Introduction to Data-Oriented Parsing

Rens Bod, Remko Scha and Khalil Sima'an (eds.)

2003

CENTER FOR THE STUDY
OF LANGUAGE
AND INFORMATION

Contents

1

A Data-driven Approach to Head-driven Phrase-Structure Grammar

Günter Neumann, DFKI, Saarbrücken

We present HPSG–DOP, a method for automatically extracting a Stochastic Lexicalized Tree Grammar (SLTG) from a HPSG source grammar and a given corpus.[1] Processing of a SLTG is performed by a specialized fast parser. The approach has been tested on a large English grammar and has been shown to achieve additional performance increase compared to parsing with a highly tuned HPSG parser. Our approach is simple and transparent. The extracted grammars are declaratively represented and have a high degree of practical applicability.

1.1 Introduction

Head Driven Phrase Structure Grammar (HPSG) has proven to be a quite successful formalism for specifying natural language grammars in a highly modular and compact manner (Pollard and Sag 1994) supporting the definition of complex linguistic information and interactions between information on different strata systematically using typed feature constraints. On the other hand, inefficiency in processing such grammars is the major obstacle for using the HPSG formalism in practical NL applications (Makino et al. 1998).

[1]This work was supported by a research grant from the German Federal Ministry of Education, Science, Research and Technology (BMBF) to the DFKI project WHITEBOARD, FKZ: 01 IW 002.

Introduction to Data-Oriented Parsing.
Rens Bod, Remko Scha and Khalil Sima'an (eds.).
Copyright © 2004, CSLI Publications.

In recent years several approaches have been proposed to improve the performance of an HPSG system. For example, Van Noord describes a selective memoization technique used in combination with head-corner parsing (van Noord 1997). Where Van Noord focuses on improved parsing strategies, (Makino et al. 1998) focus on the compilation of typed feature structures into an abstract machine for attribute-value logics. Another approach is described in (Kasper et al. 1995) which focuses on the compilation of HPSG into Tree Adjoining Grammar (TAG). Recently, (Kiefer et al. 1999) have presented a bag of useful techniques which are particularly useful to increase the efficiency of unification–based parsing. In (Torisawa et al. 2000) a method to approximate an HPSG–based grammar with a CFG is presented, which is used during HPSG parsing to filter out impossible parse trees.

Common to all of these approaches is that the coverage of the original general grammars is not affected. Thus if the original grammar is defined domain-independently it might also define a great many theoretically valid analyses covering a wide range of plausible linguistic constructions, including the rarest cases. However, in building real-world applications it has been shown to be a fruitful compromise to focus on frequency and plausibility of linguistic structures wrt. a certain domain. A number of attempts have been made to automatically adapt a general grammar to a corpus in order to achieve efficiency through domain-adaptation, e.g., using PATR-style grammars (Briscoe and Carroll 1993, Samuelsson 1994, Rayner and Carter 1996) or lexicalized TAGs (Srinivas 1997).

In this paper we apply the idea of data–oriented approaches for achieving domain-adaptation as an alternative approach to improve efficient processing of HPSG grammars. Beyond efficiency, major goals of our HPSG–DOP approach are 1) to take full advantage of the HPSG formalism when constructing the specialized grammar, and 2) to favor simple and transparent strategies and interfaces, such that the method can be used without any details of the learning strategy, and such that the extracted grammars are declarative.

1.1.1 Overview of HPSG–DOP

The basic idea of HPSG–DOP is to parse all sentences of a representative training corpus using an HPSG grammar and parser in order to automatically acquire from the parsing results a *stochastic lexicalized tree grammar* (SLTG; see sec. 1.3.4 for formal properties of an SLTG and its relationship to Stochastic Tree–Substitution Grammars STSG) such that each resulting parse tree is decomposed into a set of subtrees. The decomposition operation is guided by the head feature principle of HPSG, which requires that the head features of a phrasal sign be shared with its head daughter. This implies that in general the daughter nodes of a phrase can be divided into a head daughter and into non-head daughters (e.g., in a simple verb phrase like $VP \rightarrow V\ NP\ PP$, the verb V is the head element of the verb phrase VP, and NP and PP are the non-head daughters of the VP). Now, the decomposition operates in such a way, that each non-terminal non-head subtree (i.e., a subtree of a tree whose root node is labeled as a non-head daughter, e.g., nominal phrases NP) is cut off, and the cutting points are marked for substitution. The same process is then recursively applied to each extracted subtree. Each extracted tree is automatically lexically anchored and each node label of the extracted tree compactly represents a set of relevant features by means of a simple symbol (these are specific category labels defined as part of the HPSG source grammar, cf. 1.3.2). For each extracted tree a frequency counter is used to estimate the probability of a tree, after all parse trees have been processed.

Processing of an SLTG is performed by a two level parser. In a first step a new sentence is completely parsed using the extracted SLTG, followed by a second step where the SLTG-valid parse trees are expanded *off-line* by applying the corresponding feature constraints of the original HPSG-grammar giving our approach an LFG-style processing flavor, (Kaplan and Bresnan 1982). The performance of both steps directly depends on the number of SLTG-valid parse trees found for a new sentence. Our approach has many advantages:

- An SLTG has context-free power and meets the criteria of a lexicalized CFG.
- The off-line step guarantees that no information of the original grammar is lost (including semantics).
- Since the SLTG learning phase is driven by the HPSG principles of the source grammar, the whole extraction process is simple and transparent.
- Our approach allows a proper embedding of statistical information into HPSG structures, which supports preference-based and robust processing.
- The whole approach has an important application impact, e.g., for controlled language processing or information extraction, making use of rich and linguistically motivated information from HPSG. Furthermore, the same mechanism can also be applied to NL generation (cf. (Neumann 1997)), as well as to efficient interleaved parsing and generation (cf. (Neumann 1998b)).

The approach has been implemented on top of the LKB (Linguistic Knowledge Building) system, a unification–based grammar development, parsing and generation platform currently being developed at CSLI (Copestake 1999), and has been tested using the large HPSG–based English Resource Grammar being developed as part of the LinGO (Linguistic Grammars Online) project also at CSLI (cf. section 1.2).

The rest of the paper is organized as follows. Firstly, we provide some background information of the used HPSG source system in the next section. In section 1.3 we describe details of the learning phase. In section 1.4, the parsing strategy is described, as well as some programming issues. In section 1.5 we report on several experiments which have been performed in order to measure the performance of the SLTG parsing results. We relate our work to others in section 1.6 and give a conclusion and outline of future work in section 1.7.

1.2 The HPSG source system

1.2.1 The LinGO grammar

The grammar used in our study is the English Resource Grammar being developed as part of the LinGO (Linguistic Grammars Online) project at CSLI, Stanford University, (Copestake et al. 2000). The LinGO grammar consists of a rich hierachy of 1,663 abstract lexical types, a smaller hierarchy of 180 rules types, 58 specific phrasal and lexical rules, and 6,766 lexical entries.

The grammar uses 79 features defined on the lexical and phrasal types of the 4,562 lexical semantic predicates. Syntactic coverage of the grammar is relatively broad, with a central focus on providing precise semantic interpretations for each phenomenon that is assigned an analysis, using the Minimal Recursion Semantics framework of (Copestake et al. 1995), (Copestake et al. 1997). As one measure of breadth, the LinGO grammar provides correct syntactic and semantic analysis for 83% of 8,520 well–formed utterances found in the transcriptions of 175 human–humans dialogues recorded as part of the Verb*mobil* spoken language translation project, cf. (Wahlster 2000). Sentence length in this sample average 8.5 words, with the conversations on the topics of scheduling meetings and making hotel reservations (see (Flickinger et al. 2000, Flickinger 2000) for a detailed discussion of the grammar's architecture and coverage).

1.2.2 The LKB system

Parsing and hence the training phase (see below) with the LinGO grammar is performed using the LKB (Linguistic Knowledge Building) system, (Copestake 1999). The LKB passive chart parser uses a breadth–first CKY–like algorithm which is highly tuned for efficient processing of HPSG grammars. For example, the use of a unification quick check reduces parsing time by approximately 75%. The quick check (cf.

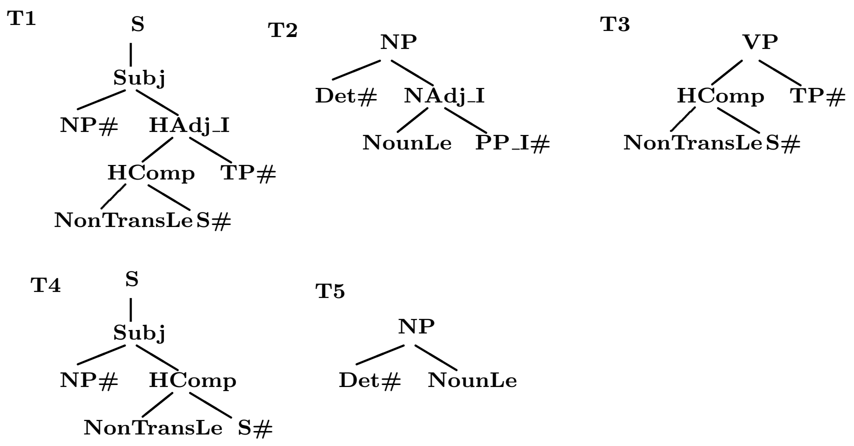

FIGURE 1 Some trees extracted from *I guess we need to figure out a day, within the next two months.* The symbols S, NP, VP, TP, Det, PP_I have been determined by means of specialization (see text). # denotes the substitution marker.

(Kiefer et al. 1999)) is invoked before each unification attempt to check the most frequent failure points, each stored as a feature path. Frequency information is obtained by processing a set of sentences in an off–line mode, and the n paths with the highest failure counts are saved.

1.3 Learning of SLTG

Learning of an SLTG starts by first parsing each sentence s_i of the training corpus with the source HPSG system. The resulting feature structure fs_{s_i} of each example also contains the parse tree pt_i, where each non-terminal node contains the label of the *HPSG-rule schema* (e.g, head-complement rule) that has been applied during the corresponding derivation step as well as a pointer to the feature structure of the corresponding sign. The label of each terminal node consists of the *lexical type* of the corresponding feature structure. For example, the lexical type of the verb `to guess` is NonTransLE decoding general properties of non-transitive verbs, and the lexical type of a noun like `month` is NounLE, the lexical type for nouns.[2]. Each parse tree pt_i is now processed by the following interleaved steps (see also figure 1).

[2]Note that this means, that we abstract from specific information of the word form like morphosyntactic information. This presupposes of course that for the processing of new sentences we do firstly have to determine the lexical types of each word before we can apply the SLTG–trees, see also sec. 1.4

1.3.1 Decomposition

Each parse tree is decomposed into a set of subtrees such that each non-head subtree is cut off, and the cutting point is marked for substitution. Testing whether a phrase is a head phrase or not can be done very easily by checking whether the top-level type of a rule's label feature structure is a subtype of a general *headed phrase* which is defined in the LinGO grammar. The same holds for adjunct phrases (see below).

The decomposition process is recursively applied on each extracted subtree. Each extracted tree is automatically lexically anchored, where the anchor consists of a lexical type and the path from the lexical anchor to the root defines a head-chain. For each extracted tree a frequency counter is used to estimate its probability after all parse trees of the training set have been processed.

In order to automatically enrich the coverage of the extracted grammar, two additional operations will be performed during decomposition. Note that these two additional operations applied on the training material can be considered as linguistically justified tree induction operations.

Firstly, each subtree of the head-chain is copied and the copied tree is processed individually by the decomposition operation (e.g., in figure 1, the tree $T3$ is copied from the head chain of $T1$.). This means that a phrase which occurs only in a head–position in the training corpus can now also be used in nonhead-positions by the SLTG-parser when parsing new sentences. We say that a phrase X occurs in head–position of a phrase Y if the root node of X is the head daughter of Y. If the root node of a phrase is a nonhead–daughter of another phrase (e.g., a modifier), then we say that the phrase is in nonhead–position.

Secondly, if the SLTG-tree has a modifier phrase attached, then a new tree is created with the modifier *unattached* (applied recursively, if the tree has more than one modifier). Unattachment of a modifier m is done by raising the head daughter of m into the position of m (e.g, in figure 2, the tree $T4$ is obtained from the tree $T1$ by replacing the subtree rooted at $HAdj\_I$ of tree $T1$ with the subtree rooted at $HComp$).[3] Note, that unattaching a modifier actually means that a *copy* of a tree is made (a copy of $T1$ is made which is then named $T4$, see figure 2) and that this copied tree is *destructively* changed by unattaching the modifier as described above (which finally yields tree structure of $T4$ below in fig. 2). In general this means that for a tree t_n with n modifiers we obtain a sequence of n trees $t_{n-1} \ldots t_{n-n}$ by iterating the process of unattachment

[3] The current grammar is binary branching, so it suffices to raise the head daughter. If a flat representation of modifiers is used one would remove the modifier daughters first, before the head daughter would be raised.

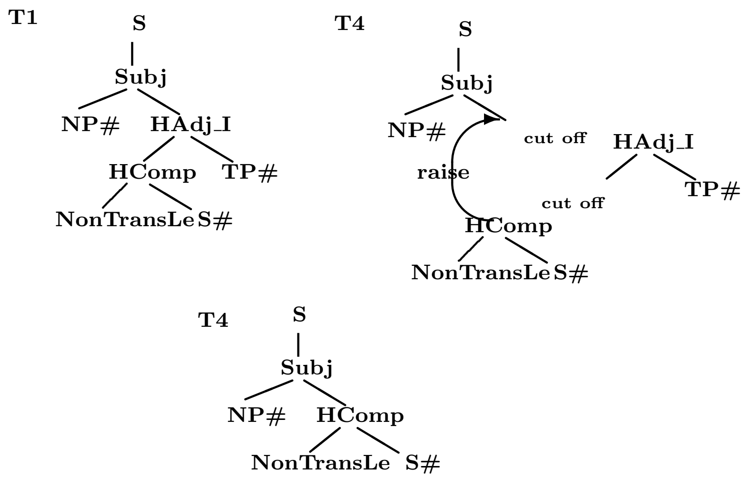

FIGURE 2 Unattachment of a modifier is obtained by first copy the tree t_1 which yields the tree t_4 in the right corner of the figure. From this tree the subtree in head–position of the tree rooted with the modifier is raised into the position of the modifier node, which yields the final tree t_4.

on each copied tree t_i unless no more modifier can be identified.

The advantage of unattaching modifiers is that we will be able to also recognize new sentences with fewer or no modifiers using our extracted grammar by using only the substitution operation. The possible maximum number n of modifier sequences is constrained by the training corpus. Note that this actually means that we do not explicitly factor out the recursion of adjuncts, as is done in recent approaches of extracting lexicalized tree adjoining grammars from tree–banks (e.g., (Xia 1999, Chen and Vijay-Shanker 2000, Chiang 2000, Frank 2001)), which would have the advantage of a higher degree of generalization. However, the disadvantage is that we would need additional adjunction operations during parsing of an SLTG.[4] In our approach, the substitution operation suffices because all possible modifier attachments are already spelled out.

[4]Implementation of the adjunction operation would also require more a fined–grained representation of the extracted trees, at least if we follow common practice in parsing of TAGS, cf. (Schabes and Waters 1995). Our parser directly operates on the SLTG-trees, where (Schabes and Waters 1995) first "explode" a tree by mapping it to a set of local trees of depth one. These local trees define the items in the chart parser. Compared to that, our system treats whole trees as items, and thus has to process significantly fewer items.

1.3.2 Specialization

The root node as well as all substitution nodes of an extracted tree are further processed by replacing the rule label with a corresponding *category label*. The possible set of category labels is defined in the type hierarchy of the HPSG source grammar. They express equivalence classes for different phrasal signs. For example, phrasal signs whose value of the LOCAL.CAT.HEAD feature is of type *noun*, and whose value of the LOCAL.CAT.VAL.SUBJ feature is the empty list, are classified as NPs. Now, if the associated feature structure of a rule label $HeadAdjunct$ of the current training sentence is subsumed by the NP type, then $HeadAdjunct$ is replaced by NP. Note that this step actually performs a specialization of the current tree, because the same tree might occur in another tree in verbal position. In that case, $HeadAdjunct$ might be replaced by the type VP.

The definition of category labels is declarative. Thus it is possible to define more fine-grained labels directly as part of the source grammar leading to more specific SLTG trees. This can be done by the grammar writer without knowing any details of the learning strategy.

1.3.3 Probability

For each extracted tree a frequency counter is maintained. After all parse trees of the training set have been decomposed and specialized, we estimate a tree's probability as follows: First we count the total number n of all trees which have the same root label α. Then we divide the frequency number m of a tree t with root α by n. This gives the probability $p(t)$. In doing so, the sum of all probabilities of trees t_i with root α is 1 (i.e., $\sum_{t_i:root(t_i)=\alpha} p(t_i) = 1$). Since we have only the substitution operation, the probability of a derivation will be the product of the probabilities of the trees which are involved in the derivation, and the probability of a parse tree is the sum of the probabilities of all derivations (see also next section).

1.3.4 Properties of an SLTG

Compared to the original HPSG source grammar, the SLTG has the following properties:

- An SLTG has context-free power, since it allows center embedding (note that it has been extracted from a finite set of parse trees). Recursion of modifiers is actually "spelled out" where the training sentence with the largest number of modifiers defines the upper bound.
- A SLTG cannot derive the empty element, because every SLTG tree has a lexical anchor and they are finitely ambiguous, because

every tree introduces a lexical item into the resulting string. Thus an SLTG meats the criteria of lexicalized CFG.

- SLTG trees have a larger domain of locality than the HPSG rule schemata or CF rules. Thus parsing can benefit from a high degree of top-down information. Furthermore, since every tree is lexically anchored, parsing will produce less spurious ambiguities than would be the case for P-CFG.
- Since decomposition is linguistically guided, the resulting SLTG-trees are linguistically oriented. On the other hand, the statistical information expresses a relationship between lexical types and their probable syntactic context.

Formally, an extracted SLTG for HPSG–DOP constitutes a particular application of the Stochastic Tree–Substitution Grammar (STSG), described in Chapter 1 in this volume. The major difference is that in SLTG each elementary tree must has at least one terminal element. Thus, one might adapt the definition of STSG as follows.
A *Stochastic Lexicalized Tree Grammar* G is a 5-tuple $\langle V_N, V_T, S, R, P\rangle$ where:

1. V_N is a finite set of nonterminal symbols.
2. V_T a finite set of terminal symbols.
3. $S \in V_N$ is the distinguished symbol.
4. R is a finite set of elementary trees whose top nodes and interior nodes are labeled by nonterminal symbols, and whose yield nodes are labeled by terminal or nonterminal symbols. At least one yield node must be a terminal symbol, i.e., $\forall t \in R : \exists x : x \in \mathit{yield}(t) \wedge x \in V_T$.
5. P is a function which assigns to every tree $t \in R$ a probability $P(t)$. For a tree t with root node $\mathit{root}(t) = \alpha$, $P(t)$ is interpreted as the probability of substituting t on a node α. Therefore, we require that $0 \leq P(t) \leq 1$ and $\sum_{t:\mathit{root}(t)=\alpha} p(t) = 1$.

The definitions for derivation, probability of a derivation and of probability of a parse tree are as for STSG.

1.3.5 Representation of an SLTG

Once an SLTG has been learned all trees are stored in an array, and a lexicon is computed using all lexical anchors (types) from the SLTG. Each lexical element keeps a list of indexes for the corresponding trees in the array. In addition a table is compiled from the HPSG source grammar, which assigns to each lexical type its corresponding phrase type. This table is used for speeding up parsing of an SLTG. All data structures of the SLTG are stored in a grammar file in readable form, so that the complete grammar can easily be loaded and processed when needed.

1.4 Parsing of SLTG

In HPSG–DOP, parsing of an SLTG is performed by a chart-based agenda-driven bottom-up parser (following (Schabes and Joshi 1991)). The two major steps are:

- tree selection: selection of a set of SLTG-trees associated with the lexical items in the input sentence,
- tree combination: parsing of the sentence with respect to this set.

After parsing, each SLTG-parse tree is "expanded" by unifying the feature constraints of the HPSG source grammar into the parse trees. If successful, it determines a complete valid feature structure wrt. the HPSG grammar (e.g., all agreement and semantic information). If unification fails, the SLTG parse tree is rejected, and the next most likely tree is expanded.

1.4.1 Tree selection

All SLTG-trees are lexically anchored. A lexical anchor is a lexical type. Since different words can have the same lexical type, they are comparable to pre-terminal elements in a CFG. Thus for each word w_i of a new input sentence we determine its lexical type l_i (in general we will have a set of types) and use this information to retrieve the corresponding set of SLTG-trees T_i with anchor l_i.[5] All extracted trees and all pairs $\langle w_i, l_i \rangle$ are used to initialize the SLTG-parser's chart, where $\langle w_i, l_i \rangle$ is interpreted as a *passive item* (i.e., an item which has no substitution node), and a selected SLTG-tree is interpreted as an *active item*, if it has at least one substitution node. The position of the word in the input sentence is used as initial span information for an item.

Note that once tree selection has been performed, a subset of the grammar has been selected, i.e., no more trees can enter the parsing step (observations from first experiments support the assumption made in (Schabes and Joshi 1991) that the tree selection phase is the most

[5] In the HPSG source system, lexical processing consists of morphological processing followed by unification of lexical-specific syntactic and semantic information. In order to speed up processing and to be able to use the whole SLTG system without the HPSG system, we have compiled the lexicon into a full-form lexicon, which directly gives us the set of possible lexical types for a given word-form in the input sentence. Lexical access is a little bit more complicated than simple lookup because we have to deal with multi-word entries, e.g., "a little bit". The current full-form lexicon has more than 25,000 entries.

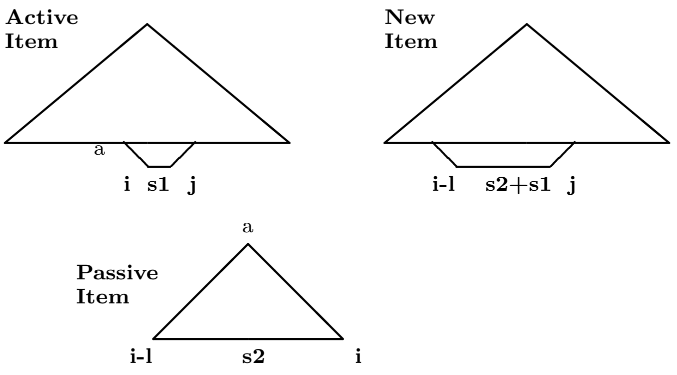

FIGURE 3 Left substitution of an active item with string $s1$ and span $\langle i,j \rangle$ with a passive item with string $s2$ and span $\langle i\text{-}l,i \rangle$ yields a new item with string $s2{+}s1$ and span $\langle i\text{-}l,j \rangle$.

significant factor for a two-phase parsing strategy; see also sec. 1.5). Furthermore, the anchor positions express restrictions on the possible combinations of trees, because an anchor actually splits the set of trees it can combine with into a left and right subset (see also (Schabes and Joshi 1991)).

1.4.2 Tree combination

This is the process of building a complete parse tree for the input sentence by combining active and passive items until one passive item for the whole input has been found. The main operation is tree substitution, which substitutes a substitution node with label α of an active item with span $\langle i,j \rangle$ with a passive item with root node α with span $\langle i\text{-}l,i \rangle$ (left substitution, where $1 \leq l \leq i$) or $\langle j,j{+}k \rangle$ (right substitution, where $1 \leq k \leq n\text{-}j$ & $n =$ length(input)) (see also figure 3).[6] The probability of a new item is obtained by multiplying the probability of the passive and active items. Terminal elements, i.e. passive items created directly from the input words, receive probability one.

Thus described, tree combination extends the string of an active item from its lexical anchor in the left as well as the right direction. However, the whole input string is processed incrementally left-to-right controlled by a chart-based agenda-driven bottom-up control strategy. Both strategies together will automatically complete all left substitution nodes of an active item before the right substitution nodes are considered from left to right (we call this the *left completion condition*). Only items contained in the chart can be combined. If tree combination creates a

[6]We are using the Gorn number for efficiently referring to a substitution node in an active item (see also (Schabes 1990)).

Well I think we need to meet, for another, two hours, to discuss this matter further.
What is a convenient time, for like a two hour slot, within the next couple weeks, for you.
Well, I am free on Monday, except for, ten to twelve in the morning.
and I am free, on Tuesday, in the afternoon.
And, actually, I am also free, Wednesday and Thursday in the afternoon.
And, Thursday all day.
Well actually, next week is not, that, great for me.
Because, Wednesday through Friday, I am like in a,
in seminars all day.
So, how does the week of, the, either the thirty thirty first of May, or some time the, beginning of June, look for you.
Well, I am, out of town, Monday through Wednesday, of that week.
I will be free, that Thursday the third, in the afternoon, or Friday morning.
And again I am also still free, afternoons of Monday and Tuesday next week.
Well then, Thursday afternoon looks good.

FIGURE 4 The first fourteen utterance from the chosen VerbMobil corpus.

new item it is inserted into the agenda from which it will eventually be inserted into the chart. The agenda mechanism sorts new items based on an item's probability p and its span $\langle i,j \rangle$ computing p+(j-i), i.e. "most probable, longest matching".

Some pruning techniques are applied in order to keep the chart as small as possible (see also 1.6). Currently, pruning is used for filtering out impossible combinations of SLTG-trees, namely those which would lead to invalid spans of new items.

1.5 Experiments

We have conducted initial experiments using a VerbMobil corpus of 2000 utterances from dialogs about scheduling of meetings, with an average sentence length of 7.8 words (see figure 4 for some example utterances from that corpus).

We run the first 1000 sentences for training of an SLTG as described above using the LKB parser system. The LKB parser was able to find a valid analysis for 831 sentences which corresponds to a coverage of

```
(21
 (("root_cl" . S)
  ("subjh" ((:SUBST . NP))
   ("hcomp" (MV_CP_PROP_NON_TRANS_LE-FIN-VERB-LEX)
   ((:SUBST . S)))))
 9 0.0059920107)

(29
 (("hspec" . NP) ((:SUBST . DET))
  ("adjh_i" ((:SUBST . ADJ)) (NOUN_PPCOMP_LE-NOUN-LEX)))
 14 0.009247027)

(2163
 (("root_cl" . S)
  ("subjh" ((:SUBST . NP)) ("hcomp" (WOULD_AUX_POS_LE)
           ((:SUBST . VP)))))
 19 0.012649801)
(2164
 (("nadj_rr" . NP) ("hcomp" (HOUR_LE) ((:SUBST . ADJ)))
                   ((:SUBST . PP-I))) 14
 0.07526882)
```

FIGURE 5 Some of the extracted lexically anchored trees in Lisp list representation. The first number is the tree's index, followed by the tree itself, followed by its frequency and its computed probability.

83.1%. In each case the first reading is selected for the training which corresponds to a corpus size of 831 parse trees.

Processing of this HPSG–based tree–bank yields an SLTG containing 3818 different lexicalized trees anchored with a lexical type (see figure 5 for some extracted trees using the external Lisp list representation). There were 180 different lexical types computed which gives an average of about 21 trees per lexical anchor. In some cases – mostly for lexical types of auxiliar and modal verbs – more than hundred trees were obtained, e.g., for the lexical type WOULD_AUX_POS_LE 138 different trees, and for the lexical type BE_C_AM_LE even 355 different trees are extracted. It is clear that this high number of trees will effect the performance of the SLTG–parser (as already observed through our experiments), and is therefore a good source for improving the performance of the parser.

From the original 2000 sentences, the next 1000 sentences were used for testing the coverage of the extracted SLTG. We first run the LKB parser to measure its coverage on this test corpus, which resulted in a coverage of 803 analyzed sentences. We used this subset of parsed sentences to test the coverage of the SLTG. For the computation of the first reading of each sentence, the LKB parser needed 1.40 seconds on average .[7] Actually this shows that the LKB HPSG parser (including rule filters and unification quick checks) is already quite fast, at least for this sort of corpus.

Then, the SLTG parser was called in off–line unification mode such that for each sentence it stopped after the first complete SLTG-tree could successfully be "expanded" by unifying the feature constraints of the HPSG source grammar. Thus seen, we consider a sentence analysis as valid, if it is consistent with respect to the HPSG constraints (including all lexical constraints, of course). Following this way, 704 sentences were recognized which corresponds to a coverage of 87.67%. On average, the SLTG parser needed 0.48 seconds per sentence.

We then run the SLTG-parser without the off-line unification step, but in exhaustive mode (computing an average of 58 alternative parse trees for each sentence). For each sentence, we checked whether the set of possible analysis also contained the first reading of the HPSG parser considering only the parse tree, i.e., no constraints other than those conflated into the category labels (cf. 1.3.2). This was the case for 551 sentences which means that we obtained an exact derivation accuracy (exact match of the resulting parse trees) of 68.61%.

We have not yet made a detailed qualitative evaluation, however it seems clear, that the coverage of an SLTG grammar can be improved, if a larger corpus is used for training. This assumption is supported by the experiments reported in (Rayner and Carter 1996) (see also section 1.6). They found, for example, that the coverage loss (compared to the orginal source grammar) for a training size of 1000 sentences is 10.8%, where it is only 5.0% for a training size of 15,000 sentences. The increase in coverage can be interpreted such that the more example analysis a system has observed, the more grammatical variations will be detected. Of course, this increase in grammatical variation might negatively effect the performance of the SLTG parser, which directly shifts our focus to performance issues.

Concerning efficiency, the current speed up is by a factor of 2.92. This is already a quite promising result. On the one hand side, the parser of the LKB system is a highly tuned one which already uses a kind of

[7]Processing was done on a SUN Ultra Enterprise 450, 4 CPUs (400MHz) and 4GB RAM.

data–driven approach through the quick check mechanism. As already described in section 1.2, the quick check results in an performance improvement of 75%. On the other hand side, our SLTG parser is still not optimized. For example, we assume that the number of trees selected for an input sentence can be reduced substantially using more specific lexical types (which would have to be defined as part of the grammar), making use of pruning strategies based on (Rayner and Carter 1996) in order to filter out unplausible intermediate tree structures or to induce more word-based statistical information, e.g., exploring strategies to statistically "connect" word forms or word stems with certain tree structures, or to make use of anchored–based n–grams.

1.6 Related work

An SLTG is structurally related to lexicalized TAGs. For that reason one might argue that an SLTG could also be obtained by first compiling an HPSG grammar into an LTAG (preserving full coverage, cf. (Kasper et al. 1995)), and then to approximate the LTAG to a lexicalized CFG. Although we are not aware of any work describing such an approach, the advantage of our approach is that it combines both steps into one.

Our approach is also related to approaches of grammar specialization based on Explanation-based Learning (EBL), cf. (Samuelsson 1994, Srinivas and Joshi 1995, Rayner and Carter 1996). Samuelsson (1994) presents a method in which tree decomposition is completely automated using the information-theoretical concept of entropy, after the whole set of parse trees has been indexed in an and-or tree. This implies that a new grammar has to be computed if the tree–bank changes (i.e., reduced incrementality) and that the generality of the induced subtrees depends much more on the size and variation of the tree–banks than ours. On the other hand, this approach seems to be more sensitive to the distribution of sequences of lexical anchors than our approach. Rayner and Carter (1996) describe an approach in which EBL and constituent pruning is presented. They showed that constituent pruning (i.e. the removal of edges of a chart that are relatively unlikely to contribute to correct analysis) in combination with EBL increases system performance considerably, such that processing times of other components (lexical and morphological processing) become crucial. It should be not difficult to integrate these or similar pruning strategies into our parser (see sec. 1.4) which would help us to keep the chart as small as possible during parsing of one sentence.

In (Carroll and Weir 1997) a classification of different methods (named

S(1) to S(4)) for encoding frequency information in lexicalized grammars is given and applied to prominent existing approaches. The one with the most globally-dependent frequencies is S(4) and Bod's DOP framework (Bod 1995) actually belongs to that class. We compute the frequency information in the same way as in DOP once a parse tree has been decomposed. The main difference here is that we only allow one lexical anchor per tree, where Bod also allows for multiple anchors. Although this implies that we might need less training data than DOP we have to pay the price of reduced lexical co-occurrence restrictions during application, which makes our approach weaker than DOP. However, it is not difficult to extend our approach to employ multiple lexical anchors (using the "tree–bank" version described in (Neumann 1998a)). In that case it would be possible to evaluate the trade-off between training size and coverage of the extracted grammar.

1.7 Conclusion and Future Work

We have presented an approach for improving parsing with HPSG by automatically extracting a stochastic lexicalized tree grammar. Grammar extraction is based on linguistically guided tree decomposition taking full advantage of the HPSG principles. An extracted grammar is still "HPSG-based" in the sense that the structures computed by the SLTG-parser are fully compatible with HPSG. Thus our approach could be characterized as a corpus-driven method for "de-compressing" the search space of an HPSG grammar rather than for compiling it into some other format. Nevertheless, the extracted grammars have interesting relationships to lexicalized TAGs which allow us to adapt basic parsing technology from that area. We also showed how to integrate stochastic information into the extracted grammars which supports incorporation of statistically based parsing strategies into our HPSG-based approach. Our next steps involve more detailed evaluation, improved pruning, as well as further experimentation with an increased set of more discriminating category labels, which can help to reduce the number of trees retrieved in parsing. Here, the use of strategies for automatically determination such a more fined-grained label set is envisaged. We also have begun to explore strategies for robust processing, especially handling of unknown words. The main idea here is to classify unknown words according to the context of SLTG-trees which are anchored by known words.

Acknowledgment

Many thanks to Dan Flickinger from yy.com. He provided me with the HPSG grammar I have used for testing the SLTG approach. Furthermore, parts of this article are based on (Neumann and Flickinger 1999), a technical report that Dan has coauthored. I also acknowledge very much Ann Copestake for developing the LKB system and for making it an open source.

References

Bod, R. 1995. *Enriching Linguistics with Statistics: Performance Models of Natural Language.* Doctoral dissertation, University of Amsterdam. ILLC Dissertation Series 1995-14.

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

Carroll, J., and D. Weir. 1997. Encoding Frequency Information in Lexicalized Grammars. In *Proceedings of 5th ACL/SIGPARSE International Workshop on Parsing Technologies.* MIT, Cambridge.

Chen, J., and K. Vijay-Shanker. 2000. Automated Extraction of TAGs from the Penn Treebank. In *6th International Workshop on Parsing Technologies (IWPT'2000).* Trento, Italy.

Chiang, D. 2000. Statistical parsing with an automatically–extracted tree adjoining grammar. In *38th ACL.* Honk Kong.

Copestake, A. 1999. The (new) LKB system. CSLI, Stanford University: http://www-csli.stanford.edu/~acc/doc5-2.pdf.

Copestake, A., D. Flickinger, R. Malouf, S. Riehemann, and I. Sag. 1995. Translation using Minimal Recursion Semantics. In *Proceedings, 6th International Conference on Theoretical and Methodological Issues in Machine Translation.*

Copestake, A., D. Flickinger, and I. Sag. 1997. Minimal Recursive Semantics: An Introduction. Technical report. Stanford University: CSLI.

Copestake, A., D. Flickinger, and I. Sag. 2000. Linguistic Grammars Online (LinGO) project. http://hpsg.stanford.edu/hpsg/lingo.html.

Flickinger, D. 2000. On building a more efficient grammar by exploiting types. *Journal of Natural Language Engineering* 6(1):15–28.

Flickinger, D., A. Copestake, and I. Sag. 2000. HPSG Analysis of English. In *W. Wahlster (ed.) Verbmobil: Foundations of Speech-to-Speech Translation*, 254–263. Springer–Verlag Berlin Heidelberg New York.

Frank, A. 2001. Treebank Conversion. Converting the NEGRA treebank to an LTAG grammar. In *Workshop on Multi-layer Corpus-based Analysis*. Iasi, Romania, July.

Kaplan, R. M., and J. Bresnan. 1982. Lexical-Functional Grammar: A Formal System for Grammatical Representation. In *The Mental Representation of Grammatical Relations*, ed. J. Bresnan. 173–281. Cambridge, MA: MIT Press.

Kasper, R., B. Kiefer, K. Netter, and K. Vijay-Shanker. 1995. Compilation of HPSG into TAG. In *33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, MA.

Kiefer, Bernd, Hans-Ulrich Krieger, John Carroll, and Rob Malouf. 1999. A Bag of Useful Techniques for Efficient and Robust Parsing. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics, ACL-99*, 473–480.

Makino, T., M. Yoshida, K. Torisawa, and J. Tsujii. 1998. LiLFeS – Towards a Practical HPSG Parser. In *Proceedings of the 36th ACL and 17th Coling*, 807 – 811. Montreal.

Neumann, G. 1997. Applying Explanation-based Learning to Control and Speeding-up Natural Language Generation. In *35th Annual Meeting of the Association for Computational Linguistics/8th Conference of the European Chapter of the Association for Computational Linguistics*. Madrid, Spain.

Neumann, G. 1998a. Automatic Extraction of Stochastic Lexicalized Tree Grammars from Treebanks. In *4th workshop on tree-adjoining grammars and related frameworks*. Philadelphia, PA, USA, August.

Neumann, G. 1998b. Interleaving Natural Language Parsing and Generation Through Uniform Processing. *Artifical Intelligence* 99:121–163.

Neumann, G., and D. Flickinger. 1999. Learning Stochastic Lexicalized Tree Grammars from HPSG. Technical report. Saarbrücken: DFKI, Juni.

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

Rayner, M., and D. Carter. 1996. Fast parsing using pruning and grammar specialization. In *34th Annual Meeting of the Association for Computational Linguistics*. Morristown, New Jersey.

Samuelsson, C. 1994. Grammar Specialization through Entropy Thresholds. In *Proceedings of the 32nd Annual Meeting of the Association forComputational Linguistics*, 188–195.

Schabes, Y. 1990. *Mathematical and Computational Aspects of Lexicalized Grammars*. Doctoral dissertation, University of Pennsylvania, Philadelphia, USA.

Schabes, Y., and A. K. Joshi. 1991. Parsing with Lexicalized Tree Adjoining Grammar. In *Current Issues in Parsing Technology*, ed. M. Tomita. 25–48. Boston: Kluwer.

Schabes, Y., and R. Waters. 1995. Tree Insertion Grammar: A Cubic-Time Parsable Formalism that Lexicalizes Context-Free Grammar without Changing the TRees Produced. *Computational Linguistics* 21:479–513.

Srinivas, B. 1997. *Complexity of Lexical Restrictions and Its Relevance to Partial Parsing*. Doctoral dissertation, University of Pennsylvania. IRCS Report 97–10.

Srinivas, B., and A. Joshi. 1995. Some Novel Applications of Explanation-Based Learning to Parsing Lexicalized Tree-Adjoining Grammars. In *33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge, MA.

Torisawa, K., N. Nishida, Y. Miyao, and J. Tsujii. 2000. An HPSG parser with CFG filtering. *Journal of Natural Language Engineering* 6(1):63–80.

van Noord, G. 1997. An Efficient Implementation of the Head-Corner Parser. *Computational Linguistics* 23:425–456.

Wahlster, W. 2000. *Verbmobil: Foundations of Speech-to-Speech Translation*. Artificial Intelligence. Springer–Verlag Berlin Heidelberg New York.

Xia, F. 1999. Extracting tree adjoining grammars from bracketed corpora. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium(NLPRS-99)*. Beijing, China.