In: Proceedings of the ESSLLI '03 workshop "Ideas and Strategies for Multilingual Grammar Development". Vienna, Austria.

Modularity of grammatical constraints in HPSG-based grammar implementations

Detmar Meurers **Kordula De Kuthy** **Vanessa Metcalf**
Department of Linguistics, The Ohio State University
{dm,kdk,vmetcalf}@ling.osu.edu

Abstract

This paper is a contribution to the discussion of the choices involved in implementing HPSG-based grammars and their consequences on the modularity and reusability of grammatical resources, a central issue in multi-lingual grammar development. Based on two examples from the English Resource Grammar (Flickinger et al., 2000), the treatment of unbounded dependencies and the analysis of optional arguments, we show that adding recursive relations to the expressive means available to the grammar writer results in more modular, transparent, and compact grammars. Given the frequent use of recursive relations in HPSG linguistics, grammar implementations integrating such relations can also be closer to the linguistic generalizations they intend to implement.

1 Introduction

The organization of a grammar in layered, reusable structures is a key methodological issue for multi-lingual grammar development, and arguably for sustainable grammar implementation efforts in general. This insight is reminiscent of the development in computer science, where as a result of the rise of software engineering in the 70s (Naur and Randell, 1968; Parnas, 1975) modular software written in high-level programming languages replaced the low-level coding of the early years. The abstraction and data encapsulation possibilities of high-level languages are viewed as essential to obtain reliable, maintainable and reusable software modules. As a result there is general agreement that efficiency should not be sought by coding at a low level but by intelligent compilation from such a high-level language to executable code.

In this paper, we want to contribute to a discussion of the expressive means of grammar implementation systems and how they can support the formulation of modular grammatical constraints that are reusable across languages. More specifically, we want to investigate the usefulness of recursive relational constraints for the implementation of HPSG-based grammars under this perspective. We base our discussion on two examples, the encoding of the treatment of unbounded dependencies and the analysis of optional complements, two key components of the English Resource Grammar (ERG, Flickinger et al., 2000) as the largest HPSG-based grammar for English currently available. We contrast the ERG encoding of these two issues with encodings that make use of recursive relations, such as the append or union relations frequently used in HPSG linguistics.[1]

2 Example 1: Unbounded dependencies

The English Resource Grammar (ERG) developed by the LinGO[2] project is a freely available broad-coverage, HPSG-based grammar of

[1] Relational goals in HPSG linguistics are often written in functional notation, e.g., $\boxed{1} \oplus \boxed{2}$ instead of append($\boxed{1}$,$\boxed{2}$,$\boxed{3}$).

[2] Cf., http://lingo.stanford.edu/

English, which is implemented in the LKB system (Copestake and Flickinger, 2000). The grammar contains a wealth of analyses of English phenomena, including a coverage of unbounded dependencies that is based on the proposal in Bouma et al. (2001) (henceforth: BMS). This proposal is particularly interesting under an implementation perspective since it replaces the need for computationally problematic empty elements with a lexical specification of ordinary, visible elements. The so-called SLASH amalgamation constraint ensuring this lexical specification is shown figure 1.

$$word \Rightarrow \begin{bmatrix} \text{LOC} \begin{bmatrix} \text{CAT} \begin{bmatrix} \text{DEPS} & \langle [\text{SLASH } \boxed{1}], \ldots, [\text{SLASH } \boxed{n}] \rangle \\ \text{BIND} & \boxed{0} \end{bmatrix} \end{bmatrix} \\ \text{SLASH} \quad \left(\boxed{1} \cup \ldots \cup \boxed{n}\right) - \boxed{0} \end{bmatrix}$$

Figure 1: SLASH amalgamation as defined in Bouma et al. (2001, p. 20)

This constraint on words collects the information about dependents of that word which are not locally realized. More concretely, the SLASH value of each of the dependents on the DEPS list is collected and the SLASH value of a word is specified to be the union of the collected values (minus the value of BIND, which in the following is ignored). To express this generalization, the SLASH amalgamation principle of BMS in figure 1 makes use of the recursive relations set union ($\cup$) and set complement ($-$), as well as the use of $\boxed{1}, \ldots, \boxed{n}$ to express a relation accessing the SLASH value of every element of the DEPS list to be unioned into the SLASH value of the word.

In an implementation platform that supports relational constraints, the SLASH amalgamation principle can be directly expressed. For example, figure 2 shows how one can encode SLASH amalgamation in the Trale system (Meurers et al., 2002), an extension of the ALE parsing and generation system (Carpenter and Penn, 1996).[3]
The relational goal `collect_slashes` collects the SLASH values of all elements of the DEPS list. Its definition in figure 3 specifies that it applies when the list of dependents `Deps` is known to be

```
word *> synsem:(loc:cat:deps:Deps,
                nonloc:slash:Slash)
        goal collect_slashes(Deps,Slash).
```

Figure 2: SLASH amalgamation in Trale

either an empty or a non-empty list. In the latter case, `coll_slashes_aux` peels off one dependent at a time and unions each `Slash` value to obtain the list of all slashes.[4]

```
collect_slashes(Deps,Slash) if
  when(Deps = (e_list;ne_list),
       coll_slashes_aux(Deps,Slash)).

coll_slashes_aux([],[]) if true.
coll_slashes_aux([(nonloc:slash:Slash)
                  |Deps], AllSlash) if
       collect_slashes(Deps,DepsSlash),
       union(Slash,DepsSlash,AllSlash).
```

Figure 3: Definition of `collect_slashes`

Since the ERG is implemented in the LKB system, which does not support relational goals, the SLASH amalgamation approach of BMS is encoded by unfolding the relation between the SLASH of a word and that of its dependents into all of the different ways in which a word in this grammar can select arguments. To be able to do this, one needs to know more about the grammar to be able to determine the maximal number of elements for which the recursive relation needs to be unfolded. For the ERG, the maximal number of arguments of a word is four, so that SLASH amalgamation for arguments can be encoded by five type constraints as shown in figure 4 on the next page.

Let us take a close look at what distinguishes the two encodings in terms of generality, modularity, and transparency:

First, the principle in figure 1 is more modular since it encodes the collection of the SLASH value from any number of dependents without requiring additional information about the specifics of the grammar. Unfolding the principle into a fixed number of disjunctive cases, on the other hand, is dependent on such additional knowledge, namely the maximal number of elements that SLASH needs to be amalgamated from, which

[3] This is essentially the encoding used in the German HPSG-based grammar (Meurers and De Kuthy, 2001) developed in Trale as part of the SFB 340 Project B8 (http://www.sfs.uni-tuebingen.de/hpsg/archive/sfb340-b4-b8/index_engl.html).

[4] Note that one could also use a difference list encoding for SLASH sets in order to avoid this call to the relation union, an issue which is orthogonal to the one we focus on in this paper.

```
basic_zero_arg := lex_synsem &
  [ LOCAL.ARG-S < >,
    NON-LOCAL [ SLASH 0-dlist ]].

basic_one_arg := canonical_synsem &
  [ LOCAL.ARG-S < [ NON-LOCAL [ SLASH #slash ]] >,
    NON-LOCAL [ SLASH #slash ]].

basic_two_arg := lex_synsem &
  [ LOCAL.ARG-S < [ NON-LOCAL [ SLASH [ LIST #smiddle,
                                        LAST #slast ]]],
                  [ NON-LOCAL [ SLASH [ LIST #sfirst,
                                        LAST #smiddle ]]] >,
    NON-LOCAL [ SLASH [ LIST #sfirst,
                        LAST #slast ]]].

basic_three_arg := lex_synsem &
  [ LOCAL [ ARG-S < [ NON-LOCAL [ SLASH [ LIST #smiddle2,
                                          LAST #slast ]]],
                    [ NON-LOCAL [ SLASH [ LIST #sfirst,
                                          LAST #smiddle1 ]]],
                    [ NON-LOCAL [ SLASH [ LIST #smiddle1,
                                          LAST #smiddle2 ]]] > ],
    NON-LOCAL [ SLASH [ LIST #sfirst,
                        LAST #slast ]]].

basic_four_arg := lex_synsem &
  [ LOCAL [ ARG-S < [ NON-LOCAL [ SLASH [ LIST #smiddle3,
                                          LAST #slast ]]],
                    [ NON-LOCAL [ SLASH [ LIST #sfirst,
                                          LAST #smiddle1 ]]],
                    [ NON-LOCAL [ SLASH [ LIST #smiddle1,
                                          LAST #smiddle2 ]]],
                    [ NON-LOCAL [ SLASH [ LIST #smiddle2,
                                          LAST #smiddle3 ]]] > ],
    NON-LOCAL [ SLASH [ LIST #sfirst,
                        LAST #slast ]]].
```

Figure 4: The five type constraint encoding SLASH amalgamation in the ERG[6]

in the ERG happens to be four.

Second, the fact that the principle of BMS can collect the SLASH of any number of elements also means that it is more general than the ERG encoding. Even with specific knowledge about the grammar, it is impossible to capture the full generality of the BMS proposal in the ERG since the key idea of that paper is to generalize over adjunct and argument extraction—but the number of adjuncts is not lexically bounded, so that it is impossible to unfold all potential instances of amalgamation. It is therefore not surprising that in the ERG only argument extraction is handled via SLASH amalgamation, not dependent extraction in general, as proposed by BMS.[7]

Third, a lack of generality and modularity of the ERG encoding derives from the fact that the ERG encoding employs five independent constraints having five different types as antecedent. So while the principle of figure 1 on the page before applies to all words and thus is dependent only on what the linguist or grammar writer decided to classify as a word in the grammar, the ERG unfolding of SLASH amalgamation depends on five separate classifications, one for each type of antecedent. As before, this is a loss of modularity since an understanding of the ERG encoding of SLASH amalgamation is dependent on knowing where five types are used in the specification of lexical entries in the grammar, whereas the original principle of figure 1 on the preceding page only requires knowledge of where a single type, *word*, is used in the specification of lexical entries in the grammar.

Fourth, a further lack of generality and transparency of the ERG encoding is caused by the fact that the ERG encoding imposes five type constraints having five different consequents, whereas a recursive encoding consists of a base clause and a recursive clause characterizing what is the case at $n + 1$ based on knowledge of the state of affairs at n. The recursive case thus is a generalization over all cases, starting from the base case; with a recursive definition it is impossible for e.g. the fourth case to differ from the third case in any other way than exactly the way in which the third case differed from the second case. In contrast, there is no such generality across cases for five separately written down constraints, such as in the ERG encoding of SLASH amalgamation in figure 4. How transparent would it be if, e.g., a couple of variable names in `basic_tree_arg` would be changed such that the SLASH value of one of the three arguments is not collected? Interestingly, closer inspection of the ERG encoding reveals just such a non-generality across cases: the `basic_one_arg` case happens to be special in that it requires the word from which it collects slashes to have a `canonical_synsem`, whereas all other cases require a `lex_synsem`. There thus is a clear contrast to the SLASH amalgamation principle of BMS which generalizes over all cases. This generalization is not expressed in the ERG encoding; instead, one needs to look at every one of the five constraints separately to know what

[7] In languages exhibiting coherence or restructuring phenomena (e.g., German, Dutch, and the Romance languages), even the number of arguments is not bounded in the lexicon since under the normal HPSG analyses of those languages, certain verbs are specified to attract the arguments of their complement.

[7] The constraints are shown without the specification of the QUE and REL attributes. These are amalgamated as well, so that the actual ERG constraints are three times as large as the ones shown in figure 4.

exactly happens to be encoded in each one.

To address the gap in generality, modularity and transparency between the linguistic principle of BMS and the ERG encoding of it, the recursive relations used in the linguistic principle need to be supported by the grammar implementation system. To address all of the issues we raised above, including the unbounded number of potential dependents, recursive relations need to be fully supported by the system in the sense that the runtime environment must support the execution of recursive relations. The overhead associated with such a runtime support of relations can often be avoided though by unfolding and inlining the relation calls at compile time. This corresponds to inlining of functions and unfolding of loops as a standard option of compilers for many programming languages. The use of SLASH amalgamation made in the ERG is an instance where unfolding of the relation at compile-time is possible (unless SLASH amalgamation is also applied to adjuncts). Note that as long as the grammar is specified with a relation explicitly encoding the generalization to be captured, most of the shortcomings of the ERG encoding we discussed above do not apply, independent of whether the relational constraints are ensured by unfolding them at compile-time or by executing them at run-time.

3 Example 2: Optional complementation

The second example we want to look at in this paper concerns the analysis of optional complements in the ERG, which is also discussed in Flickinger (2000).[8] The empirical issue of verbs with optional complements is illustrated by the sentences in (1), which are licensed by the ERG.

(1) a. Kim bet Tom five dollars that they hired Cindy.
b. Kim bet Tom five dollars.
c. Kim bet Tom that they hired Cindy.
d. Kim bet five dollars that they hired Cindy.
e. Kim bet five dollars.
f. Kim bet that they hired Cindy.
g. Kim bet Tom.
h. Kim bet.

In sentence (1a), the verb *bet* takes a subject *Kim* and three complements, the NPs *Tom* and *five dollars,* as well as the sentential complement *that they hired Cindy*. The other sentences in (1) exemplify that each of those three complements is optional.

The brute-force method for licensing these structures would be to posit eight independent lexical entries for *bet*, one for each of the environments exemplified above. But this would miss the generalization that *bet* has three complements, each of which can be realized or not. As discussed by Flickinger (2000), the ERG takes this generalization into account and posits only the single lexical entry shown in figure 5.[9]

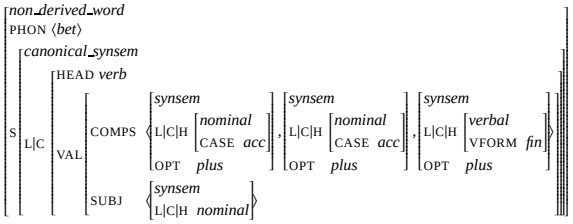

Figure 5: Lexical entry for *bet*

The key aspect here is the specification of the complement requirements on the COMPS list. The list contains three elements, each of which is marked as optional with the help of an attribute OPT(IONAL) appropriate for *synsem* objects.

In figure 6 on the following page we see the structure that is licensed for a sentence in which none of the optional complements are realized, i.e., sentence (1h). The entry of *bet* can construct as the head daughter of such a head_subject_phrase even though it has not yet realized its complements. This is possible since, different from the traditional HPSG analysis (Pollard and Sag, 1994), the head daughter is not required to be saturated, i.e., have a COMPS value of type *e_list*. Instead, a sign is also understood to be saturated for complements if it has only optional complement requirements left.

[8] Whether the treatment of optional complements proposed in Flickinger (2000) is the best analysis for this phenomenon is an orthogonal issue. Our focus is on how the proposed analysis is reflected in the implementation.

[9] Here and in the following figures, only the specifications relevant to the issue of optionality are shown. For space reasons, attributes are sometimes abbreviated by their first letter.

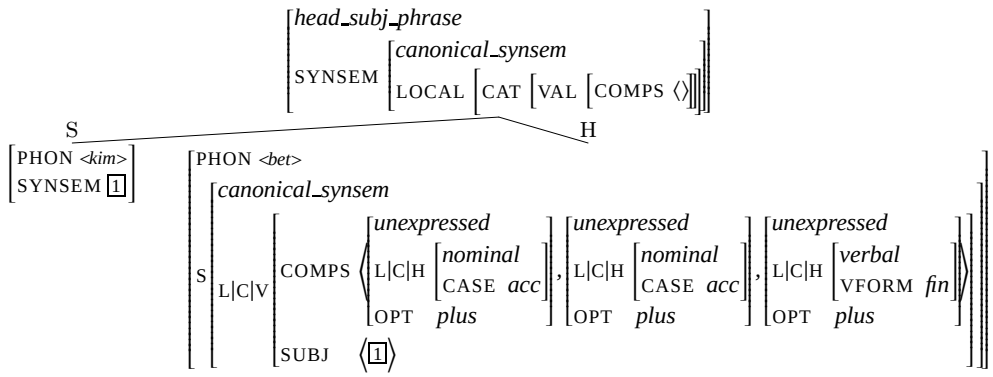

Figure 6: A sentence with three unrealized complements

Adding the head-complement phrase of the ERG to the picture, one can also license (1b) and (1g), which are sentences in which one or two complements are realized and the other complements, which are more oblique than the ones that are realized, are missing.[10] Figure 7 shows the relevant aspects of the definition of head-complement phrases in the ERG. Note that it is always the first element of the COMPS list that is realized as the *non_head_dtr* of such a phrase.

head_comp_phrase →

$$\begin{bmatrix} \text{SYNSEM}\begin{bmatrix} \textit{canonical\_synsem} \\ \text{LOCAL|CAT|VAL|COMPS } \boxed{2} \end{bmatrix} \\ \text{HEAD\_DTR|SYNSEM}\begin{bmatrix} \textit{canonical\_synsem} \\ \text{LOCAL|CAT|VAL|COMPS } \langle \boxed{1}\, \boxed{2} \rangle \end{bmatrix} \\ \text{NON\_HEAD\_DTR|SYNSEM } \boxed{1}\textit{canonical\_synsem} \end{bmatrix}$$

Figure 7: The realization of COMPS requirements in the head-complement rule of the ERG

Figure 8 on the following page shows the structure that the ERG assigns to the sentence (1g). The lower tree is an instance of a head_comp_phrase, in which the first subcategorization requirement on COMPS, namely the NP *Tom* bearing the tag 2, is realized. The *head_subject_phrase* on top is licensed just as in the previous example, marking the remaining optional elements on the COMPS list of the head daughter *bet Tom* as unexpressed.

Since the *head_comp_phrase* in the ERG always realizes the first element of the COMPS list, a problem arises if one wants to license a sentence in which the least oblique complement, i.e., the first element on the COMPS list is optional and not realized. Note that this is not an accidental oversight in the formulation of the rule licensing *head_comp_phrases* in the ERG; rather it is a consequence of the fact that the LKB system does not support relational goals as attachment to phrase structure rules. We will see in the next section that an implementation platform that includes such relational goals can express the relevant generalization, namely that the head_comp_phrase realizes the first requirement on COMPS which is not marked as unrealized optional element. In the ERG as implemented in the LKB system, the problem is addressed by introducing additional types of phrases which eliminate the unrealized optional subcategorization requirements from the front of the COMPS list in order to bring the requirement intended to be realized to the first position of the COMPS list. For this purpose, in addition to the ordinary *head_comp_phrases*, the ERG needs two additional rules: the *head_opt_comp_phrases* which eliminates one optional complement from the front of the COMPS list, and the *head_opt_two_comp_phrases* which eliminate first two complement requirements from the COMPS list. Further additional phrases would be needed if the grammar had COMPS lists longer than three.

Figure 9 on the next page illustrates the structure licensed for sentence (1e), in which only the second most oblique complement is realized. The unary structure at the bottom of the tree is an instance of the additional *head_opt_comp_phrase*, whose purpose is the elimination of the first com-

[10] The COMPS is ordered by obliqueness, with the least oblique complement being the first element of the list.

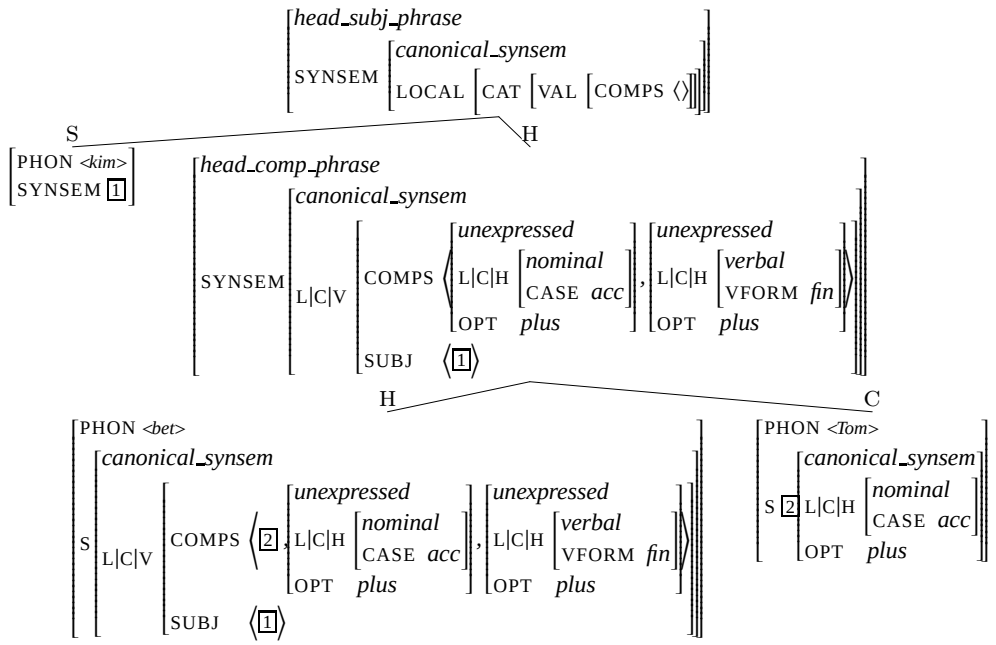

Figure 8: A sentence in which the two most oblique complements are not realized

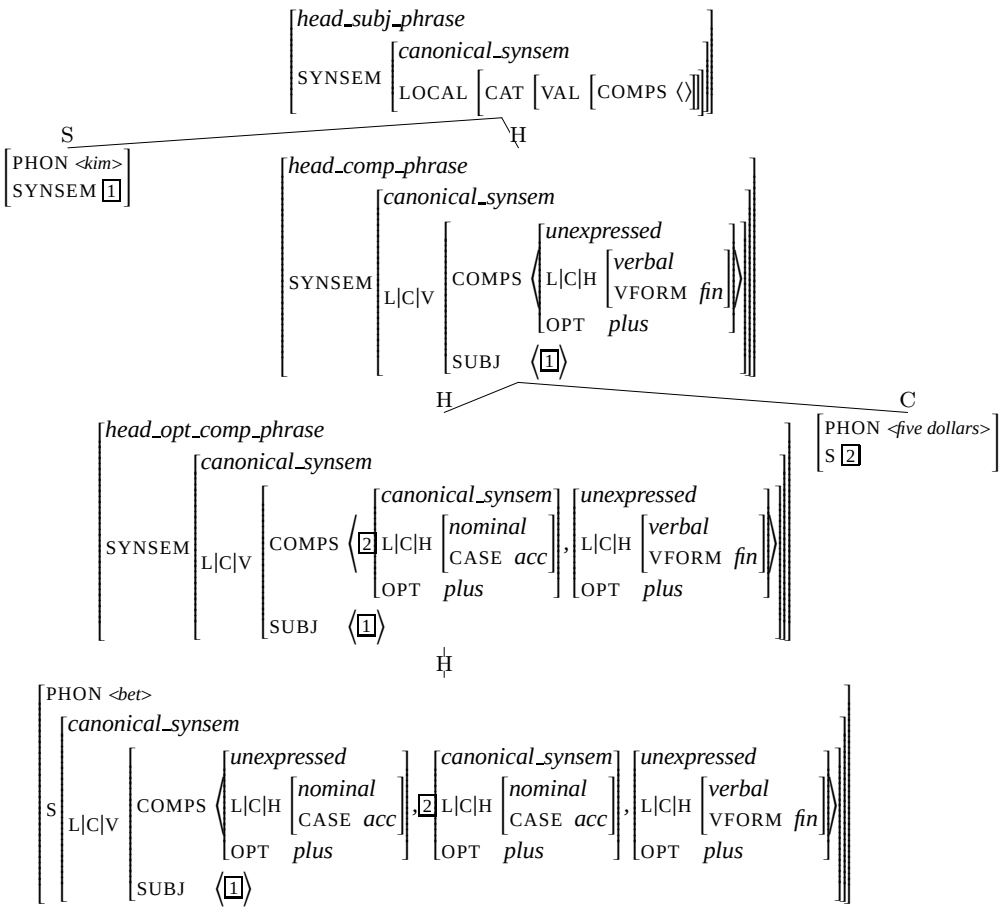

Figure 9: The ERG analysis of a sentence in which only the second most-oblique object is realized

plement requirement, an unexpressed optional object NP, in order to bring the requirement 2 to the front of the COMPS list. That complement (*five dollars*) is then realized in the *head_comp_phrase* dominating the *head_opt_comp_phrase*.

Capturing the missed generalization We saw above that the ERG analysis of optional complements requires three different head-complement rules since in the LKB system there is no way to express the relevant generalization that one wants to realize the first element on the comps list that is not an unrealized optional argument.

The revised head_complement rule in figure 10 shows how the intended generalization can be expressed using an append relation (⊕) to state that the element 1 to be realized can be preceded by an *o_list*, the type used in the ERG to refer to a list of unrealized optional elements. In a grammar

head_comp_phrase →

$$\begin{bmatrix} \text{SYNSEM} \begin{bmatrix} \textit{canonical\_synsem} \\ \text{LOCAL|CAT|VAL|COMPS } \boxed{2} \end{bmatrix} \\ \text{HEAD\_DTR|S} \begin{bmatrix} \textit{canonical\_synsem} \\ \text{LOCAL|CAT|VAL|COMPS } \textit{o\_list} \oplus \langle \boxed{1}, \boxed{2} \rangle \end{bmatrix} \\ \text{NON\_HEAD\_DTR|SYNSEM } \boxed{1}\textit{canonical\_synsem} \end{bmatrix}$$

Figure 10: Generalized realization of COMPS requirements in the revised head-complement rule

including this revised *head_complement_phrase* instead of the original one from the ERG we saw in figure 7 on page 5, the types and definitions for *head_opt_comp_phrases* and *head_opt_two_comp_phrases* are no longer needed. Interestingly, the LKB encoding of the ERG using a *head_complement_phrase* plus the two 'auxiliary' phrase types used to unearth the first complement requirement to be realized can be seen as the result of unfolding the first three calls to the append (⊕) relation in the revised *head_complement_phrase* defined in figure 10, i.e., the LKB encoding can result from a compilation step taking the more general encoding as its input. This means that the issue of enabling the grammar writer to express the full generalization in the grammar as shown by the revised encoding is independent of the as yet unresolved question of the relative efficiency of parsing systems with and without runtime support for relational goals.

4 Summary and Outlook

In this paper we discussed the use of relational constraints in the implementation of HPSG-based grammar in pursuit of a modular and reusable grammar encoding. We based the discussion on two examples from the English Resource Grammar, the largest HPSG-based grammar for English currently available, and an insightful collection of analyses of many aspects of English syntax.

In the first example, we showed that the ERG unfolds the general SLASH amalgamation principle of Bouma et al. (2001) into the specific cases assumed for this particular grammar, which is necessary since relational goals are not supported in the LKB system. The resulting encoding does not capture the full generality of the principle and is less modular and transparent than the formulation of Bouma et al. (2001) or its computational encoding in a framework incorporating recursive relations, such as the Trale system.

In the second example, we discussed how the ERG captures the optionality of arguments through the use of a single lexical entry, coupled with an ontology of markings distinguishing optional from obligatory and unrealized from realized elements. Subject-head and head-complement structures are modified accordingly, but due to the lack of a possibility to use recursive relations in grammars implemented in the LKB system, the ERG analysis fails in treating optional arguments in a general way, requiring two new types of 'auxiliary' phrases which are otherwise unmotivated. The focus on a very lean system without relational goal attachments to phrase structure rules thus results in a loss of generality and thereby transparency and modularity of the grammars that can be expressed. Apart from the software engineering aspect, this also breaks the clear link of the implementation to linguistic theory, which in Copestake and Flickinger (2000) is identified as the hallmark distinguishing the ERG from other grammar implementation efforts such as those around the Alvey Natural Language Tools (Briscoe et al., 1987).

We showed that a recoding of the analysis of optionality in a system supporting relational attachments can overcome this shortcoming by making use of a single recursive relation, append, used

to select the first non-optional argument on a list. A system including relational attachments thus is better suited to achieve the goal of a modular and linguistically informed grammar implementation.

We are in the process of reimplementing the English Resource Grammar to also investigate in practice which expressive means are useful for writing modular and transparent HPSG-based grammars. The Milca English Resource Grammar (MERGE, De Kuthy et al., 2003) is implemented in the Trale system (Meurers et al., 2002), an extension of the ALE parsing and generation system (Carpenter and Penn, 1996). In line with the arguments presented in Götz and Meurers (1997) and standard practice in HPSG linguistics, the Trale system supports both implicational constraints and relational goals such as the ones used in the approach to unbounded dependencies of Bouma et al. (2001) and the analysis of optional arguments of Flickinger (2000) which were discussed in this paper.

References

Gosse Bouma, Rob Malouf and Ivan A. Sag, 2001. Satisfying Constraints on Extraction and Adjunction. *Natural Language and Linguistic Theory*, 19(1):1–65.

Ted Briscoe, Claire Grover, Bran Boguraev and John Carroll, 1987. A formalism and environment for the development of a large grammar of English. In *Proceedings of IJCAI-87*. Milan.

Bob Carpenter and Gerald Penn, 1996. Compiling Typed Attribute-Value Logic Grammars. In Harry Bunt and Masaru Tomita (eds.), *Recent Advances in Parsing Technologies*, Kluwer, Dordrecht, pp. 145–168.

Ann Copestake and Dan Flickinger, 2000. An open source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC-00)*, Athens.

Kordula De Kuthy, Vanessa Metcalf and W. Detmar Meurers, 2003. The Milca English Resource Grammar (MERGE). Trale grammar code and documentation. Department of Linguistics. The Ohio State University. To be released in Dec. 2003; for preliminary versions cf. http://ling.osu.edu/~dm/merge/.

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

Dan Flickinger, Ann Copestake and Ivan A. Sag, 2000. HPSG Analysis of English. In Wolfgang Wahlster (ed.), *Verbmobil: Foundations of Speech-to-Speech Translation*, Springer, Berlin, Artificial Intelligence, pp. 254–263.

Thilo Götz and W. Detmar Meurers, 1997. Interleaving Universal Principles and Relational Constraints over Typed Feature Logic. In *Proceedings of the 35th Annual Meeting of the ACL and 8th Conference of the EACL*. Madrid, pp. 1–8. http://ling.osu.edu/~dm/papers/acl97.html.

W. Detmar Meurers and Kordula De Kuthy, 2001. Case Assignment in Partially Fronted Constituents. In Christian Rohrer, Antje Roßdeutscher and Hans Kamp (eds.), *Linguistic Form and its Computation*, CSLI Publications, Stanford, CA, pp. 29–63.

W. Detmar Meurers, Gerald Penn and Frank Richter, 2002. A Web-based Instructional Platform for Constraint-Based Grammar Formalisms and Parsing. In *Effective Tools and Methodologies for Teaching NLP and CL*. Proceedings of the Workshop held at 40th Annual Meeting of the ACL. Philadelphia, PA.

Peter Naur and Brian Randell (eds.), 1968. *Software Engineering: Report on a conference sponsored by the NATO Science Committee, Garmisch-Partenkirchen, Germany, 7–11 Oct. 1968*. Science Affairs Division, Nato, Brussels.

David Lorge Parnas, 1975. Software Engineering or Methods for the Multi-Person Construction of Multi-Version Programs. In Clemens Hackl (ed.), *Programming Methodology, 4th Informatik Symposium, IBM Germany, Wildbad, September 25-27, 1974*, Springer Verlag, Lecture Notes in Computer Science, pp. 225–235.

Carl Pollard and Ivan A. Sag, 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications.