must specify a non-empty list of start (or root) types, types which subsume original rule definitions, and thus subsumes potential rule instantiations. These types specify wellformedness conditions that a feature structure must satisfy to be a legal utterance (e.g., empty subcategorization list). Now let T be the set of all start types, R the set of extracted CF rules so far, and s the new top-level start symbol.

ComputeStartProductions(T, R, s) :$\Longleftrightarrow$
  local $\mathsf{S} = \emptyset$;
  for each type $\in \mathsf{T}$ **do**
    for each $\alpha \in \mathsf{R}$ **do**
      if type $\succeq N(L(\alpha))$

[4] The reason why Kiefer and Krieger's operation was so 'successful' in reducing the number of TFSs probably comes from the fact that it merely compares TFSs, whereas rule subsumption in our approach must take more than one TFS into account (LHS TFS plus RHS TFSs); recall the definition 5.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

```
        S = S ∪ {s → L(α)};
    end for;
end for;
return S;
```

In the English ERG/LinGO HPSG grammar developed at CSLI, sentential phrases are subsumed by the type *root_strict*, and thus a start symbol in the extracted CFG can be determined by checking whether the name of a LHS symbol is subsumed by this start type. Non-sentential saturated phrases in ERG (e.g., PPs and NPs) are characterized by *root_phr*, thus if we want find all saturated phrases in the approximated CFG, we have to declare these two types to be start types.

3.6 Rule frequency

As we said in section 2.2, rule objects possess a frequency field which will gain importance if we are interested in a simple PCFG model which predicts probability distributions over CFG derivations. Since the counter is updated during the approximation process, we obtain a very basic and indirect disambiguation of UBGs for free, exploiting the fact, that ranked CF trees can be deterministically replayed by a deep unification parser; see Kiefer *et al.* (2002).

Exactly the frequency counter is set to 1 during the initialization of a rule object and is incremented by 1 in case a structural equivalent rule has been detected.

Concerning the frequency field f of a new rule α that has replaced more specific rules $\alpha_1, \ldots, \alpha_n$, we have

$$f(\alpha) := \left(\sum_{i=1}^{n} f(\alpha_i) \right) + 1, \quad \text{for all } \alpha_i \prec \alpha$$

since the more general rule now acts as a representative for the deleted specialized rules. Assuming the contrary, the frequency counters of truly general rules α_i ($1 \leq i \leq n$) are incremented by 1 when penalizing α:

$$f(\alpha_i) := f(\alpha_i) + 1, \quad \text{for all } \alpha_i \succeq \alpha$$

Considering the unary start productions from section 3.5, the frequency counter of a start production $s \rightarrow l$ is set to the sum of the frequencies of those rules $\alpha \in R$, whose LHS is exactly l, given the set of all CF rules R:

$$f(s \rightarrow l) := \sum_{\alpha \in R} f(\alpha), \quad \text{where } \alpha = l \rightarrow r_1 \ldots r_n$$

To obtain a proper probability distribution, we have to normalize the rule frequency counter in the standard way (Charniak 1993). Now let $R(l) := \{l \rightarrow r_{i_1} \ldots r_{i_n} \mid i, n \in \mathbf{N}\}$ be the set of all extracted CF rules sharing the same LHS l, $f(\alpha)$ be the value of the frequency counter for rule $\alpha \in R(l)$, and $F(l) := \sum_{\alpha \in R(l)} f(\alpha)$. We then compute the probability $p(\alpha)$ for a context-free rule $\alpha = l \rightarrow r_1 \ldots r_n$ as

$$p(\alpha) := \frac{f(\alpha)}{F(l)}$$

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

which gives us

$$p(R(l)) := \sum_{\alpha \in R(l)} p(\alpha) = 1$$

As we said above, by using the likelihood of a rule, we obtain a simple PCFG model for free. At this moment of writing, we have not tested its disambiguation capability, but it is very likely that it outperforms an oracle which *randomly* chooses a CF rule in case more than one rule is applicable. We want to make some further remarks here.

Firstly, such a model should usually take only complete HPSG derivations into account, i.e., only sets of deep passive edges which fully cover the input and which are true readings. But because we are very practically interested to have a robust CFG, we have decided for every passive edge of the deep parser.

Secondly, obtaining lexicalized PCFGs from the approximated CFGs is not hard, since we can easily stuff word-specific information into the context-free symbols (with the disadvantage of having more rules, of course).

Thirdly, as an extension of the last point, CFGs (and so PCFGs) in our setting can embody dependecies that go beyond the capabilities of hand-written and trained (P)CFGs, reminding us of N-gram models and extensions thereof. This is due to the fact that the deep passive edges in the unification parser capture (unbound) dependencies (`SUBCAT`, `SLASH`), so that specific instantiations of such constructions can show up in the annotated CF symbols of the rules of the approximated grammars (if this is desired).

Finally, it is clear that the above simple statistical model can be replaced by a more advanced model, using, e.g., the inside-outside algorithm for unsupervised training of PCFGs (Lari and Young 1990; Pereira and Schabes 1992). Kiefer *et al.* (2002) have shown that the disambiguation quality of approximated grammars clearly benefit from such a treatment.[5]

4 Other approaches

In this section, we present several other approaches to compilation and approximation, some of them *sound* (Pereira and Wright 1991), that is, the language generated by the resulting grammar is a superset of the language of the original grammar, others *unsound*.

Goldstein (1988) was the first who converted an HPSG to a CFG. Goldstein, however, made several simplified assumptions: (i) English is context-free; (ii) the number of categories/feature structures is finite; (iii) complex CF (non)terminals are not allowed; (iv) in the early development stage of HPSG, no coreferences and types are used. However, rules are underspecified and lexicalization is already present. This version of HPSG is essentially a modified GPSG. Since Goldstein did

[5] We have not yet carried out experiments in this direction, but would like to take advantage of the LinGO Redwoods treebank (`http://redwoods.stanford.edu/`) to enrich the approximated grammars with a probabilistic model for the disambiguation of readings.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

not address the (possible) accumulation of information (e.g., under SUBCAT, CONT, or DTRS), a restrictor is clearly not needed. Rule instantiation is realized through an active bottom-up parser, using the chart to represent the feature structures.

Carroll (1993) reported in his PhD thesis on the extraction of CF grammars from unification-based grammars, written in the ANLT (Alvey Natural Language Tools) grammar formalism. Grammars in ANLT usually consist of several hundred of relatively specific GPSGish rules with features having only finitely many values. Categories (= sets of feature-value pairs) can be defined in ANLT, but are not explicitly arranged in a type hierarchy. A single ANLT unification rule is mapped one-to-one onto a CF rule by introducing atomic symbols on the CF side that abstract from categories in the ANLT grammar. To avoid subsumption over CF symbols, subsuming categories are mapped onto a representative that encodes the least upper bound, similar to our operation $\cup_{\preceq}$. A related approach is described in Nakazawa (1995).

Diagne, Kaspar and Krieger (1995) and Kasper and Krieger (1996) present a distributed parsing approach that is distinguished by the use of a very restricted HPSG whose derivation trees are reparsed deterministically (in fact, in parallel) by the original HPSG. The two parsers mutually restrict their search space, using a specialized protocol.

Kasper *et al.* (1996) employ the same idea, but substitute the restricted HPSG through the (relatively specific) context-free backbone of an HPSG-like grammar. By using the pure backbone and a corpus, a PCFG is trained and used online in order to obtain the *n*-best paths of a word lattice. The collection of these paths constitutes the initial chart of the second parser that uses the HPSG essentially for semantic construction.

Neumann and Flickinger (1999) and Neumann (2003) describe an approach that obtains a stochastic lexicalized tree grammar (SLTG) for a given corpus. The idea here is that the training corpus is parsed using an HPSG grammar and an HPSG parser, and derivation trees are iteratively decomposed top-down, resulting in nonterminal nonheaded subtrees, where the cutting points are marked for later substitution. SLTGs are processed by an LTAG-like parser in a two-step process, consisting of an initial all-path parsing phase, followed by the application of the relevant HPSG feature constraints. This idea is related to LFG parsing, but has the clear advantage that a larger tree context is involved (although SLTGs are of context-free power). Since the extraction process of the SLTG grammar works on the derivation trees of a tree bank, viz., tsdb (Oepen and Flickinger 1998), the node labels are relative coarse generalizations of the information embodied in the feature structures used during HPSG parsing. The vagueness or underspecification of the node labels, however, is partly compensated by the larger tree context. The approach furthermore applies two postprocessing techniques: a linguistically-motivated decomposition of trees and a specialization of node labels.

Moore (1999) describes a compilation method that turns unification grammars with finitely-valued features into context-free grammars. The grammar, he reports on, is written in the core language engine formalism of SRI Cambridge, and consists of about 900 relatively specific phrasal rules. Moore only consider finitely-valued

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

features. In order to avoid the combinatorial explosion of rules, Moore only instantiates those features in daughter categories that are constrained by the unification rule and considers only combinations of feature values by unifying active and passive edges, as in a bottom-up active chart parser. He also throws away useless rules. Moore's paper presents a new left recursion elimination algorithm, specifically tuned for the grammar formalism GSL of the Nuance speech recognizer (Nuance 2004), taking advantage of the fact that GSL allows regular expressions. Moore also considers various combinations of acoustic and symbolic information in a language model and is a great overview paper. In a certain sense, Moore's approach is the forerunner for Rayner *et al.* and Dowding *et al.*, laying the foundations for the more sophisticated enumerative compilation techniques below. Moore's expansion technique will probably not work for lexicalized theories, such as HPSG.

Kiefer and Krieger (2000, 2002, 2004) present a sound approximation method that turns unification-based grammars, such as HPSG or PATR-II into context-free grammars. The method does not rely on a corpus, but is purely grammar-driven. In an initial phase, the method generalizes the set of all lexicon entries, by abstracting from word-specific information. The abstraction is specified by means of a restrictor. After that, the grammar rules are instantiated by unification, using the abstracted lexicon entries and resulting in derivation trees of depth 1. A rule restrictor is applied 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 results in TFSs encoding a derivation, and again the TFSs are restricted. The iteration is proceeded until a fixpoint is reached, meaning that further iteration steps will not compute additional information. Given the TFSs from the fixpoint, it is then easy to generate context-free productions, using the full feature structures as symbols of the CFG. The speedup factor for the *aged* and *eng2000* testsuites within a two-stage parsing architecture are between 1.7–2.7.

In Kiefer *et al.* (2002), an extension of the method is presented which easily allows the disambiguation of UBG readings, by indirectly relying on a trained PCFG, derived from the approximated CFG. Considering a random baseline of 72% for the exact match task, the method shows an increase of 16% (= 88% precision).

In Cancedda and Samuelsson (2000), a corpus-based specialization method is introduced which directly operates on rules written in the LFG framework. Because the LFG formalism allows RHSs of grammar rules to consist of regular expressions (REs), the idea of this framework is to expand RHSs into RE-free sequences of symbols, guided by the training data. Although Kleene star and complementation in REs as well as specialized operators like *shuffle* may introduce spurious ambiguities, such simplified rule instantiations clearly speed up parsing (up to a factor 6). The downside of this method is that one might lose coverage (about 13%). To compensate for the loss in coverage, a two-stage parsing architecture is proposed in which a second stage, consisting of the original grammar, is only invoked in case the

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

first specialized parser failed. Even with this backup mechanism, a speedup between 1.8–2.7 was obtained.

Dowding *et al.* (2001) compare the approach to grammar approximation in (Moore 1999) to that in Kiefer and Krieger (2000). As a basis for the comparison, they choose a command-and-control grammar written in the Gemini/CLE formalism. The motivation for this enterprise comes from the use of the resulting CFG as a context-free language model for the Nuance speech recognizer (Nuance 2004). The measurements in (Dowding et al. 2001) differ from those later conducted in Kiefer and Krieger (2002, 2004). Unfortunately, it is not clear why Dowding's implementation of Kiefer and Krieger's method comes off so badly. Dowding report on an average ambiguity per sentence of 15.4, whereas Kiefer and Krieger comes up with only 1.49 (ambiguity rate for the original grammar was 1.44). The compilation time to obtain the approximated grammars differs too: 11 minutes vs. 34 seconds. With a slightly different setting, Kiefer and Krieger even obtain a correct approximation, showing that the UBG is in fact of only context-free power. The paper by Dowding *et al.* (2001) also explores techniques for transforming CFGs into weakly equivalent grammars with less ambiguity. These investigations are important since the use of linguistically-motivated CFGs as language models often lead to confidential paths in the language model labeled with the same recognition hypothesis, so that other good hypotheses are forced out of the beam of the recognizer.

Rayner *et al.* (2000) conduct a series of experiments that employ approximated CFGs as language models in the Nuance speech recognizer. Instead of using a domain-specific UBG, Rayner *et al.* (2000) start with a general, linguistically-motivated grammar, but use a domain-specific lexicon, resulting in a domain-specific CFG. Compiling a UBG is done by enumerating all possible instantiations of features in rules. To make this approach tractable, a relatively complex mechanism is implemented to arrive at features, having only finitely-many values, similar to the approach in Moore (1999). A kind of rule folding is also applied here. The compilation failed for a mid-to-large-size UBG, so they started with a small grammars and incrementally add further rules and lexicon entries. It turned out that a more complex clause structure was not responsible for the poor recognition performance of the resulting CFGs, but instead a small number of rules, basically centered around relative clause modification. The findings in this detailed paper might also be of interest to our method. In Rayner *et al.* (2001a), the explicit assumption is made that each UBG feature has a finite range of possible values. How this can be guaranteed is not explained in the paper. Further technical aspects of the above compilation method are elaborated. In the related paper (Rayner et al. 2001b), it is argued that agreement constraints from the UBG should be kept in the CFG to obtain better language models.

Bos (2002) comes up with a compilation method that is very related to that of Moore and Rayner *et al.* above. Again, features only have finitely many values and complex values are not allowed. What makes his approach unique, however, is that he shows how to transfer compositional semantics from the UBG into the CFG, using the grammar specification language GSL from the Nuance recognizer package.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

GSL supports slot filling for constructing semantically-relevant output, similar to W3C's VoiceXML or SRGS, and Sun's JSGF. The challenge for saving the semantics on the CFG side is elimination of left recursion in the rules and no support for features and unification. In the end, deriving a logical form for a string is achieved by running the CF parser on that string, using the approximated CFG, followed by several β-reduction steps. Recognition performance in terms of speed and word error rate is not affected by his method. Bos' way to achieve a compositional semantics is quite close to our proposed treatment of semantic construction, using additional semantic rules (see section 5.4).

In this section, we have listed existing frameworks to grammar approximation and have tried to relate our approach to some of them (if this was possible). For instance, Carroll's use of symbol subsumption or the idea of applying postprocessing steps (as done in, e.g., Moore's or Dowding's framework). What makes our approach unique is that it operates on the most-specific information made available by a deep grammar, viz., the charts of a deep parser that are enriched by typed feature structures (a kind of partial evaluation). In order to obtain proper CF recognition grammars, we have adopted the concept of quick-check paths or quick-check filtering in order to equip context-free symbols with the most prominent failure points. And because we have access to the feature structures of a chart (and so to information stored in arbitrary deep-nested daughters), we can even escape the flat domain of CF rules (we haven not tried this yet).

5 Summary and extensions

As already explained in the article, the corpus-driven approximation method is *unsound*, that is, given a corpus C of training sentences and a set of annotation paths A (from which we determine the annotation of a context-free symbol), the approximated CF language is usually *not* a superset (but also *not* a subset) of the language accepted by the HPSG (see Figure 5). This is due to the fact that not all linguistic constructions licensed by the HPSG are covered by the training corpus, but also because not every piece of information from the TFS is encoded in the annotation of a CF symbol:

$$\forall\, C, A\,.\, \mathscr{L}(\text{HPSG}) \not\subseteq \mathscr{L}(\text{CFG}, C, A)$$

It is easy to see that more training samples C' result in a broader language:

$$\forall\, C', C \subseteq C'\,.\, \mathscr{L}(\text{CFG}, C, A) \subseteq \mathscr{L}(\text{CFG}, C', A)$$

The subset relation turns around with more annotations:

$$\forall\, A', A \subseteq A'\,.\, \mathscr{L}(\text{CFG}, C, A) \supseteq \mathscr{L}(\text{CFG}, C, A')$$

In both cases, more training sentences and/or more annotation paths result in larger rule set. Overall, we can say that the more information from the feature structure is put into the annotations of the context-free symbols, the better the CFG approaches the HPSG in terms of the number of readings and the less it overgenerates w.r.t. linguistic constructions contained in the training corpus.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

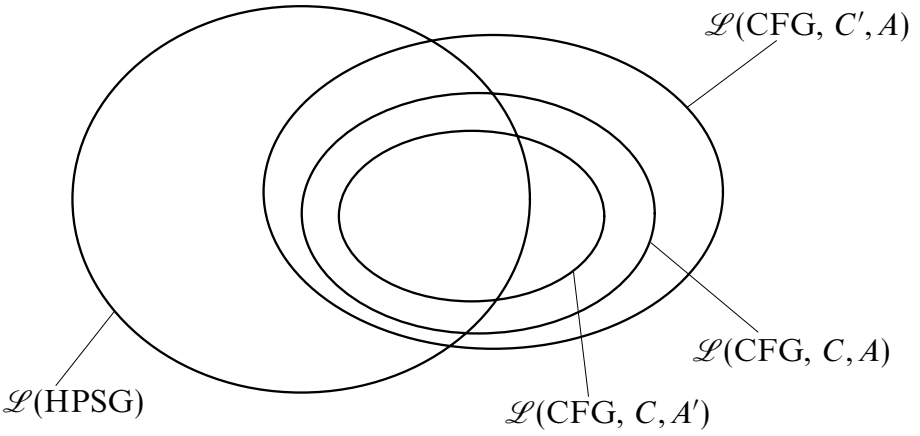


Fig. 5. The correlation between the language accepted by the HPSG $\mathscr{L}$(HPSG) and the approximated context-free language $\mathscr{L}$(CFG, C, A), given a corpus C and a set of annotation paths A. The approximated language $\mathscr{L}$(CFG, C', A) monotonically grows when given a larger corpus $C' \supseteq C$. The language $\mathscr{L}$(CFG, C, A') usually shrinks when given a larger set of annotation paths $A' \supseteq A$ (but the *number* of rules grows).

Remember, annotated CF rules approximate HPSG (passive) chart edges, and the existence of more specific CF symbols and rules helps to better mimic the behavior of the HPSG during CF parsing. Finding the right annotation paths, of course, depends on the application domain in which the extracted CFG is employed.

The correlation between UBGs and the family of approximated CFGs w.r.t. a given corpus and a set of annotation paths is depicted in Figure 5.

As motivated in section 3.4, when fixing a corpus C and a set of annotation paths A, the language obtained under $\cup_{\equiv}$ is always a subset of the language resulting from the application of $\cup_{\leq}$, given the same HPSG source grammar:

$$\mathscr{L}_{\equiv}(C, A) \subseteq \mathscr{L}_{\leq}(C, A)$$

However, the more general grammar has less rules due to the fact that $\cup_{\leq}$ might delete more than one specialized rule when favoring a more general rule.

We explained in the beginning that we have taken *all* passive edges of the HPSG parser into account – this is a compromise. Two extremes obviously stand out: (1) using only passive edges which have led to complete HPSG readings and (2) taking all, i.e., active *and* passive edges into account. Approach 1 clearly generates a smaller, but more approaching language when compared to the language of the HPSG source grammar. Approach 2 leads to a much broader language due to the fact that active edges contain uninstantiated daughter positions which are usually not much restricted by the rule schemata in HPSG.

Another avenue which is worth to follow results from widening our domain of locality by moving from context-free rules to trees of depth greater than 1. This is not hard to achieve since the transmitted chart of the HPSG parser not only allows us to refer to the immediate daughters of an edge, but even to every daughter

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

covered by this edge. Our open framework encourages experimentation towards this direction.

We will now discuss several postprocessing steps which might be applied after a CFG has been extracted.

5.1 Elimination of useless rules

Since the extracted annotated grammars are still context-free, we can remove those symbols which will never contribute to a reading. These symbols and the productions using them are called *useless* and there exist a fast decidable algorithm that, given a grammar, produces a smaller grammar which does not contain useless productions (Hopcroft and Ullman 1979). The runtime efficiency of a CF parser clearly benefits from the removal of such useless productions, helping the parser to avoid dead derivation branches. Note that, in general, unsuccessful rule instantiations in the unification parser can *not* be computed beforehand, due to the infinite number of 'categories' in a unification-based grammar (the set of all feature structures produced by a UBG is usually infinite).

5.2 Automatic lexicon extension

New utterances not covered by the corpus are only recognized by the extracted CFG if they can be mapped to already parsed syntactic constructions *and* to already seen words. One can imagine that the relevant domain-specific syntactic constructions can be covered by a sufficiently large training corpus. Lexical gaps, however, should not be attacked by taking more samples into account. One way to enlarge the CF lexicon is by mapping lexicon entries from the HPSG to related terminal symbols in the CFG, originating from parsing the training corpus. Assume, the training corpus contains

Bill gives Mary a book

but does not mention the word *cookie*. A sentence such as

Bill gives Mary a cookie

will then not be recognized.

The extraction process has represented *book* by the unary production

`book_n1[...]` → `"book"`

where `book_n1` can be seen as a preterminal, speaking in terms of CFGs.

By inspecting the HPSG lexicon, we will then find that *book* is of the same lexical type as *cookie*, viz., `book_n1`. Thus we can enlarge the CF lexicon by

`cookie_n1[...]` → `"cookie"`

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

By exploiting this procedure, we can at least guarantee that the CFG will have the same lexical coverage than the UBG.[6]

5.3 Grammar postprocessing

Lexicon extension is related to a postprocessing step which "massages" the approximated CFGs, more precisely, the rules which have a preterminal symbol such as `in[prep*, ...]` on their RHS. For instance, we might have

```
in[prep*, ...] → "in"
hcomp[prep*, ...] → in[prep*, ...] proper_np[noun_or_nomger, ...]
```

Now assume that our training corpus does not contain the preposition *on* and that automatic lexicon extension has added the following "lexicon lookup" (or lexical) rule

```
on[prep*, ...] → "on"
```

However, we are still missing the very likely CF rule

```
hcomp[prep*, ...] → on[prep*, ...] proper_np[noun_or_nomger, ...]
```

Constructing such new rules, however, is not that difficult. Clearly, when addressing this topic properly, the need for a larger training corpus is probably no longer that demanding.

A much simpler way to achieve more generality is, however, to omit the primary category symbol, so that the above example reduces to

```
_[prep*, ...] → "in" | "on" | ...
hcomp[prep*, ...] → _[prep*, ...] proper_np[noun_or_nomger, ...]
```

Clearly, this strategy can also be applied to every nonterminal, resulting in smaller grammars, since a lot of rules will probably collapse. The results of deleting the primary category symbol of each nonterminal are presented for a small test corpus in section 6.3.

[6] The latest version of the LKB system now supports lexicons stored in a relational DB via Postgres (thanks to Ben Waldron). Given the DB, all fullforms can be generated, and by passing these fullforms to PET as a single (ungrammatical) sentence, we obtain the passive edges for each word together with the corresponding feature structures. From this chart, we then construct the lexicon lookup rules as explained before. We have applied this idea during our measurements (cf. section 6.3) in order to rule out lexical gaps in the CF stage.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

5.4 Constructing meaningful output

In case we are not only interested in a context-free boolean recognizer, but would like to see some useful output, e.g., MRS-like structures (Copestake, Lascarides and Flickinger 2001), we can either apply two-stage parsing here (see above), letting the UBG deterministically replay the CF parse trees, or (automatically) equip the approximated CFGs by 'semantic rules', similar to attribute grammars, a framework extensively used in syntax-directed translation, dating back to the early sixties (Aho, Sethi and Ullman 1986).[7] The latter approach completely obviates UBG parsing and has the further advantage that overgeneration within the extracted CFG will not be prohibited by a subsequent, more restrictive UBG reparse.

Such semantic rules can *not* be obtained directly from the passive edges of a chart (or even from the extracted CF rules), but instead must be reconstructed from the CF rules with the help of the HPSG rule schemata, together with the total surjective mapping from CF rules to HPSG rule schemata.

Let us give an example. Assume that the training corpus contains the sentence

Tigger meows

for which we obtain, among other rules,

$$\texttt{subjh[verb,...]} \rightarrow \texttt{proper_np[noun,...]}\ \ \texttt{verb_infl_rule[verb,...]}$$

The unique (slightly simplified) HPSG rule schema associated with the LHS of `subjh[verb, ...]` plus the inherited HPSG principles gives us a feature structure which exactly specifies the 'transport' of information from the daughters to the mother ($\oplus$ denotes list append):

$$\begin{bmatrix} \texttt{SYNSEM|LOCAL|CONT} & \begin{bmatrix} \texttt{LISZT} & \boxed{1} \oplus \boxed{2} \\ \texttt{INDEX} & \textbf{event} \end{bmatrix} \\ \texttt{HEAD-DTR|SYNSEM|LOCAL|CONT|LISZT} & \boxed{2} \\ \texttt{NON-HEAD-DTR|SYNSEM|LOCAL|CONT|LISZT} & \boxed{1} \\ \ldots\ldots & \end{bmatrix}$$

This transport is expressed through coreferences ($\boxed{1}$ and $\boxed{2}$) and has to be remodeled in the semantic rules attached to the CF rule. We thus enrich the above extracted CF production by the following two simplified semantic rules (we use a kind of PATR-II notation, where 0 denotes the LHS, 1 the first RHS daughter, etc.):

$$\texttt{0.SEM.LISZT} := \texttt{1.SEM.LISZT} \oplus \texttt{2.SEM.LISZT}$$
$$\texttt{0.SEM.INDEX} := \texttt{event}$$

The process of semantic composition must be grounded in the lexicon, thus we equip each lexicon entry of the CFG by the relevant semantic rules which can be obtained from the corresponding lexicon entries of the HPSG (let $\langle \ldots \rangle$ denotes a

[7] Such semantic rules nowadays gain importance in the speech grammar community, most notably in the *Semantic Interpretation for Speech Recognition* (Van Tichelen 2003) framework for the *Speech Recognition Grammar Specification* (Hunt and McGlashan 2004).

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

list of elements):

$$\texttt{Tigger[...]} \rightarrow \texttt{"Tigger"}$$
$$\texttt{0.SEM.LISZT} := \langle \texttt{Tigger} \rangle$$

But perhaps the much simpler procedure of annotating CF symbols with additional, semantically-relevant information from the feature structures might suffice. In this case, successful CF derivation trees will be reinspected in a second phase, where the according semantic information is reconstructed. This approach could also be the starting point for approximating not only recognition grammars, but even useful generation grammars.

6 CSLI's English Resource Grammar (LinGO)

We have applied our method to the large English LinGO grammar (June 2002), developed at CSLI Stanford. The grammar consists of 61 rule schemata, 8,082 non-leaf types, and a lexicon of 6,930 stems.

6.1 aged

We used the *aged* test suite (Oepen and Callmeier 2000), consisting of 96 syntactically highly diverse sentences to measure the quality of our approximation (average sentence length: 8.4; maximal length: 19). *aged* consists of 202 stems that cover a great deal of morphological and lexical variation. 719 full forms were computed from these stems. The PET parser which produced the input charts of passive edges (section 3.1) ran under no restrictions and computed 267,651 passive edges overall. One of the sentence from *aged* even contributed 139,028 edges. Approximation was performed under rule equivalence and rule subsumption. The length of the annotation vector was varied between 0 and 32.

Figure 6 shows the asymptotic behavior for the number of nonterminal and rules, when compared to the length of the annotation (see also Figure 3). The explosion of rules from five to six annotations is due to the path `SYNSEM|LOCAL|KEYS|KEY` who potentially results in more than 4,000 possible values, representing relatively word-specific information. It is usually a good idea *not* to use such path values as annotations in case the rule set gets too large. Furthermore, approximated CF rules will get too specific when incorporating such information. An optional strategy to cope with such a value overload is described in Kiefer and Krieger (2002), viz., type generalization.[8]

The next two tables present runtime measurement and show that the useless rules outweigh when moving to larger annotation vectors. The numbers also show (at least for *aged* and *csli*, see next section) that the grammars obtained under rule equivalence are nearly of the same size as those obtained under rule subsumption,

[8] Features such as `KEY`, `PHON`ology, or surface `FORM` are assigned values that are specific to certain words. In order to reduce such a high variance, we can generalize the critical values (which are types) up to certain points in the type hierarchy, nevertheless keeping still enough restrictions (instead of throwing them away). Since the approximated CF grammars are still of moderate size, we have not tried this strategy yet.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

#annotations	#E-rules	#S-rules	#associations	#substitutions
0	3,475	3,475	0	0
1	3,783	3,539	28	264
2	3,890	3,552	52	1,210
3	4,047	3,579	71	1,539
4	4,048	3,580	71	1,536
5	4,141	3,670	69	1,510
6	8,585	7,406	733	4,547
7	8,662	7,574	605	3,895
8	9,430	7,998	760	4,096
9	10,050	8,459	891	3,969
10	11,364	9,335	1,189	4,032
11	11,469	9,341	1,200	4,199
12	11,532	9,641	1,079	3,624
13	11,591	9,667	1,087	3,670
14	11,771	9,704	1,123	3,856
15	11,857	9,800	1,120	3,896
16	11,909	9,829	1,111	3,921
17	11,928	9,830	1,123	3,954
18	11,928	9,850	1,107	3,931
19	12,127	9,913	1,183	4,057
20	12,169	9,926	1,201	4,050
25	13,182	10,867	1,295	4,020
32	15,173	12,389	1,454	4,014

Fig. 6. Total number of rules for the *aged* test suite, obtained under both equivalence (E-rules) and subsumption (S-rules). The explosion in the number of rules from 5 to 6 is due to the path `SYNSEM|LOCAL|KEYS|KEY`.

#annot.	#useful	#useless	#ppedges	#readings	#errors	time[s]
0	3,453	22	122,311	$4.294E+29$	2	3.09
2	3,737	153	73,010	$6.527E+18$	3	0.72
5	3,943	198	64,364	$1.408E+17$	4	0.54
10	7,160	4,204	23,607	617,068	9	0.13
18	7,291	4,637	21,970	307,483	9	0.12
32	8,236	6,937	22,858	174,533	9	0.10
110	10,237	15,455	19,440	11,614	15	0.09
226	13,682	32,025	18,855	11,195	15	0.09

Fig. 7. Useful and useless rules for *aged*, obtained under rule equivalence (without lexicon lookup rules). All measurements were conducted on a 833 MHz Linux workstation under Allegro Common Lisp 6.0.1. ppedges refers to packed passive edges.

but are overall better when regarding the number of readings (#readings). This will clearly gain importance in the second stage of an UBG-replay approach. Considering the UBG parser, 1,589 readings for *aged* were found. PET failed for 17 of the 96 sentences when using the English HPSG.

A plot of the number of rules obtained under rule equivalence against the number of samples is shown in Figure 9.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

#annot.	#useful	#useless	#ppedges	#readings	#errors	time[s]
0	3,453	22	122,311	$4.294E+29$	2	3.09
2	3,418	134	68,027	$3.644e+19$	3	0.69
5	3,475	195	64,878	$2.303E+18$	4	0.65
10	7,022	2,313	31,509	3,202,651	9	0.14
18	7,080	2,770	25,369	1,617,231	9	0.13
32	7,391	4,998	23,569	326,149	9	0.13
110	8,919	12,694	19,347	17,004	15	0.09
226	12,824	30,205	18,262	13,253	15	0.09

Fig. 8. Useful and useless rules for *aged*, obtained under rule subsumption (without lexicon lookup rules). The runtime is measured for the first stage of two-step parsing, i.e., for the CF stage.

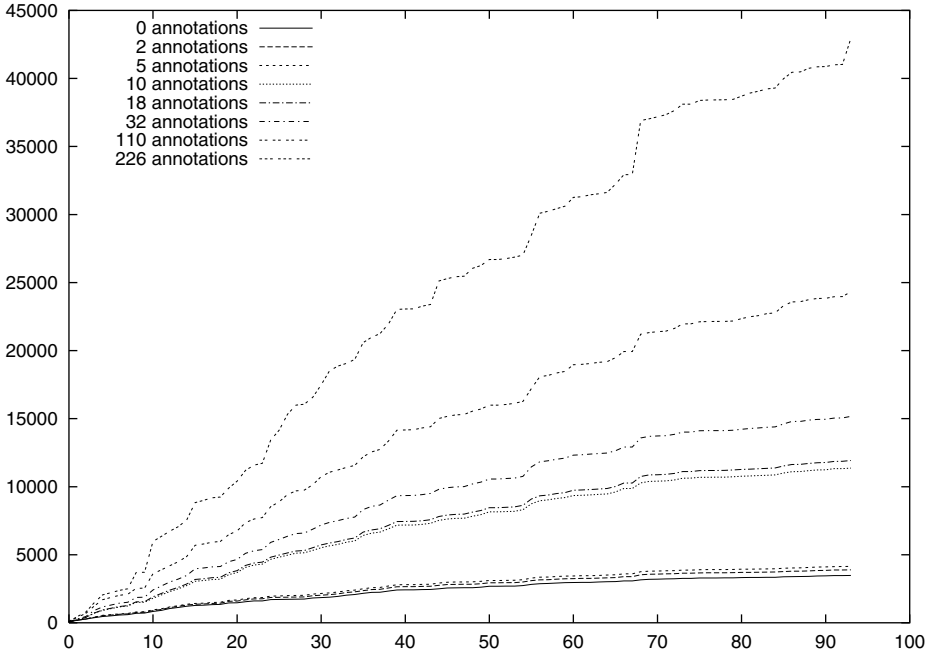


Fig. 9. Plot of the number of all rules obtained under rule equivalence against the number of samples from *aged*. Note that we depict the sum of useful and useless rules without start productions here. The table clearly shows that, at least for 226 symbol annotations, the training corpus is too small.

We also conducted measurements for two grammars under rule equivalence. The pure HPSG parser, fully equipped with quick-check filtering, resulted in an overall parse time of 1,361.5 seconds. Total time of two-stage parsing (CF parsing plus UBG replay using the full HPSG) for *aged* was

- **110 annotations**: 59.4 sec = 22.9× speedup
- **226 annotations**: 52.8 sec = 25.8× speedup

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

#annotations	#E-rules	#E-nonterms	#S-rules	#S-nonterms
0	11,432	3,372	11,432	3,372
2	12,845	3,561	11,736	3,473
5	14,007	3,796	12,252	3,637
10	52,848	16,508	41,530	11,056
18	54,376	17,212	43,203	11,623
32	65,808	24,373	54,875	17,940

Fig. 10. Number of rules (including start productions and lexicon lookup rules) and nonterminals for the *csli* test suite, obtained both under equivalence (E-rules, E-nonterms) and subsumption (S-rules, S-nonterms). The rule set both contains useful and useless rules.

Considering runtime, the grammar with 226 annotations was better, since HPSG replay had carried out less unifications.

6.2 csli

The *csli* test suite which originated from the old HP test suite developed at Hewlett Packard Labs (Oepen and Flickinger 1998), consists of 5,720 grammatical and ungrammatical samples. It is intended to cover a great deal of the syntactical constructions of English. Average sentence length is 5.0, maximal length 20. 585 full forms (= #CF terminal symbols), e.g., `"little"`, and 3269 lexicon lookup rules (= #preterminal symbols or lexical categories), e.g.,

```
little_det[det, na, 0-dlist, ...] → "little"
```

can be found in every approximated grammar (i.e., average ambiguity rate per stem ≈ 5.6). The maximal number of passive chart edges in the PET parser was set to 100,000. 7,981 readings were found by PET. 3,246 sentences of 5,720 could not be analyzed by PET, given the HPSG and the edge limitation. 91 sentences of *csli* contain a lexical gap not covered by the lexicon of the HPSG.

For *csli*, we obtained the following measurements for the approximated CFGs (see Figures 10 and 11).

Some of the numbers from Figure 11 were astounding on first sight. Firstly, the decrease of useful rules from 10 to 18 annotations seems to contradict to what has already been said before. However, we have always argued that the total number of *all* rules increases when moving to a larger annotation. This decrease is due to the fact that "older" useful rules have been outdated, i.e., have become useless through the specialization of some of their CF symbols (more annotations).

Secondly, the number of readings might increase when we equip the CF symbols with more information. Again, this does not stand in contrast to what has already been said. Clearly, a more specialized CF grammar (226 annotations) recognizes a smaller language than a more general CFG (110 annotations). This fact is supported by the number of errors in Figure 11 (more errors). "Older" CF rules, here, are

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

#annot.	#useful	#useless	#ppedges	#readings	#errors	time[s]
0	8,161	2	2,981,080	$6.492E+41$	356	86.12
2	9,138	438	2,116,619	$3.026E+32$	510	40.22
5	10,059	679	1,752,303	$2.464E+25$	627	26.02
10	**32,366**	17,383	589,711	241,665,239	1,744	4.66
18	**31,575**	19,702	545,788	56,629,513	1,898	4.63
32	32,354	30,489	534,236	38,030,870	1,920	4.20
110	40,742	79,520	436,342	**18,266**	**2,926**	4.08
226	58,394	187,755	377,187	**22,497**	**3,065**	4.12

Fig. 11. Useful and useless rules for *csli*, obtained under rule equivalence (without lexicon "lookup" rules). Less than a fourth of all rules are useful under the most specific CFG. Note the *decrease* of useful rules when moving from 10 to 18 annotations, although the total number of rules increases. Note also the *increase* of readings from 110 to 226 annotations, even though the recognized language gets smaller (cf. number of errors). Overgeneration against the UBG (7,981 readings) w.r.t. training corpus was between 2.29 and 2.82 for the two largest grammars. Recall that the original HPSG produces 3,246 errors which are properly approximated by the most specific CFG above (viz., 3,065). The runtime is measured for the first stage of two-step parsing, i.e., for the CF stage.

split up into new specialized instances when adding more annotations. In a certain sense, new information might add spurious ambiguities. Thus, the CFG with 110 annotations can be seen to better approximate the UBG than the bigger CFG. This behavior should be taken into consideration in the context of two-stage UBG replay parsing.

However, the measurements for two-stage parsing (110 and 226 annotations) are of a different kind and show that the restrictedness of the CFG (= number of errors) should be taken into account when judging the practical usefulness of a grammar. The grammar with 226 annotations per symbol leads to fewer unification in the UBG replay stage. HPSG baseline for the 5,720 sentences was 2,777.3 seconds (again, quick-check filtering was switched on).

- **110 annotations**: 30.7 sec = 90.5× speedup
- **226 annotations**: 28.5 sec = 97.4× speedup

This speedup of nearly two magnitudes shows the enormous potential of our method. We finish this section with a plot of the number of rules obtained under rule equivalence against the number of samples (Figure 12).

6.3 Putting it to the test

Similar to Kiefer *et al.* (2002), we chose a sample of sentences from *csli* as input to a test run. The corpus was divided into a training set (90% = 5148 sentences) and a test set (10% = 572 sentences) to measure the quality of the approximated CFG w.r.t. overgeneration, recognition of ungrammatical sentences, and speedup (see Figure 13).

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

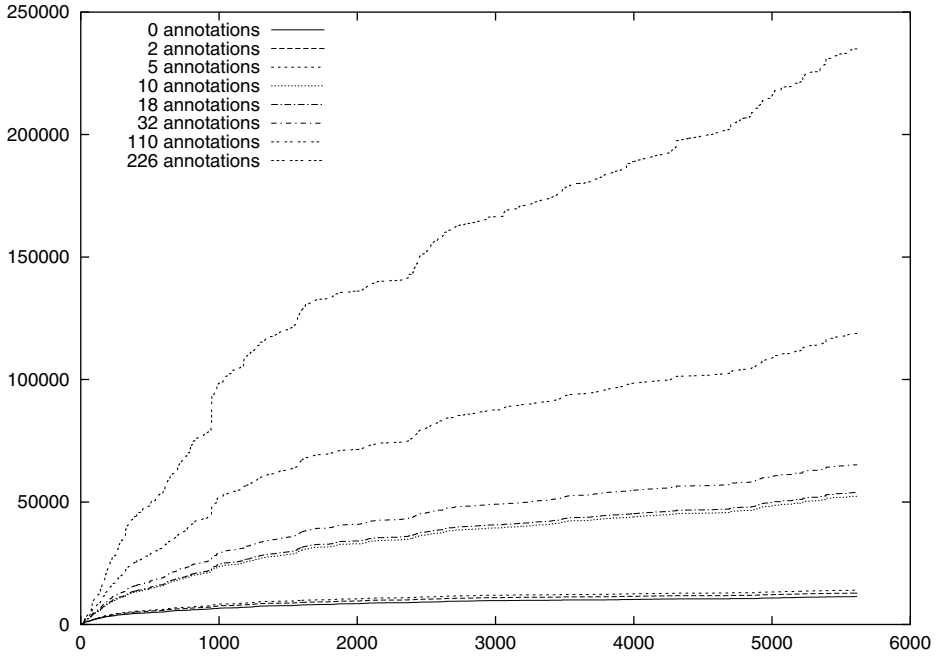


Fig. 12. Plot of the number of all rules obtained under rule equivalence against the number of samples for *csli*. Note that we depict the sum of useful and useless rules without start productions here. Again, the table clearly shows that, at least for 226 symbol annotations, the training corpus is too small.

As already indicated by Figure 12, the training corpus is clearly too small for the approximated grammar with 226 annotations (and probably also too small for 110 annotations). Nevertheless, we conducted a test with both grammars to see how far the two-step approach is away from the HPSG baseline. The HPSG alone resulted in 382 readings and 384 errors for the 572 test sentences (= 188 well-formed sentences w.r.t. the HPSG). Two-step parsing with the approximated grammars resulted in 459 and 494 errors, resp. That is, 75/110 (= 459/494 - 384) of 188 (= 572 - 384) grammatical sentences were classified as ungrammatical according to the missing rules hidden in the training corpus (see figure 13). The results are not as bleak as it seems, since *csli* is a highly diverse test suite, where the deletion of a single training sentence (and its specific syntactic structure) usually results in an approximated grammar that does not recognize this sentence. More redundant, homogeneous real-world data (say, newsgroups) will probably show a better adaption of the approximated grammars to a test sample.

The situation, however, can be improved by omitting the main category symbol in each nonterminal (of each rule). As a result, original incompatible rules suddenly collapse into a new rule (see section 5.3). This strategy in part compensates the unseen data, yielding slightly better results: 444/480 errors have now been detected (see Figure 13).

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

#annot.	#useful	#useless	#ppedges	#readings	#errors	time[s]
32+CAT	31189	28880	38016	2522	312	1.04
110+CAT	35490	75677	31085	332	459	0.98
226+CAT	41937	178846	25305	196	494	1.02
110–CAT	35279	75888	44299	385	444	1.38
226–CAT	41808	178669	35212	235	480	1.49
HPSG	–	–	66402	382	384	268.25

Fig. 13. Useful and useless rules for the 90% training corpus from *csli*, obtained under rule equivalence (without lexicon "lookup" rules). Runtime measurements (#ppedges, #readings, #errors, runtime[s]) are performed against the 10% test corpus (572 sentences) for full two-stage parsing (sum of both stages). Row 3 and 4 presents the results for the two grammars after deleting the main category symbols in CF rules (-CAT). Speedup is more than 100 here (compare this with the results from last subsection), since failed CF parses are, of course, *not* replayed by the second HPSG stage.

Overall, the relatively small amount of training material is responsible for recognizing only half of the grammatical sentences (recall the curves for the two most restrictive grammars in figure 11). In case we do not have enough training material, a more "liberal grammar" might be a good compromise here for two-stage parsing, since it produces less errors (see the first line in Figure 13 for the CFG with 32 annotations). However, such a grammar clearly shifts a larger workload to the HPSG side.

7 Conclusion and outlook

In this article, we have described a corpus-driven method for extracting domain-specific context-free grammars. We have presented a variation of this method that will usually yield smaller grammars while having the drawback of being more general. The measurements have shown that we can not take much advantage of the second method, but one can imagine "weird" UBGs where this method produces CFGs that are exponentially smaller than grammars produced by the first method (see also Kiefer and Krieger 2002). We have also indicated that the approximated CFGs can be turned into PCFGs for disambiguating UBGs in a two-step parsing approach, similar to the treatment in Kiefer *et al.* (2002). Furthermore, the approximated CFGs are of interest to domain-specific NLP applications which are eagerly waiting for cheap and easily-to-produce recognition grammars, e.g., information extraction or language modeling. Finally, we suggested that, starting from the results reported in Rayner *et al.* (2001b), the approximated CFGs are a good starting point for obtaining better language models.

The approach neither generates a true superset nor a true subset of the language accepted by the UBG, but clearly better fits the UBG when given a larger

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

training sample and more annotation values. As we have already indicated, this seeming misbehavior is a desired property, when looking for robust domain-specific grammars. The measurements presented here are very encouraging, but needs to be scrutinized in a domain-specific NLP system.

In case we are not interested in a domain-specific context-free subgrammar of the UBG, even the World Wide Web can be seen as a huge training corpus, due to the following argument. Given a large-scale UBG and a UBG parser (e.g., PET), it is clear that even ungrammatical utterances or utterances not covered by the UBG are worth to be partially parsed, since the UBG parser always comes up with a chart from which we can compute further CF rules which approximate, at least, legal phrases, although the UBG parser has failed overall. This idea clearly rises or falls with the quality of the UBG. Unknown words not covered by the UBG (mostly named entities) are also an important topic that needs to be addressed. A viable solution here is to have some kind of named entity grammars, processed in a weaker formalism. Such a shallow formalism will then be invoked in a preprocessing phase when unknown words are detected, producing lexicon entries for the UBG, which ultimately will show up in the CFG later.[9]

As indicated in section 5.4, it is worth to extract not only recognition rules, but also to have semantic rules as in attribute grammars, so that our CFGs will come along with some meaningful output. This idea would obviate the need for a second unification-based grammar that replays the CF derivations for the mere purpose of semantic construction. However, a third line can be taken here by employing ideas from Diagne *et al.* (1995), Kasper and Krieger (1996) and Kasper *et al.* (1996). Instead of using the full feature structures of a UBG G during replay, we only employ the related grammar G' derived from G: rules and lexicon entries in G' are exactly the feature structures from G that have been evaporated under an appropriate restrictor R (Shieber 1985): $G' = G \setminus R$. Since we are interested in successful UBG derivations and since we like to keep the CF language generalizations in the second replay stage, we delete those constraints on the UBG side which potentially lead to a unification failure (mostly the syntactic constraints). In HPSG-I (Pollard and Sag 1987), specifying the restrictor is easy – mostly the information under `SYN` must be deleted. And in case we are interested in a more shallow semantics, some semantic information from `SEM` has to be deleted too.

Within the next months, we hope to expand this implemented framework in three directions.

Firstly, by using the latest version of the HPSG and more training material, we will try to extend the coverage of the approximated grammars. We will also experiment with more annotations attached to the CF symbols in order to make the CFGs even more specific. The experiments with the *csli* test suite have shown that even a duplication of annotations (from 110 to 226) does not increase the number of useful rules that much (40,742 vs. 58,394 rules).

[9] The shallow processor *SProUT* (Becker et al. 2002; Krieger et al. 2004) is obviously a good candidate to implement this task, since it is a unification-based formalism and has the ability to read in PET grammars.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

Secondly, we will cary out experiments to measure the disambiguation quality of the approximated grammars in a two-step parsing framework, similar to the treatment in Kiefer *et al.* (2002). Since our CFGs are more specific than those produced by the purely grammar-driven approximation in Kiefer *et al.* (2002), we hope to obtain better disambiguation results.

Thirdly, by compiling the resulting CFGs into language models (using, for instance, Nuance and its GSL formalism), it would be interesting to see how much the CFGs can contribute to the quality of the search in the speech recognizer. We hope to answer this question in a students' software project in a few months.

Acknowledgements

First of all, my thanks go to Bernd Kiefer for providing me with the modified parsers and for helping me adapting the grammars. His help during the measurements is gratefully acknowledged. I have also benefitted from many discussions with him. I am also grateful to the three JNLE reviewers for their detailed and well-meaning comments. Thank you. Finally, I would like to thank John Carroll for encouraging me to submit the manuscript to Natural Language Engineering. This work was supported in a first phase by the Verb*mobil* project (grant no. 01 IV 701 V0), resulting in a Lisp implementation of the approach in summer 1998, realized on top of the Verb*mobil* deep core engine (Kiefer et al. 2000). This first implementation of the extraction method was used by Nederhof (2000) to produce CFGs of different size and specificity during his experiments with regular approximation of context-free languages. An extended first Java reimplementation was finished in autumn 2002 and was further extended in spring 2004. Parts of the implementation, the writing of this article, and the measurements were funded by the German BMBF under grant no. 01 IW C02 (QUETAL).

References

Aho, A. V., Sethi, R. and Ullman, J. D. (1986) *Compilers: Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley.

Alshawi, H. (ed.) (1992) *The Core Language Engine*. ACL-MIT Press Series in Natural Language Processing. MIT Press.

Becker, M., Drożdżyński, W., Krieger, H.-U., Piskorski, J., Schäfer, U. and Xu, F. (2002) SProUT—Shallow Processing with Unification and Typed Feature Structures. *Proceedings of the International Conference on Natural Language Processing, ICON-2002*.

Bos, J. (2002) Compilation of Unification Grammars with Compositional Semantics to Speech Recognition Packages. *Proceedings of the 19th International Conference on Computational Linguistics, COLING 2002*, pp. 106–112.

Briscoe, T. and Carroll, J. (1993) Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. *Computational Linguistics* **19**(1): 25–59.

Callmeier, U. (2000) PET—A Platform for Experimentation with Efficient HPSG Processing. *Natural Language Engineering* **6**(1): 99–107.

Cancedda, N. and Samuelsson, C. (2000) Experiments with Corpus-based LFG Specialization. *Proceedings of the 6th Conference on Applied Natural Language Processing*, pp. 204–209.

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

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

Carroll, J., Briscoe, T. and Grover, C. (1991) A Development Environment for Large Natural Language Grammars. Technical Report 233, Computer Laboratory, Cambridge University, UK.

Carroll, J. A. (1993) *Practical Unification-based Parsing of Natural Language*. PhD thesis, University of Cambridge, Computer Laboratory, Cambridge.

Charniak, E. 1993. *Statistical Language Learning*. Cambridge, MA: MIT Press.

Copestake, A., Lascarides, A. and Flickinger, D. (2001) An Algebra for Semantic Construction in Constraint-Based Grammars. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL-2001*, pp. 132–139.

Diagne, A. K., Kasper, W. and Krieger, H.-U. (1995) Distributed Parsing With HPSG Grammars. In *Proceedings of the 4th International Workshop on Parsing Technologies, IWPT'95*, pp. 79–86. (Also available as DFKI Research Report RR-95-19.)

Dowding, J., Hockey, B. A., Gawron, J. M. and Culy, C. (2001) Practical Issues in Compiling Typed Unification Grammars for Speech Recognition. *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics, ACL-2001*, pp. 164–171.

Flanagan, D. (2002) *Java in a Nutshell*. Beijing: O'Reilly.

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

Goldstein, S. D. (1988) Using an Active Chart Parser to Convert Any Context Free Grammar to Backus-Naur Form. Master's thesis, Massachusetts Institute of Technology.

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

Hunt, A. and McGlashan, S. (2004) Speech Recognition GrammarSpecification Version 1.0. Technical report, W3C Recommendation 16 March 2004, http://www.w3.org/TR/2004/REC-speech-grammar-20040316/.

Kaplan, R. and Bresnan, J. (1982) Lexical-Functional Grammar: A Formal System for Grammatical Representation. In: J. Bresnan, editor, *The Mental Representation of Grammatical Relations*, pp. 173–281. Cambridge, Mass: MIT Press.

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

Kasper, W., Krieger, H.-U., Spilker, J. and Weber, H. (1996) 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*, pp. 77–88. Berlin: Mouton de Gruyter.

Kiefer, B. and Krieger, H.-U. (2000) A Context-Free Approximation of Head-Driven Phrase Structure Grammar. *Proceedings of the 6th International Workshop on Parsing Technologies, IWPT2000*, pp. 135–146.

Kiefer, B. and Krieger, H.-U. (2002) 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*, pp. 49–76. CSLI Publications.

Kiefer, B. and Krieger, H.-U. (2004) A Context-Free Superset Approximation of Unification-Based Grammars. In: H. Bunt, J. Carroll and G. Satta, editors, *New Developments in Parsing Technology*, pp. 229–250. Kluwer Academic.

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

Kiefer, B., Krieger, H.-U. and Nederhof, M.-J. (2000) Efficient and Robust Parsing of Word Hypotheses Graphs. In: W. Wahlster, editor, *Verbmobil: Foundations of Speech-to-Speech Translation*, pp. 280–295. Berlin: Springer.

Kiefer, B., Krieger, H.-U. and Prescher, D. (2002) A Novel Disambiguation Method For Unification-Based Grammars Using Probabilistic Context-Free Approximations.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

Proceedings of the 19th International Conference on Computational Linguistics, COLING2002.

Krieger, H.-U. (2004) A Corpus-Driven Context-Free Approximation of Head-Driven Phrase Structure Grammar. In: G. Paliouras and Y. Sakakibara, editors, *Proceedings of the 7th International Colloquium on Grammatical Inference, ICGI-2004*, pp. 199–210. No. 3264, Lecture Notes in Artificial Intelligence. Springer.

Krieger, H.-U., Drożdżyński, W., Piskorski, J., Schäfer, U. and Xu, F. (2004) A Bag of Useful Techniques for Unification-Based Finite-State Transducers. *Proceedings of KONVENS 2004*, pp. 105–112.

Krieger, H.-U. and Schäfer, U. (1994) $\mathcal{TDL}$ – A Type Description Language for Constraint-Based Grammars. *Proceedings of the 15th International Conference on Computational Linguistics, COLING-94*, pp. 893–899. (An enlarged version of this paper is available as DFKI Research Report RR-94-37.)

Lari, K. and Young, S. J. (1990) The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language* **4**: 35–56.

Malouf, R., Carroll, J. and Copestake, A. (2000) Efficient feature structure operations without compilation. *Natural Language Engineering* **6**(1): 29–46.

Moore, R. C. (1999) Using Natural-Language Knowledge Sources in Speech Recognition. In: K. Ponting, editor, *Computational Models of Speech Pattern Processing*, Springer.

Nakazawa, T. (1995) Construction of LR Parsing Tables for Grammars Using Feature-Based Syntactic Categories. In: J. Cole, G. Green, and J. Morgan, editors, *Linguistics and Computation*, pp. 199–219. CSLI Lecture Notes.

Nederhof, M.-J. (2000) Practical Experiments with Regular Approximation of Context-Free Languages. *Computational Linguistics* **26**(1): 17–44.

Neumann, G. (2003) Data-driven Approaches to Head-Driven Phrase Structure Grammar. In: R. Bod, R. Scha and K. Simaán, editors, *Data-Oriented Parsing*, pp. University of Chicago Press.

Neumann, G. and Flickinger, D. (1999) Learning Stochastic Lexicalized Tree Grammars from HPSG. Technical report, German Research Center for Artifical Intelligence (DFKI), Saarbrücken.

Nuance (2004) Nuance Home `http://www.nuance.com`.

Oepen, S. and Callmeier, U. (2000) Measure For Measure: Parser Cross-Fertilization. *Proceedings of the 6th International Workshop on Parsing Technologies, IWPT 2000*, pp. 183–194.

Oepen, S. and Flickinger, D. P. (1998) Towards Systematic Grammar Profiling. Test Suite Technology Ten Years After. *Journal of Computer Speech and Language* **12**(4): 411–436.

Pereira, F. C. and Schabes, Y. (1992) Inside-Outside Reestimation from Partially Bracketed Corpora. *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics, ACL-92*, pp. 128–135.

Pereira, F. C. and Wright, R. N. (1991) Finite-State Approximation of Phrase Structure Grammars. *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics, ACL-91*, pp. 246–255. (An enlarged version is available in E. Roche and Y. Schabes, editors, *Finite-State Devices for Natural Language Processing*. Cambridge, MA: MIT Press.

Pollard, C. and Sag, I. A. (1987) *Information-Based Syntax and Semantics. Vol. I: Fundamentals.* CSLI Lecture Notes, Number 13. Stanford: Center for the Study of Language and Information.

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

Rayner, M., Dowding, J. and Hockey, B. A. (2001a) A Baseline Method for Compiling Typed Unification Grammars into Context Free Language Models. *Proceedings of EUROSPEECH*.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press

Rayner, M., Gorrell, G., Hockey, B. A., Dowding, J. and Boye, J. (2001b) Do CFG-Based Language Models Need Agreement Constraints. *Proceedings of the 2nd Conference of the North American Chapter of the ACL, NAACL2001*.

Rayner, M., Hockey, B. A., James, F., Bratt, E. O., Goldwater, S. and Gawron, J. M. (2000) Compiling Language Models from a Linguistically Motivated Unification Grammar. *Proceedings of the 18th International Conference on Computational Linguistics, COLING 2000*, pp. 670–676.

Shieber, S., Uszkoreit, H., Pereira, F., Robinson, J. and Tyson, M. (1983) The Formalism and Implementation of PATR-II. In: B. J. Grosz and M. E. Stickel, editors, *Research on Interactive Acquisition and Use of Knowledge*, pp. 39–79. Menlo Park, CA: AI Center, SRI International, November.

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

Uszkoreit, H. (1986) Categorial Unification Grammars. *Proceedings of the 11th International Conference on Computational Linguistics*, pp. 187–194.

Van Tichelen, L. (2003) Semantic Interpretation for Speech Recognition. Technical report, W3C Working Draft 1 April 2003, http://www.w3.org/TR/2003/WD-semantic-interpretation-20030401/.

Zeevat, H., Klein, E. and Calder, J. (1987) 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*, pp. 195–222. Centre for Cognitive Science, Edinburgh University, UK.

https://doi.org/10.1017/S1351324906004128 Published online by Cambridge University Press