Adapting HPSG-to-TAG compilation to wide-coverage grammars

Tilman Becker and Patrice Lopez

DFKI GmbH
Stuhlsatzenhausweg 3
D-66123 Saarbrcken, Germany
{becker,lopez}@dfki.de

Abstract

The HPSG-to-TAG compilation algorithm proposed in (Kasper et al., *1995) has been the basis of large scale experiments in VerbMobil, a speech–to–speech dialouge translation system in the scheduling and travel domain. The results here refer to the English HPSG grammar developed at CSLI. Several non-trivial theoretical problems have been discovered by the practical application of this algorithm. This paper presents these experiments, the main shortcomings of the initial algorithm and some of the solutions we have developed in order to use the resulting compiled LTAG grammar in a real world system.*

1. Introduction

The LTAG formalism is a mathematical tool that has proven to be attractive for the modeling of natural language syntax. In parallel to pure-LTAG grammar developments, some researches have addressed the relation between LTAG and existing formalisms both for theoretical and practical reasons. In particular, compiling a LTAG grammar from a HPSG grammar has been proposed by (Kasper, 1992). Such a compilation is interesting for several reasons:

- **Sharing of resources** between the two formalisms, in particular the syntactic lexicon. For instance, since both formalisms are lexicalized, the syntactic lexicon which gives all possible predicative frames for each lemma is very costly to write.
- **Speed efficiency:** The precompilation process allows to identify substructures of the HPSG grammar that are not context-dependent. The extracted partial backbones can be tabulated (chart parsing, memoization) which results in more time efficient systems than a direct HPSG parser/generator.
- **Capturing dependencies:** An LTAG elementary tree directly encodes a full syntactic context by the way of an extended domain of locality. Elementary trees are combined in order to realize dependency relations between the syntactic contexts they represent. Thus the construction of a sentence can be obtained very easily just with a dependency tree indicating the elementary trees that are involved and their mutual dependencies. This information is represented only indirectly in an HPSG derivation.
- **Exploiting HPSG's** expressivity as well as utilizing existing HPSG grammars is interesting for the LTAG community. HPSG grammars usually include the syntax-semantics interface and a semantic level that is ignored in existing LTAG grammars. HPSG grammars also define explicitely all dependency relations (Pollard & Sag, 1994) while LTAG

grammars are limited by a tree structure which is problematic for, e.g. coordination and equi-verbs. Finally, there is a large amount of linguistic research which done in the HPSG framework.

Moreover, studying how such a compilation can be performed is an opportunity to identify the assets and the limits of the LTAG formalism. Which relations given in a HPSG grammar should be localized in the LTAG elementary trees in order to obtain a grammar that is either linguistically meaningful or computationally efficient?
We first recapture the basic principles of the compilation algorithm as described in (Kasper *et al.*, 1995). Then we present the various problems and limits of this initial algorithm and the adaptations that have been necessary for the practical HPSG-to-TAG compilation of a wide-coverage grammar.

2. The initial compilation algorithm

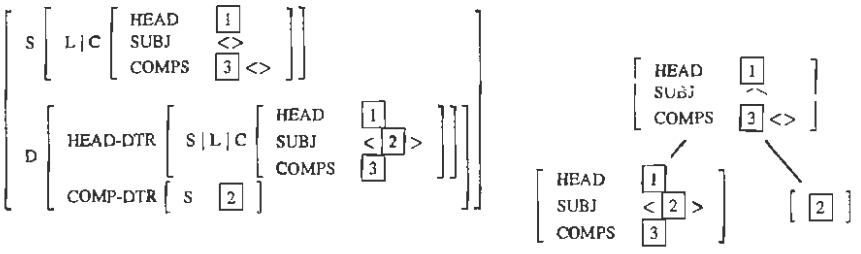

Figure 1: HPSG Head-Subj Schema and its representation as a local tree.

We assume that the rule schemata in the HPSG grammar only correspond to binary or unary rules. For instance, the *Head-Subj-Schema* given in figure 1 can be represented by a partial tree. The algorithm presented in (Kasper *et al.*, 1995) is based on the following mechanisms:

- **Selection/Reduction process:** The features which constrain a possible argument are called *Selection Features* (SF). Given a binary schema S, if some SF are expressed in S, we say that the daughter which contains these features is the *Selection Daughter* (SD), the other one is the non-SD. The single daughter of unary rules is the SD. Given the SF of the SD, we say that a schema reduces the SF, if the value of at least one of the features that select the non-SD for this schema is not contained in the feature value of the mother node. In the example figure 1, we see that the SF of attribute SUBJ is reduced.

- **Tree production iteration:** The basic algorithm starts with the creation of a node for the lexical type. A root node n is first added to this initial node with a copy of all its features. Then we instantiate each schema S which actually reduces at least one SF of n when n is unified as the SD of S. Finally, we add an additional root node dominating the instantiated schema. This step is repeated until the *termination condition* is met (see below).

- **Raising Features Across Domination Links:** this principle determines which features are raised (copied) into the additional root nodes. In the first phase of the algorithm, all and only the SF are raised. In the additional phases, some SF are not raised (see below).

- **Detecting foot nodes:** A tree is an auxiliary tree if the root node and one of the leaf nodes (non-anchor) have some non-empty SF value in common. This leaf node becomes the foot node of the auxiliary tree.

- **Termination:** A SF is not reduced anymore if its value is an empty list or it shares its value with a feature at a leaf node other than the foot node.

- **Additional phases:** Systematically raising all possible SF across domination links (i.e., considering only complete projections) results in redundant projections for multiple dependency structures as raising verbs or equi-verbs and consequently corresponding trees that can not be combined. In order to avoid these redundant projections, (Kasper *et al.*, 1995) propose additional phases in order to create new trees without redundant projections for double dependencies. Their decision is to keep the redundant dependencies in the auxiliary trees and consequently re-compile all initial trees, ignoring the SF which are responsible for the redundancy.

At the end of the process, the SF can be deleted from the resulting trees since they express constraints that have been captured in the tree structure. The next section will show that this initial algorithm raises both practical and theoretical problems.

3. Algorithmical problems

3.1. Choice of Selection Features

A given phrase structure (derived tree) can be obtained with different LTAG grammars, where the derivation trees might differ. A lot of choices in the compilation process (SF & SD) depend on the kind of derivation tree we want to obtain and its role given a particular task (generation or parsing). Moreover, the algorithm proposed in (Kasper *et al.*, 1995) aims to capture the phrase structure of the HPSG grammar in the LTAG structure, which is only one choice among other possibilities.

However, even the original algorithm leaves open the choice of SF. This choice, together with the termination criteria influences the resulting elementary trees (and thus the dependency structures) while the derived trees are still isomorphic to the HPSG derivation. In theory, the SF must be chosen such that at least one of them is reduced in every HPSG schema. In practice no such set of SF can be determined and some schemata must be applied in the compilation algorithm with less strict criteria such as a mere change (without reduction) of SF or even (non-recursive) applications with both daughters as possible SD.

The interface between deep syntax and derivation trees highly depends on the LTAG grammar resulting from the compilation algorithm and thus from the choice of SF and termination criteria. Since HPSG is based on lexical projections as expressed, e.g., in the *head feature principle*, there is a certain straightforward choice of SF. However, the HPSG schemata localize syntactic dependencies and not semantic dependencies as classically in LTAG grammars. Especially in generation, when mapping from semantic dependencies, this mismatch becomes apparent, e.g. in auxiliaries (see section 5), raising and equi-verbs, modifier extraction, etc.

As an example for how the choice of selector feature can change the selector daughter for an HPSG schema and thus the resulting elementary TAG trees, we look at the Head-Specifier schema in the HPSG grammar we use. Figure 2 shows how choosing either SPR or SPEC as a SF results in an auxiliary tree anchored at DET (β_1) or an initial tree anchored at N (β_2) respectively. The surprisingly different structures are possible due to a case of double dependencies, where the SPR and SPEC features mutually constrain each other.

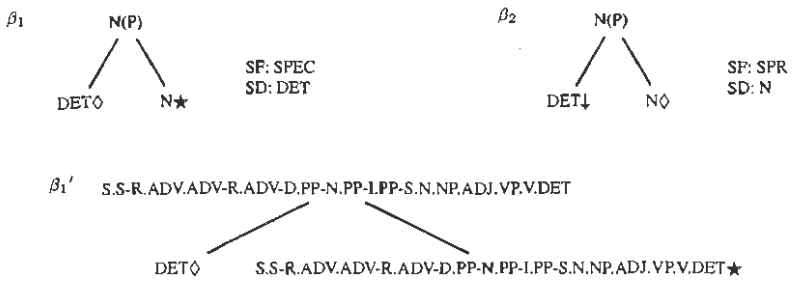

Figure 2: Possible projections for the HPSG Head-Specifier Schema.

Whenever the selector daughter is *not* the head daughter, the property of HPSG that it is *head driven* becomes important: because almost all information (i.e., features) that is raised comes from the head (the non-selector daughter in this case), the root node is very much underspecified in these cases. Thus, e.g. the category of root and foot node is highly underspecified. See the example β_1' in figure 2 (the dot in the node labels indicates a disjunction of categories).
Note however, that in the case of the Adjunct-Head schema, where the selector daughter is clearly not the head daughter, the MOD feature supplies a lot of constraints about the head. Thus the MOD feature is used as a selector feature. However, the HPSG grammar often encodes constraints in the semantics of the MOD feature which does not immediately constrain the category.

3.2. *Adequacy of HPSG and LTAG categories*

The HPSG grammar does not define statically the syntactic categories of nodes as in the elementary trees of LTAG grammars. The original algorithm assumes that all possible values of the SF appear during the compilation and thus can be mapped (collapsed) to a finite set of not obviously meaningful categories. In the HPSG framework, syntactic categories are usually computed only for complete derived trees and they are represented as (disjoint) underspecified feature structures (typically with values only for the SF) such that in a derived tree only one syntactic category (feature structure) unifies with a node. Using these HPSG categories, one gets a set of meaningful categories.
But since elementary trees resulting from the compilation algorithm correspond to *partial* parsing trees obtained by the application of several HPSG schema, it is not possible to determine a unique syntactic category per node in the compiled LTAG elementary trees. As a consequence, nodes of elementary trees must often be labeled with a disjunction of syntactic categories. Duplicate trees in order to avoid such disjunctions would result in a critical explosion of the number of trees. Note that we can observe in our resulting compiled grammar disjunction of more than twenty syntactic categories.

3.3. *Raising of non-SF features*

Following (Kasper *et al.*, 1995), only SFs are raised across dominance links. In practice, non-SFs features are very important for the selection and the filtering of the HPSG schema that are applicable in production of an elementary tree. Without raising them across dominance link, too many HPSG schema would be applied, resulting in an dramatically overgenerating and much larger grammar. Naturally, raising non-SF features can result in an *undergenerating* grammar. But in generation, this is often less of a problem than overgeneration. Also, by extending (relaxing) the termination criteria, we can ensure the generation of all necessary elementary

trees.

3.4. Anchoring the Projection

In HPSG as in LTAG, there is a separation between the grammar and the lexicon such that a lexical entry specifies the word, its semantics and a *lexical type* or *tree family*. If this separation is clean in the HPSG grammar, as it is in our case, the compilation process can start from the lexical types and becomes independent from the lexicon.
However, many lexical types only differ wrt. semantics and the compilation process only extracts the syntactic part, so either (i) for a set of lexical types that generate the same tree-families, we must determine the most specific subsuming type in the type hierarchy or (ii) eliminate redundancies in a post-process.
Note that since the HPSG grammar has the option to locate constraints either in syntax or semantics, some of the syntactic features are highly underspecified. This leads to the above-mentioned redundancies between the syntax of lexical-types and also to the underspecifications in the syntactic categories.
We also have found another source of redundancy: a sizable number of trees appear in more than one tree-family and a further reduction could be achieved by storing them only once an introduce pointers to the tree-families.
Note that this kind of underspecification seems very undesirable but it is inherent in the specifications of the HPSG grammar and therefore cannot be avoided easily. The only principled solution is a change of the HPSG grammar.

4. Specific linguistic phenomena

4.1. Coordination

Coordination is problematic in the LTAG formalism since the tree structures are not able to localize the multiple dependencies that this phenomena introduces. The HPSG analysis of this phenomena exploits at the syntactic level the type hierarchy of features, particularly by introducing new morpho-syntactic ones. Feature coindexing at the level of the semantics are also used for crossed-dependencies for instance. These techniques are really far from the existing ones in the LTAG-world based on explicit structural dominance link (Sarkar & Joshi, 1996). The processing solutions of this phenomena in the two respective formalisms are too specific to expect the capture of the HPSG approach by TAG.

4.2. Double Dependencies

The double dependencies, where a given phrase structure is the argument of two different predicates, are a problem in the LTAG formalism which can only capture one of these dependencies in the structure of elementary tree. For equi-verbs for instance, as the verb *want*, the classical choice corresponds to the elementary trees given in figure 3. These trees are obtained by the initial HPSG-to-TAG compilation algorithm (β_1 and α_1) but the additional phases generate also some other trees ignoring some SF, i.e. some predicate-argument relations (β_2 and α_2). Considering that the elementary trees generated for the main verbs localize all possible set of predicate-argument relations (see figure 3), we obtain redundant projections of substitution nodes. One can see that we obtain the same derived tree by combining β_1 and α_2 or β_2 and α_1, but in both cases only a part of the dependencies are captured.
Consequently we can question the relevance of the multiple phase part of the algorithm. Fully executing the additional phases as described in the original algorithm is impractical since it generates far too many trees. Therefore, we have added by hand those extensions of the termination and raising criteria that are needed to obtain the elementary trees needed in our domain.

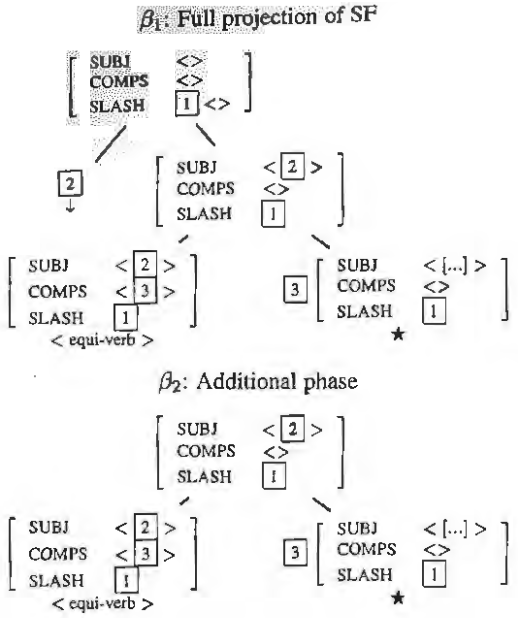

Figure 3: Elemenary trees for equi-verbs.

In general, the construction of elementary trees stops when SF are reduced to empty features or lists. In practice however, SF often are never empty and only a detailed analysis of the content (e.g., the type) of the SF can determine whether the projection must stop, i.e., that the SF can/must not be reduced further. Also, in order to mimic the effect of the additional phases (see paragraph below), we have to relax the termination criteria to apply even when some SF are not reduced. E.g., the projection of auxiliaries results in VP substitution nodes, thus adding VP nodes (with a reduced COMP feature and an unreduced SUBJ feature) to the list of terminating nodes.

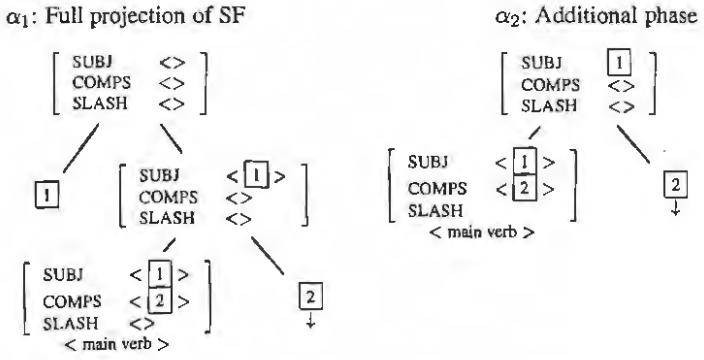

Figure 4: Elemenary trees for main verbs.

4.3. Idioms

LTAG can represent idioms directly, including the fact that idioms have a single (non-compositional) semantic representation. Since HPSG grammars cannot do this directly, this is also true for the compiled TAG grammar. Even multiple anchors (like in particle verbs and the (semantically empty) prepositions of prepositional arguments) are not incorporated into the resulting trees of the compilation algorithm. One solution would be the extension of the compilation algorithm to expand those leaf nodes (e.g., prepositional complements) that include semantically empty, syntactic arguments. I.e., instead of a PP substitution node, expand it to its anchored P daughter and its NP substitution daughter. Since this would add even more trees, we have instead chosen to include these expansions either in the microplanning or the preprocessing (see section below) phases.

4.4. Futher issues

The LTAG formalism can not capture all long dependencies that can be represented easily in HPSG with feature percolation. One way to capture these phenomena is to compile the HPSG grammar into an extension of the LTAG formalism as the DTG formalism (Rambow *et al.*, 1995).

In the current version of the algorithm, semantics is not taken into account. We have conducted some experiments by compiling the semantic level in a specific LTAG grammar that could be synchronized to the classical compiled LTAG grammar. The interest of this approach highly depends on the compositionality of the resulting semantic grammar which still needs some futher investigations.[1]

One can also note that some linguistic constraints are not represented in the usual HPSG grammars, such as modifier, e.g. adjective, ordering and topic/focus distinctions.

5. Interface to the non-syntactic level of HPSG

As discussed above, the dependency structure of the resulting TAG grammar depends mainly on the dependencies that are specified in the HPSG grammar and the choice of selector features only has a limited effect. This is especially important when generating with the resulting TAG grammar. Typically, the input to a syntactic realizer that is based on TAG will be a dependency structure that can be interpreted as the derivation structure. As an example from our system, the HPSG grammar specifies auxiliaries as the lexical heads of sentences, taking a subject and a VP as arguments. So the microplanning step in the generator that maps from the semantic input to the syntactic dependencies must not only plan word choice and map the semantic roles to syntactic arguments (e.g., the *giver* to a *subject*), it also must be prepared to insert an auxiliary (e.g., *have*) and rearrange syntactic arguments (e.g., ensure that the *giver* becomes the subject of *have* and not *give* in *We have given..*). In order to keep a more general interface between microplanner and syntactic realizer, we have chosen not to include the auxiliaries in the microplanner but rather add a preprocessing module to the syntactic realizer which adapts the dependency structure to the specifics of the HPSG/TAG grammar. Thus we can switch to other syntactic realizers (based on other TAG grammars) more easily.

This touches on a more general point which is not really discussed in the original work: The interface between the extracted subgrammar and the full HPSG grammar. As proposed, the compiled TAG grammar actually overgenerates since it represents all possible phrase structures

[1] Since the extraction of just a subset of the features of the HPSG grammar amounts to an (overgenerating) approximation, it is very important to include as many of the constraining features as possible. Many of them are entangled into the semantics though, so a clearer separation in the HPSG grammar is needed. See also the work on context–free approximation of HPSG in (Kiefer & Krieger, 2000).

but omits some of the constraints, especially those of semantics. Ideally, the semantics of the HPSG grammar would be purely compositional, thus the compiled TAG language would be identical. However, in practice there are non-compositional elements in the HPSG semantics and it turned out to be impractical to extract the semantics for every compiled elementary tree and use these partial semantic expressions for microplanning.[2] Thus we have developed the microplanning rules only semi-automatically which also allowed the inclusion of a large subset of planning rules that deal robustly with all kinds of problems in our input (Becker *et al.*, 2000).

6. Practical Results and Conclusion

In the context of the generation module of the Verbmobil project (Becker *et al.*, 1998), we have implemented our adaptation of the compilation algorithm in Common Lisp as an addition to the PAGE system which is used to specify and parse the HPSG grammars. We currently cover an English and a Japanese grammar; the English grammar has around 350 lexical types and 40 schemata and a lexicon with around 6,800 entries. The Japanese grammar has a similar size, with a smaller lexicon. Compilation takes about 15 minutes CPU time on a 400MHz Ultrasparc resulting in around 2,500 elementary trees.

We found the adapted compilation process to be useful in a real system, since we could influence the design of the HPSG grammars, which is an important factor. Also, work on a German HPSG grammar is under way. Given the growth in computational power, we hope to be able to explore a complete application of the original algorithm in the near future.

References

BECKER T., FINKLER W., KILGER A. & POLLER P. (1998). An efficient kernel for multilingual generation in speech–to–speech dialogue translation. In *Proceedings of COLING/ACL-98*, Montreal, Quebec, Canada.

BECKER T., KILGER A., LOPEZ P. & POLLER P. (2000). An extended architecture for robust generation. International Natural Language Generation Conference (INLG), Mitzpe Ramon, Israel.

KASPER R. (1992). Compiling Head-Driven Phrase Structure Grammar into Lexicalized Tree Adjoining Grammar. In *Proceedings of the TAG+ workshop 1992*, University of Pennsylvania, Philadelphia.

KASPER R., KIEFER B., NETTER K. & VIJAY-SHANKER K. (1995). Compilation of HPSG to TAG. In *Proceedings of ACL'95*, p. 92–99, Cambridge, Mass.

KIEFER B. & KRIEGER H.-U. (2000). A context–free approximation of head–driven phrase structure grammar. In J. CARROLL, Ed., *Proceedings of the Sixth International Workshop on Parsing Technologies, IWPT2000*, p. 135–146, Trento, Italy.

POLLARD C. & SAG I. (1994). Head-driven phrase structure grammar. In *CSLI series*. University of Chicago Press.

RAMBOW O., SHANKER K. V. & WEIR D. (1995). D-Tree Grammars. In *33rd Conference of the Association of Computational Linguistics (ACL'95)*, p. 151–158.

SARKAR A. & JOSHI A. (1996). Coordination in tree adjoining grammars: Formalization and implementation. In *COLING'96, Copenhagen*, p. 610–615.

[2]To make things worse, the HPSG grammar only includes an intermediate semantic representation (MRS) in its declarative feature structures and the semantic representation that is actually used in our system is derived by procedural code.