Documentation of the Implementation of the Milca English Resource Grammar in the Trale system

Kordula De Kuthy
kdk@ling.osu.edu

Vanessa Metcalf
vmetcalf@ling.osu.edu

Detmar Meurers
dm@ling.osu.edu

Revision : 1.31

Contents

Chapter 1

Introduction

The Milca English Resource Grammar (MERGE) is an HPSG-based grammar for English. It is implemented in the Trale system with the intention of exploring and documenting which expressive means are useful for writing HPSG-based grammars. The grammar is modeled on the English Resource Grammar (ERG, version 2002-01-16) (Flickinger et al. 2000) which has been implemented in the LKB system (Copestake, 2002). We are grateful to Dan Flickinger and his colleagues on the German Verbmobil project and member institutions of the LinGO consortium (`http://lingo.stanford.edu/`) for producing the ERG and for making it freely available to the research community. We hope that our work exploring potential alternatives for encoding such a grammar will further progress in the field of HPSG-based grammar implementation and the computational platforms provided to support them.

The discussion in this document focuses on the issues that are related to the implementation of large HPSG-based grammars and, more specifically, the MERGE. A discussion of the general aspects of the logical architecture, the formalization, and the implementation of Head-Driven Phrase Structure Grammars can be found in Meurers (1994).

The general strategy for the implementation of the MERGE in Trale was to stay as close as possible to the ERG in order to facilitate a comparison of the expressive devices used in specific areas. In the chapter 2 we discuss some of these specific areas: In section 2.1.1 we discuss the modularity of grammatical constraints in HPSG-based grammar implementations and how it can be supported by a grammar implementation platform. In section 2.1.2 we turn to the use of types in HPSG-based implementations and argue that a system should clearly distinguish between two distinct notions: the use of types as introducing linguistically necessary distinctions and the use of macros as abbreviations for commonly used specifications. Section 2.2 then discusses correspondences and differences in the implementation of the MERGE compared to the ERG.

Turning to the documentation of the MERGE as such, in chapter 3 we start with a listing of examples for the phenomena covered by the grammar. Chapter 4 discusses the different components in the grammar specifications. On this basis, chapter 5 then explains how the different phenomena are licensed by the grammar.

Chapter 2

From ERG to MERGE

2.1 Some general issues

2.1.1 Modularity of grammatical constraints in HPSG-based grammar implementations[1]

2.1.1.1 Introduction

The organization of a grammar in layered, reusable structures is a key methodological issue for sustainable grammar implementation efforts. 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.[2]

[1] The material of this section is based on our publication by the same name in the Proceedings of the ESSLLI '03 workshop "Ideas and Strategies for Multilingual Grammar Development". Vienna, Austria.

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

2.1.1.2 Example 1: Unbounded dependencies

The English Resource Grammar (ERG) developed by the LinGO[3] project is a freely available broad-coverage, HPSG-based grammar of 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 2.1.

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

Figure 2.1: 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 2.1 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 2.2 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).[4]

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

Figure 2.2: SLASH amalgamation in Trale

The relational goal `collect_slashes` collects the SLASH values of all elements of the DEPS list. Its definition in figure 2.3 on the following page specifies that it applies when the list of dependents `Deps` is known to be 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.[5]

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

[4] 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`).

[5] 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.

```
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 2.3: 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 2.4 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 2.1 on the preceding page 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 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 2.1 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

[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.

```
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 2.4: The five type constraint encoding SLASH amalgamation in the ERG[6]

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 2.1 on page 9 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

[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 2.4.

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 2.4 on the preceding page. 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 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.

2.1.1.3 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.

[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.

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 2.5.[9]

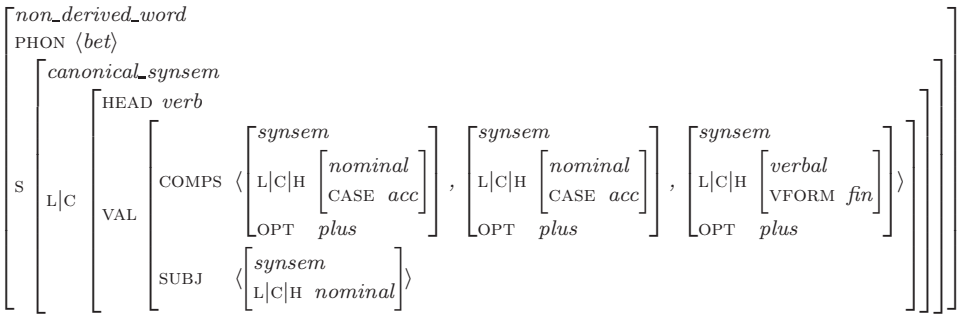


Figure 2.5: Lexical entry for *bet*

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 2.6 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.

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 2.7 on the next page 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.

[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.

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

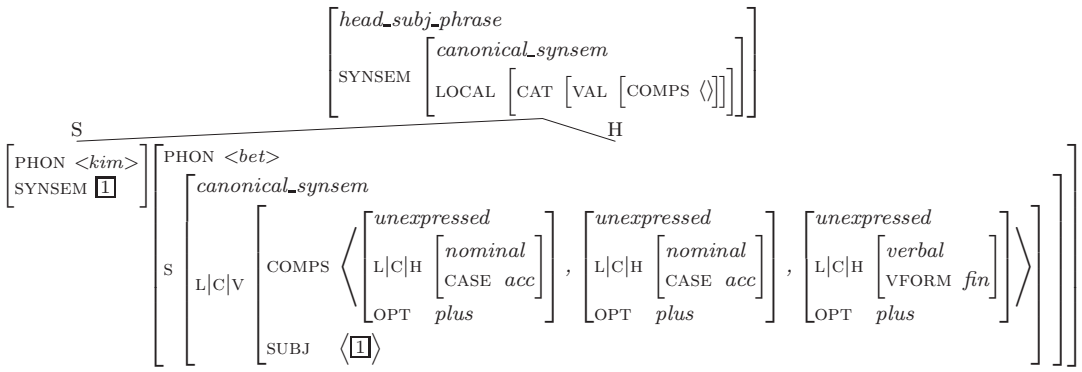


Figure 2.6: A sentence with three unrealized complements

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

Figure 2.7: The realization of COMPS requirements in the head-complement rule of the ERG

Figure 2.8 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

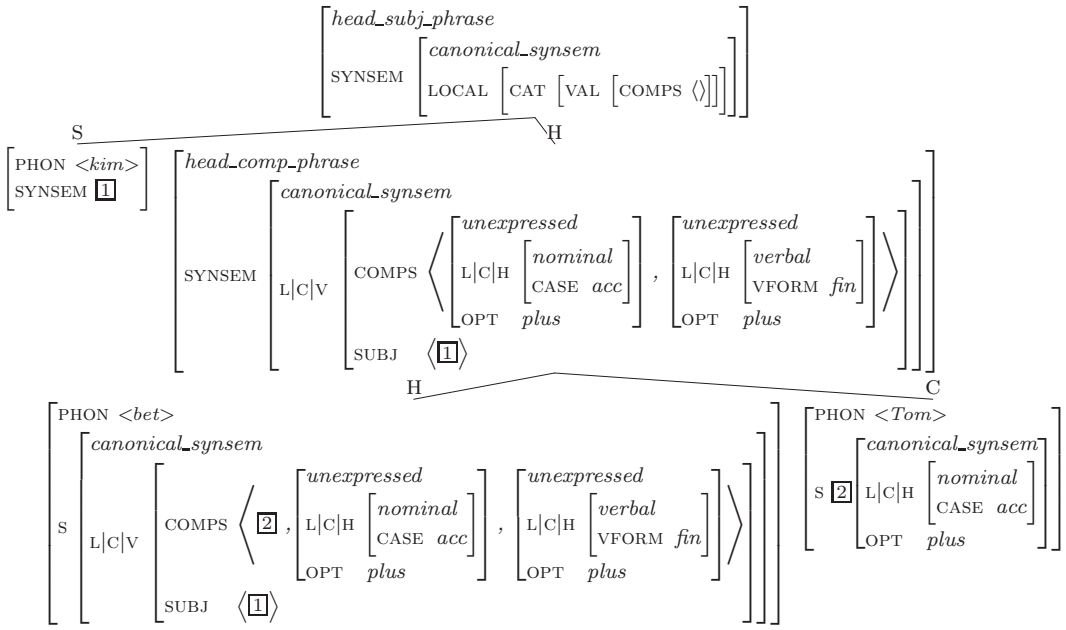


Figure 2.8: A sentence in which the two most oblique complements are not realized

if the grammar had COMPS lists longer than three.

Figure 2.9 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 complement 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 2.10 on page 17 shows how the intended generalization can be expressed using an append relation ($\oplus$) to state that the element $\boxed{1}$ 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 including this revised *head_complement_phrase* instead of the original one from the ERG we saw in figure 2.7 on the page before, 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 ($\oplus$) relation in the revised *head_complement_phrase* defined in figure 2.10 on page 17, i.e., the LKB encoding can result

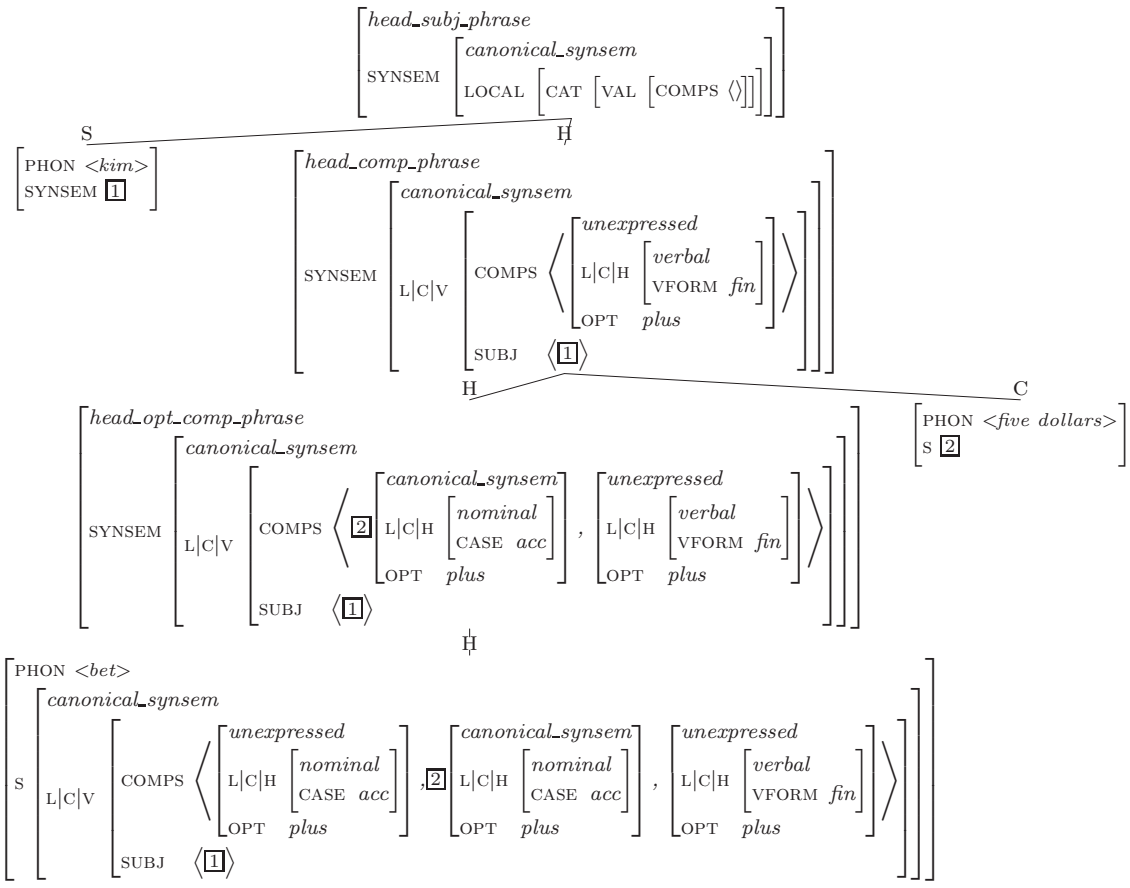


Figure 2.9: The ERG analysis of a sentence in which only the second most-oblique object is realized

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.

2.1.1.4 Summary

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

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

Figure 2.10: Generalized realization of COMPS requirements in the revised head-complement rule

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.

2.1.2 Towards meaningful criteria for data structure and grammar design in HPSG-based implementation efforts

2.1.2.1 Introduction

For parallel development of grammars for different languages, clearly articulated criteria for determining the nature of the data structures to be used and guidelines for the organization of the grammatical constraints are an essential prerequisite. In this section we want to address the issue of developing such criteria for HPSG-based grammar implementation and present some questions which we believe need to be answered to obtain a well-motivated standard practice in grammar development. The questions pertain to where information should be encoded and how it should be organized in order to obtain a compact and transparent gram-

mar. In the context of HPSG-based grammar implementation, this includes the following questions: What motivates introducing a type in a grammar? When should attributes be introduced? How should descriptions used throughout the grammar be organized compactly?

While the design of each grammar implementation platform makes particular assumptions as to how these questions could be answered, there is little explicit discussion of these assumptions. We believe that a thorough discussion of the questions is essential for guiding the further development of implementation platforms towards a well-motivated implementation standard for HPSG-based grammars.

As a contribution to this discussion, this section presents an experiment in which we contrast the use of types in the English Resource Grammar (Flickinger et al., 2000, ERG) with a more conservative use of types in a recoding of the ERG that we have undertaken. Our experiment shows that it is possible to eliminate two thirds of the types that are defined in the ERG without changing the meaning of the grammar in terms of which signs are licensed by the grammar and what linguistic properties are associated with them. This is possible since the types which we eliminated encode distinctions which are already encoded by other types in the grammar. These redundant types thus seem to be an unnecessary complication that should be eliminated to obtain a more transparent and compact grammar.

2.1.2.2 Types and what they are used for

Typed languages make it possible to verify the well-formedness of the complex data structures used in current linguistic theories by requiring an explicit declaration of the modeled domain. In HPSG (Pollard and Sag, 1994), a signature defines which types of objects exist, and which attributes with which values are appropriate for the different types of objects. A hierarchy of types is defined to make it possible to refer to classes and subclasses of objects. To encode a particular distinction to be modeled in the grammar, one thus has a choice of introducing a new attribute as appropriate for a type, or to define new subtypes of that type. For example, one could encode a distinction between inverted and non-inverted verbs as a type distinction under *verb*, or via a boolean attribute defined for *verb*.

Once a type is introduced, the instances of that type can be constrained by specifying a type constraint associating the type with a description of the required values for its attributes. This mechanism is used to express the grammatical constraints in HPSG linguistics, i.e., such type constraints are used to define which linguistic objects are grammatical and which are not.

In HPSG-based grammar implementation, the option of introducing new types and associating them with descriptions through type constraints has also led to a second use of types, which is not motivated by empirical distinctions that need to be made to correctly model the domain. Instead, types are introduced as mere names for collections of descriptions, i.e., abbreviations (Flickinger, 2000). While such a use of types is a possibility, based on an example from the English Resource Grammar, we will illustrate below that it introduces a significant level of unmotivated complexity into the modeled domain.

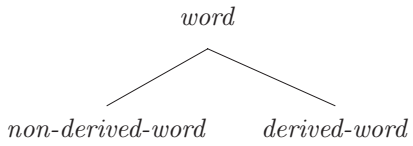
2.1.2.3 Types in the English Resource Grammar

In the version of the English Resource Grammar (Flickinger et al., 2000, ERG) we worked with, there are 3559 distinct types, 1294 of which are glb types which are introduced by

the LKB compiler to obtain unique greatest lower bounds for each pair of types. For our illustration we focus on the 225 subtypes of *word* (ignoring glb types). These types are arranged in the relatively complex type hierarchy under *word* shown in figure 2.11 on the next page.

2.1.2.4 An experiment reducing the number of types to what is empirically required

For the experiment, we went through the subtypes of *word* and considered for each of the types whether it introduces a linguistic distinction that is necessary to formulate the grammatical constraints encoded in the ERG. Each type which was found not to introduce such a necessary distinction was eliminated from the hierarchy and instead defined as an abbreviation to keep the grammar specified compactly.[11] This process resulted in the following, significantly reduced type hierarchy below *word*:



Since only types duplicating information encoded elsewhere were eliminated, the simplification of the type hierarchy below *word* did not affect the coverage or the analyses provided by the grammar, which was verified using the 1348 item test suite provided with ERG. The complex type hierarchy of figure 2.11 thus appears to be irrelevant in terms of the structures licensed by the grammar. Furthermore, the simplification did not result in any negative side-effects with respect to computational issues, such as introducing disjunctions into the grammar or requiring complex antecedents for implicational constraints.

2.1.2.5 Summary

The purpose of this section was to contribute to a discussion of the criteria guiding the development of a well-motivated standard practice for HPSG-based grammar development. Focusing on the motivation for type distinctions introduced in a grammar, we applied the principle of Occam's razor to a part of the ERG type hierarchy and eliminated all type distinctions which are not motivated in terms of the sentences licensed by the grammar and the analyses assigned to them. This eliminated almost the entire hierarchy under *word* and thus resulted in a significant simplification of the grammar. In terms of outlook, we are continuing this experiment with the intention of eliminating all type distinctions which are not motivated in terms of the sentences licensed and the analyses assigned to them. Based on the experience with the *word* subtypes described in this section, we are confident that the resulting grammar will be significantly more transparent and compact than the original. We therefore propose such a conservative use of types as a model which HPSG-based grammar writers can follow to achieve a compact and transparent encoding.

[11] The experiment was carried out with a recoding of the ERG in the Trale system (Meurers et al., 2002), which includes the possibility of defining macros to abbreviate descriptions.

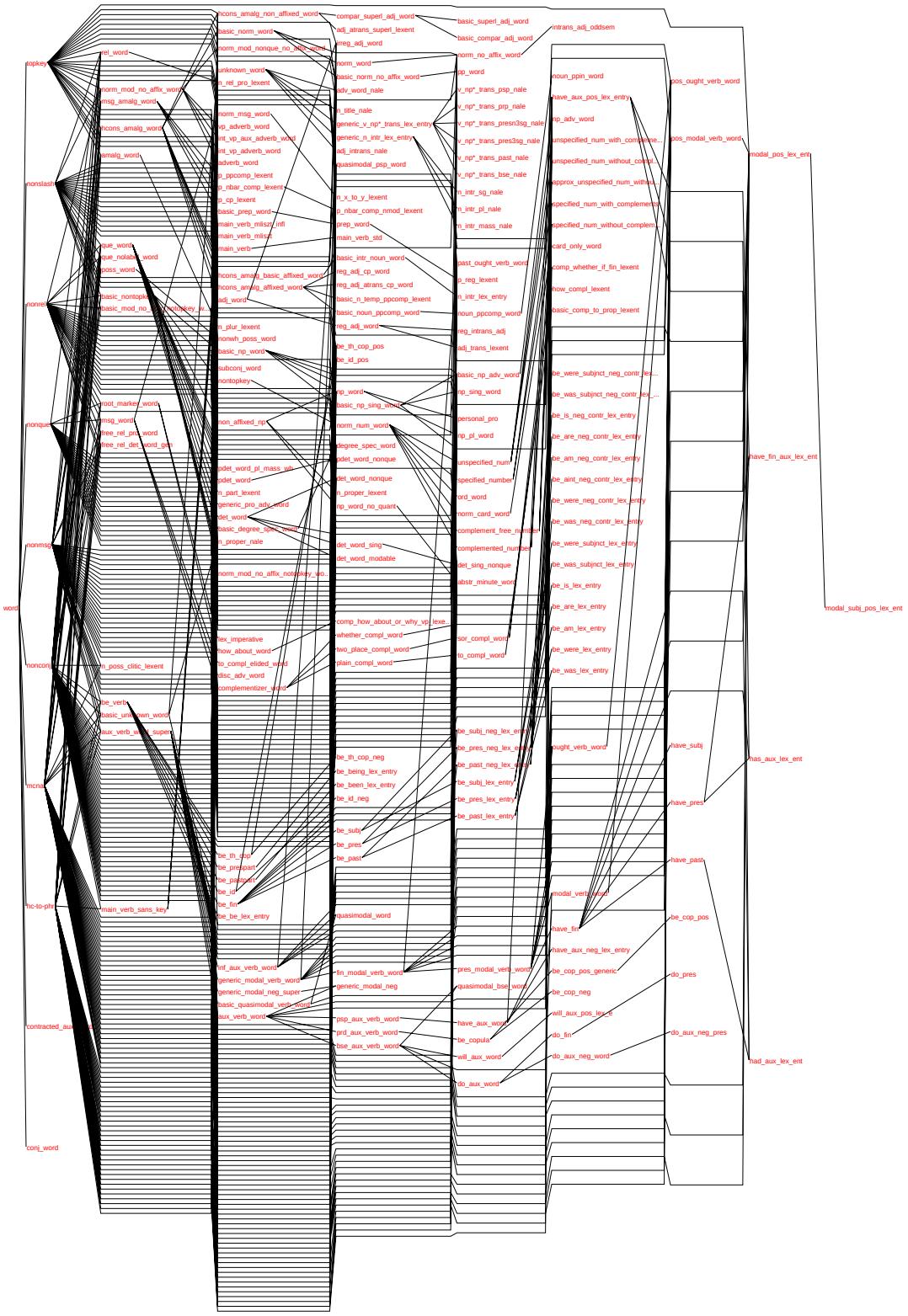


Figure 2.11: The hierarchy below *word* in the ERG

2.2 Notes on correspondences and differences

2.2.1 The signature

2.2.1.1 Basic types and appropriateness conditions

The two systems Trale and LKB differ fundamentally regarding the way how the signature is encoded. In Trale, the signature as the place where the data structure is declared, i.e., a type hierarchy and the appropriateness conditions are specified, and the theory are two separate parts of the grammar that is implemented. In LKB no such distinction between the signature and the theory is made. Types are introduced together with the appropriateness conditions and the constraints on them. In our Trale grammar, the type declarations of the ERG therefore need to be split up into that part that belongs into the signature and that part that belongs into the theory.

The signature in the Trale grammar mostly contains what in the ERG is specified as the fundamental types. These are the types that specify the basic linguistic data structures and their appropriateness condition which is in HPSG the architecture of signs, synsem objects, local and non-local objects etc. Furthermore, the signature contains all the types that are below *sort* in the ERG, i.e., all the types need for tense, aspect, case, vform, bool, etc.

In the LKB system, the grammar writer does not have to specify the greatest lower bound for all types. The system does insert these so called *glbtypes* into the hierarchy automatically during compile time. The Trale system, however, requires the specification of all unique greatest lower bounds in the signature. The hierarchy below many types in our signature contains therefore more subtypes than the same one specified in the ERG. This is illustrated by the two hierarchies for the type *luk* (the three-valued sort for bool named after the Polish logician Jan Lukasiewicz) in figures 2.12 and 2.13. The hierarchy in figure 2.12 is the one

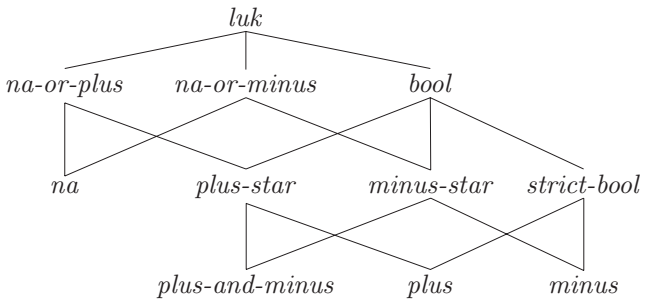


Figure 2.12: Type hierarchy under *luk* as specified in the ERG grammar

specified in the ERG where the types *na-or-plus* and *na-or-minus* do not have a unique greatest lower bound.

The hierarchy in figure 2.13 is the one specified in the Trale grammar where an additional type *na-or-plus-and-minus* has been introduced below *luk*. This ensures that *na-or-plus* and *na-or-minus* have a unique greatest lower bound, namely *a-or-plus-and-minus*.

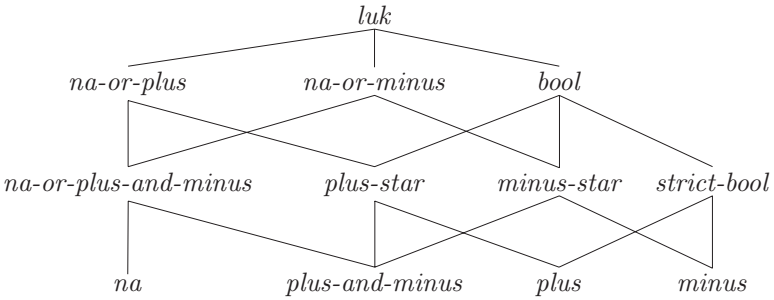


Figure 2.13: The type hierarchy under *luk* as specified in the Trale grammar

2.2.1.2 Lexical types

The type hierarchy below *word* as specified in the Trale grammar is shown in figure 2.14. The hierarchy below *word* specified in the ERG was already shown in figure 2.11 on page 20.

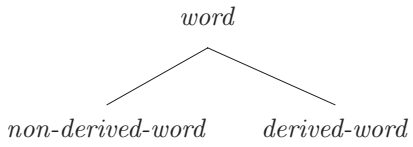


Figure 2.14: The hierarchy below *word*

It is significantly larger, since subtypes for all kinds of words (nouns, verbs, transitive verbs, finite verbs, ergative verbs, ...) are introduced. As already explained above, this is done because there are no macros in LKB. All these lexical types are used as abbreviations in the specifications of lexical entries. We therefore decided to encode these not as types in our grammar but as macros. An example for the encoding of such lexical types will be given in the section about lexical entries below.

2.2.1.3 Phrasal types

The major part of the type hierarchy below *phrase* in the MERGE is shown in figure 2.15.

In comparison, the hierarchy below phrase used in the ERG is shown in figure 2.16 on page 24. Many of these types below *phrase* are only introduced to abbreviate the specification of the mother in phrase structure rules. Similarly to the lexical types, we encoded such types as macros in our grammar. This will be discussed in more detail in section 2.2.2. The only distinctions that are necessary in the grammar are the ones between unary and binary phrases and headed and non-headed phrases. Every maximally specific type below *phrase* corresponds to a phrase structure rule. The hierarchy in the Trale grammar will therefore still get bigger the more rules we introduce.

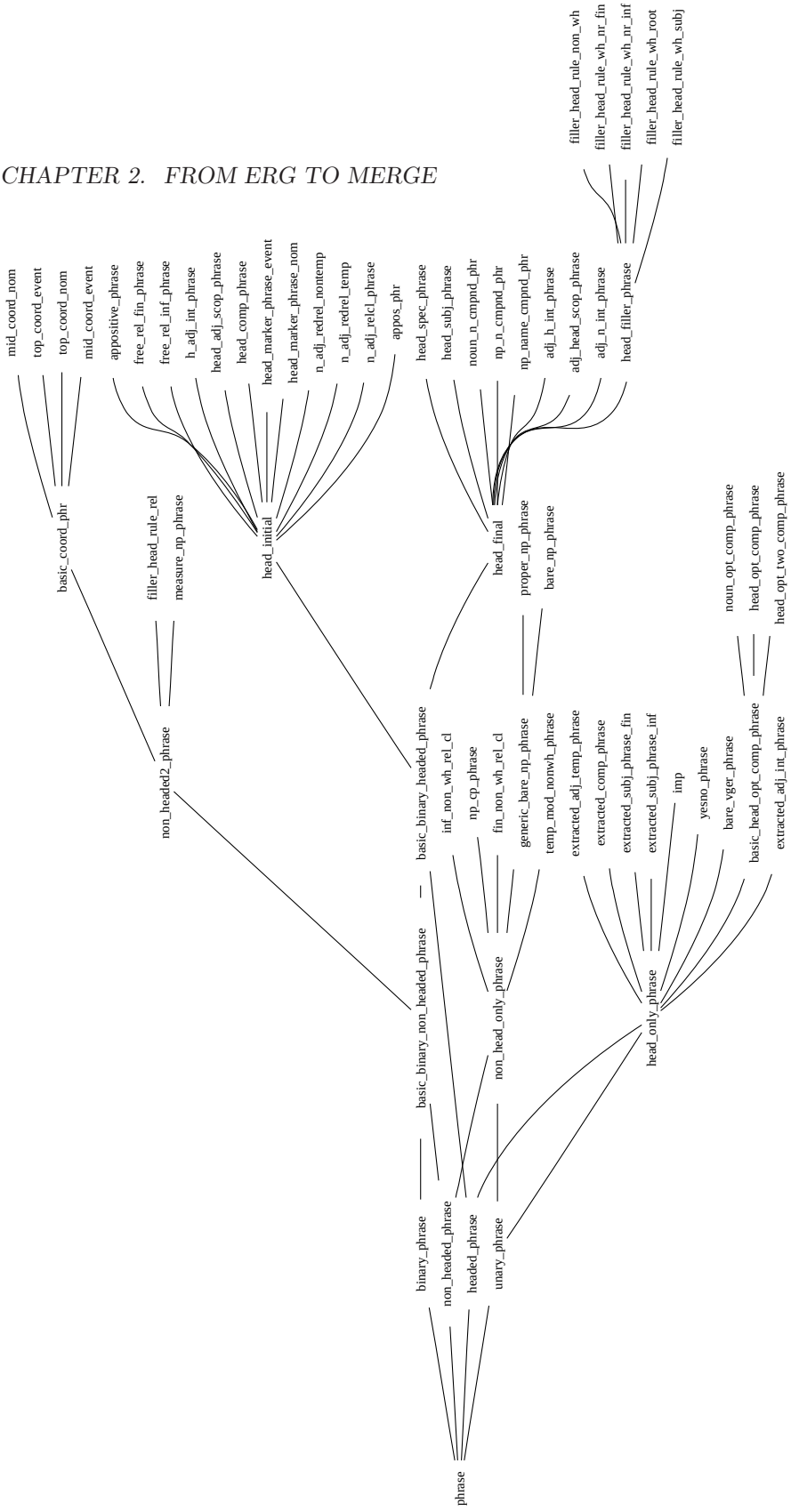


Figure 2.15: The hierarchy below *phrase*

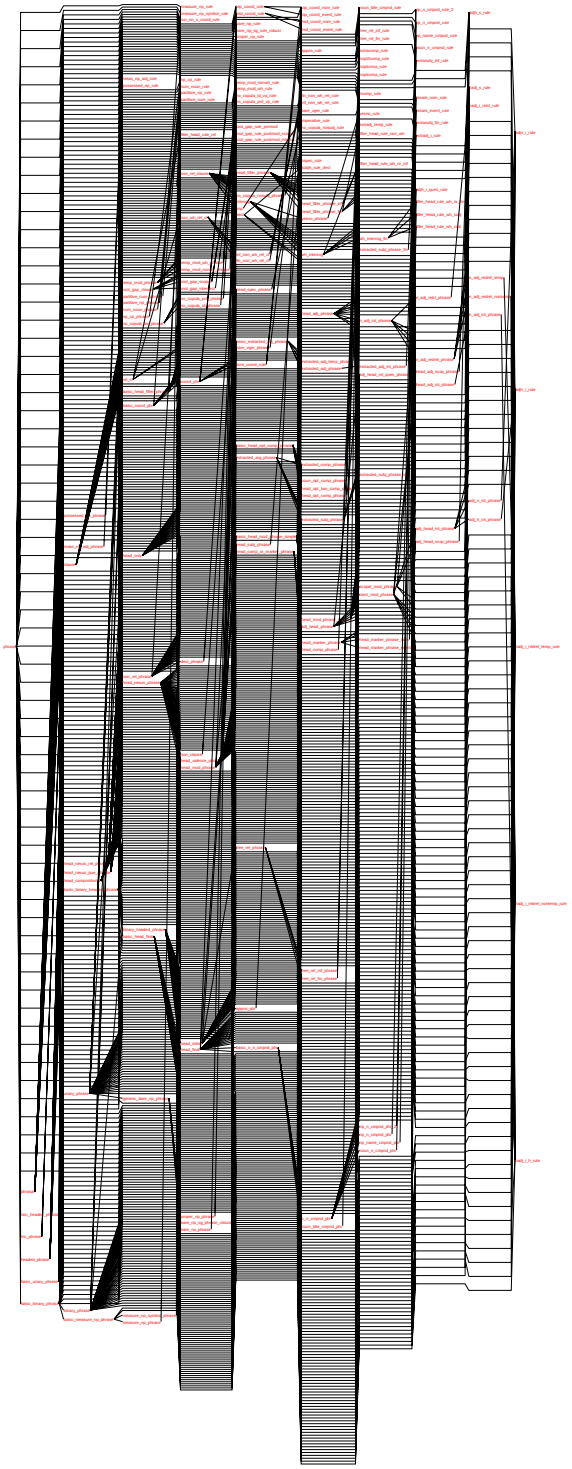


Figure 2.16: The hierarchy below *phrase* in the ERG

2.2.1.4 Lists

Another fundamental difference between LKB and Trale is the way how lists are encoded. The LKB system does not provide any kind of list operations. That means that the grammar writer has to encode the append of lists with the help of difference lists which have to be explicitly declared in the signature. The Trale system, however, does provide list operations like append. The only thing that is necessary is the declaration of a simple *list* type with HEAD and TAIL as appropriate features. Those type constraints in the ERG that only take care of the append of lists can therefore be encoded as much simpler constraints in the Trale grammar.

One example where these two different list encodings make a crucial difference is the slash amalgamation which ensures that each word collects the NON-LOCAL values of all its arguments onto its own NON-LOCAL values. In the ERG, four different types are introduced, *basic_one_arg, basic_two_arg, basic_three_arg, basic_four_arg*, for words with one argument, two arguments, etc. For each of these types a constraint is formulated which ensures that the NON-LOCAL values of the arguments (encoded as difference lists) are passed onto the *non-local* lists of the word itself. The constraint for words with two arguments is shown in figure 2.17.

```
basic_two_arg := lex_synsem &
  [ LOCAL.ARG-S < [ NON-LOCAL [ SLASH [ LIST #smiddle,
                                        LAST #slast ],
                                REL [ LIST #rmiddle,
                                      LAST #rlast ],
                                QUE [ LIST #qmiddle,
                                      LAST #qlast ] ],
                    LOCAL.CONT.INDEX individual ],
                  [ NON-LOCAL [ SLASH [ LIST #sfirst,
                                        LAST #smiddle ],
                                REL [ LIST #rfirst,
                                      LAST #rmiddle ],
                                QUE [ LIST #qfirst,
                                      LAST #qmiddle ] ],
                    LOCAL.CONT.INDEX individual ] >,
    NON-LOCAL [ SLASH [ LIST #sfirst,
                        LAST #slast ],
                REL [ LIST #rfirst,
                      LAST #rlast ],
                QUE [ LIST #qfirst,
                      LAST #qlast ] ] ].
```

Figure 2.17: A type constraint used for slash amalgamation in the ERG

In our Trale grammar, on the other hand, only one principle for all words, regardless of how many arguments they have, is necessary. This principle is shown in figure (2.18).

The relational goals `collect_slashes, collect_rel,collect_que` ensure that the NON-

```
(non_derived_word,
 synsem:lex:plus)
*>
(synsem:(local:arg_s:X,
         nonlocal:(slash:Slash,
                   rel:Rel,
                   que:Que)))
 goal
    (collect_slashes(X,Slash),
     collect_rel(X,Rel),
     collect_que(X,Que)).
```

Figure 2.18: The slash-amalgamation principle in the Trale grammar

LOCAL values are collected from the elements on the ARG-S list. The definition of the relation *collect_slashes* is shown in figure 2.19.

```
% collect_slashes(+ArgSt,-Slash)
collect_slashes(ArgSt,Slash) if
  when( ArgSt = (e_list;ne_list),
        undelayed_collect_slashes(ArgSt,Slash)).

%undelayed_collect_slashes(+list-of-synsem,-list-of-local) if true.
undelayed_collect_slashes([],[]) if true.

undelayed_collect_slashes([(nonlocal:slash:X)|Rest_y], All_l) if
       collect_slashes(Rest_y,Rest_l),
       append(X,Rest_l,All_l).
```

Figure 2.19: Definition of `collect_slashes` used for SLASH amalgamation

Similar definitions are implemented for `collect_rel` and `collect_que`.

2.2.2 The theory

2.2.2.1 The lexicon

The lexicon in the ERG is organized in terms of a huge lexical hierarchy below the type *word*. Complex descriptions are encoded in the lexical types which are used in the formulation of lexical entries.

Figure 2.20 shows the Ergo lexical entry for the unergative verb *sleep*.

The declaration of the lexical types used in the description of this lexical entry are shown in figure 2.21.

The type *v_unerg_le* used in the lexical entry of *sleep* is a subtype of *main_verb*. This lexical type is a subtype of *main_verb_sans_key, topkey, mcna* and *nonmsg*. The latter three

```
sleep_v1 := v_unerg_le &
  [ STEM < "sleep" >,
    SYNSEM.LOCAL.KEYS.KEY _sleep_rel ].
```

Figure 2.20: Lexical entry in the ERG grammar

```
nonconj := word &
  [ SYNSEM.LOCAL.CONJ cnil ].

mcna := word &
  [ SYNSEM.LOCAL.CAT.MC na ].

nonmsg := word &
  [ SYNSEM.LOCAL.KEYS.MESSAGE 0-dlist ].

topkey := word &
  [ SYNSEM.LOCAL [ KEYS.KEY #key,
                   CONT.--TOPKEY #key ] ].

hc-to-phr := word &
  [ SYNSEM.LOCAL.CAT.HC-LEX -* ].

main_verb_sans_key := nonconj & hc-to-phr &
  [ SYNSEM.LOCAL [ CAT.HEAD verb* & [ AUX -*,
                                      INV -* ],
                   CONT.LISZT.LIST < v_event_rel, ... >,
                   KEYS [ KEY.LABEL *cons*,
                          ALTKEY role_rel ] ] ].

main_verb := main_verb_sans_key & topkey & mcna & nonmsg &
  [ INFLECTED -,
    SYNSEM.LOCAL [ KEYS.KEY #key,
                   CONT.LISZT <! #key !> ] ].

v_unerg_le := main_verb &
  [ SYNSEM unerg_verb ].
```

Figure 2.21: Lexical type declarations in the ERG

types are then subtypes of word.

In order to reduce the complexity of the signature in our Trale grammar we decided that the same kind of lexical hierarchy can also be encoded as a macro hierarchy. Thus, the lexical entry for *sleep* in the Trale grammar calls the macro `@v_unerg_le` as shown in figure 2.22.

The the macro hierarchy above `@v_unerg_le` corresponding to the lexical hierarchy in the

```
sleep  ~~> @v_unerg_le.
```

Figure 2.22: Lexical entry in the Trale grammar

ERG is shown in figure 2.23.

```
nonconj macro (non_derived_word, synsem:local:conj:cnil).

mcna macro (non_derived_word, synsem:local:cat:mc:na).

nonmsg macro (non_derived_word,
              synsem:local:keys:message:e_list).

topkey macro (non_derived_word,
               synsem:local:( keys:key:Key,
                              cont:topkey:Key) ).

hc_to_phr macro (non_derived_word, synsem:local:cat:hc_lex:minus).

main_verb_sans_key macro (@nonconj, @hc_to_phr,
                          synsem:local:(cat:head:(verb,
                                                  aux:minus,
                                                  inv:minus),
                                                cont:psoa )).

main_verb macro (@main_verb_sans_key, @mcna, @nonmsg, @topkey,
                 inflected:minus).

v_unerg_le macro  (@main_verb,
                   synsem: @unerg_verb).
```

Figure 2.23: Lexical macros used in the Trale grammar

2.2.2.2 The phrase structure rules

Both systems, LKB and Trale, are parsing systems, which means that the grammar writer needs to specify a set of phrase structure rules for those strings that the system is supposed to recognize.

Figure 2.24 shows the specification of such a phrase structure rule, in this case the head-complement rule, in the Ergo grammar.

Surprisingly, this rule does not show any explicit specification of a mother or daughters. The only things specified are the name of the rule (hcomp), a type *hcomp_rule* and a feature declaration. The actual specification of the mother and the daughters is "hidden" in the supertypes of *hcomp_rule*. A subpart of this type hierarchy is shown in figure 2.25.

```
hcomp := hcomp_rule &
  [ RULE-NAME 'hcomp ].
```

Figure 2.24: A phrase structure rule in the ERG

```
head_initial := binary_headed_phrase &
  [ HEAD-DTR #head,
    NON-HEAD-DTR #non-head & [ SYNSEM.LOCAL.CONJ cnil_or_numconj ],
    ARGS < #head, #non-head > ].

head_comp_or_marker_phrase := head_valence_phrase & head_compositional &
                              head_initial &
  [ SYNSEM canonical_synsem &
           [ LOCAL [ CAT [ MC #mc,
                           VAL [ SUBJ #subj,
                                 COMPS #comps,
                                 SPR #spr ] ],
                     KEYS [ ALTKEY #altkey,
                            MESSAGE #hmsg ] ] ],
    HEAD-DTR.SYNSEM.LOCAL [ CAT [ MC #mc,
                                  VAL [ SUBJ #subj,
                                        COMPS < #synsem . #comps >,
                                        SPR #spr ] ],
                            KEYS [ ALTKEY #altkey,
                                   MESSAGE #hmsg ] ],
    NON-HEAD-DTR.SYNSEM #synsem & canonical_synsem,
    C-CONT [ LISZT <! !>,
             H-CONS <! !> ] ].

head_comp_phrase := head_comp_or_marker_phrase &
  [ SYNSEM [ LOCAL [ CAT.POSTHEAD #ph,
                     CONJ cnil ],
             LEX #lex ],
    HEAD-DTR [ SYNSEM.LOCAL.CAT [ POSTHEAD #ph,
                                  HC-LEX #lex ] ] ].

hcomp_rule := binary_rule_left_to_right & head_comp_phrase.
```

Figure 2.25: Phrasal types in the ERG

The actual daughters specification needed for the parser is the ARGS list specified in the type declaration of *head_initial*. The mother of a phrase that is licensed by the hcomp rule is thus of type *hcomp_rule* which is a subtype of *head_comp_phrase* and *bibary_rule_left_to_right*. *head_comp_phrase* is then a subtype of *head_comp_or_marker_phrase* which itself is a subtype

of *head_initial*.

The phrase structure rules in the Trale grammar differ from this encoding in two ways. Figure 2.26 shows the encoding of the head-complement rule in the Trale grammar.

```
hcomp ## @head_comp_phrase(HeadDtr,NonheadDtr)
  ===>
cat> HeadDtr,
cat> NonheadDtr.
```

Figure 2.26: A phrase structure rule in the Trale grammar

The first difference with respect to the head-complement rule in the ERG is that in Trale the left-hand side and the right-hand side have to be explicitly specified in the rule. We therefore introduced the macro `@head_comp_phrase(HeadDtr, NonheadDtr)` with two arguments corresponding to the two daughters of the rule. As can be seen in figure 2.27, the definition of this macro nearly entirely corresponds to the type declaration of *head_comp_phrase* in the ERG.

```
head_comp_or_marker_phrase(NonheadDtr) macro (@head_valence_phrase,
                                              head_initial,
    synsem:( canonical_synsem ,
             local: (cat:( mc:Mc,
                           val:( subj:Subj,
                                 comps:Comps,
                                 spr:Spr ) ),
                     keys:( altkey:Altkey,
                            message:Hmsg ) ) ),
    head_dtr:synsem:local:( cat:( mc:Mc,
                                  val:( subj:Subj,
                                        comps:[Synsem|Comps],
                                        spr:Spr ) ),
                            keys:( altkey:Altkey,
                                   message:Hmsg ) ),
    non_head_dtr:(NonheadDtr,synsem:(Synsem,canonical_synsem))).

head_comp_phrase(HeadDtr,NonheadDtr) macro (head_comp_phrase,
                                            @head_comp_or_marker_phrase(NonheadDtr),
    synsem: (local:( cat:posthead:Ph,
                     conj:cnil),
             lex:Lex),
    head_dtr: (HeadDtr, synsem:local:cat:( posthead:Ph,
                                           hc_lex:Lex )) ).
```

Figure 2.27: Phrasal macros in the Trale grammar

This leads to the second difference in the encoding of phrase structure. We decided that it is sufficient to characterize phrases according to their headedness and the kinds of daughters they have (the resulting type hierarchy below *phrase* was shown in figure 2.15. We did not encode the additional classification of phrases with respect to the phrase structure rules which license them, as it is done in the ERG with the help of the type *rule* and its subtypes, two of which are *hcomp_rule* and *binary_rule_left_to_right* shown in figure 2.25. And, similar to the lexical types, we decided that many of the phrasal types in the ERG are simply used as abbreviations in the specifications of phrase structure rules and we therefore encoded these as macros, as for example `@head_comp_phrase(HeadDtr,NonheadDtr)` and `@head_comp_or_marker_phrase(NonheadDtr)` shown in figure 2.27.

2.2.2.3 The lexical rules

Most of the lexical entries specified in the lexicon of the Ergo grammar are not real words, but so-called lexemes, i.e., lexical items without any inflectional endings and without any agreement information. All these lexemes are input to lexical rules and only the outputs of the lexical rules are the actual words that occur in sentences. All verbs, nouns and adjectives are specified as such lexemes in the lexicon, categories like determiners and prepositions are specified in the same way in the lexicon but are not transformed via lexical rules. All lexemes in the lexicon are subtypes of *word*. All words that are the output of a lexical rule are a subtype of *lex_rule_supermost*. The type hierarchy for *word* in the ERG is shown in figure 2.28. An example for a lexical rule in the ERG is shown in figure 2.29.

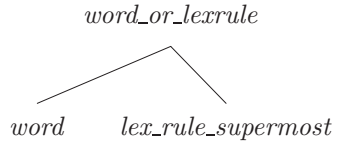


Figure 2.28: Hierarchy for words in the ERG

```
third_sg_fin_verb_infl_rule :=
%suffix (!s !ss) (!ss !ssses) (ss sses) (!ty !ties) (ch ches) (sh shes) (x xes) (z zes)
lex_rule_infl_affixed &
[ NEEDS-AFFIX +,
  SYNSEM.LOCAL third_sg_fin_verb ].
```

Figure 2.29: Inflectional lexical rule in the ERG

Similar to the phrase structure rules described in the previous section, the relation between the input and the output does not have to be explicitly specified within the specification of the rule. In the case of the inflectional lexical rule shown in figure 2.29, this is done in the declaration of the type *lex_rule_infl_affixed*. This type declaration is shown in figure 2.30.

The inflectional lexical rule in the Trale grammar which is shown in figure 2.31 differs from the lexical rule in the ERG in two ways.

```
lex_rule_infl_affixed := lex_rule_compos &
  [ INFLECTED +,
    KEY-ARG #keyarg,
    SYNSEM #synsem,
    ARGS < #dtr >,
    ROOT #root,
    POSSCL -,
    DTR #dtr & [ INFLECTED -,
                 KEY-ARG #keyarg,
                 SYNSEM #synsem,
                 ROOT #root ],
    C-CONT.LISZT <! !> ].
```

Figure 2.30: Declaration of *lex_rule_infl_affixed* in the ERG

```
third_sg_fin_verb_infl_rule ##
    Dtr
**>
   ( lr:lex_rule_infl_affixed,
    needs_affix:plus,
    synsem:local: @third_sg_fin_verb,
    @lex_rule_infl_affixed_macro(Dtr))
morphs
     X      becomes  Y when irregpres(X,Y),
    (X,S,s) becomes (X,S,ssess) when letter_set_s(S),
    (X,ss)  becomes (X,sses),
    (X,T,y) becomes (X,T,ies) when letter_set_t(T),
    (X,ch)  becomes (X,ches),
    (X,sh)  becomes (X,shes),
    (X,x)   becomes (X,xes),
    (X,z)   becomes (X,zes),
    (X,C)   becomes (X,C,s) when letter_set_s(C).
```

Figure 2.31: A lexical rule in the Trale grammar

First of all, the relation between the input and the output of the rule has to be explicitly specified within the rule with the help of the macro `@lex_rule_infl_affixed_macro(Dtr)`. Secondly, the output of lexical rule in the Trale grammar is of type *derived_word*. The information with which lexical rule a word was derived is encoded with the help of the feature LR and is not directly reflected by the type of the word itself as it is done in the ergo grammar. The encoding of the inflection, in this case the generation of the correct third person singular form of the verb, with the help of "morphs" is a direct reflection of the way the inflection is encoded in the ERG. Classes of letters are determined after which a certain ending has to be added to the word to generate the correct form.

2.2.2.4 The principles

As already mention in section 2.2.1, in grammars written for the LKB system there is no distinction made between the signature and the theory, with the effect that types are introduced together with the constraints on these. In our Trale grammar, however, we need to take these declaration apart into the declaration of the type in the signature and the constraint that is introduced for this type.

The Head Feature Principle is a good example to illustrate these two different encodings. Figure 2.32 shows the declaration of type *headed-phrase* as a subtype of *phrase* together with the constraint on it, the sharing of certain information between the head daughter and the phrase itself, as specified in the Ergo grammar.

```
headed_phrase := phrase &
  [ ROOT -,
    SYNSEM.LOCAL [ CAT [ HEAD head & #head,
                         HC-LEX #hclex ],
                   AGR #agr,
                   CONJ #conj,
                   KEYS.KEY #key ],
    HEAD-DTR.SYNSEM.LOCAL local &
                 [ CAT [ HEAD #head,
                         HC-LEX #hclex ],
                   AGR #agr,
                   CONJ #conj,
                   KEYS.KEY #key ] ].
```

Figure 2.32: The declaration of *headed-phrase* in the ERG

The declaration of the type *headed-phrase* as part of the type hierarchy below *phrase* in the Trale grammar was already shown in figure 2.15. The figure 2.33 shows the constraint formulated for *headed-phrase* in the MERGE.

In a similar way, most of the type declarations of the ERG have to be taken apart and re-encoded in the MERGE.

```
headed_phrase *>
   (root:minus,
    synsem:local:( cat:( head:Head,
                         hc_lex:Hclex),
                   agr:Agr,
                  conj:Conj,
                  keys:key:Key ),
    head_dtr:synsem:local:(cat: (head:Head,
                                 hc_lex:Hclex ),
                           agr:Agr,
                          conj:Conj,
                          keys:key:Key ) ).
```

Figure 2.33: Constraint on *headed-phrase* in the Trale grammar

Chapter 3

Coverage of the grammar

As stated in the previous section, the general aim of this project is to implement a large-scale grammar for English in Trale that has the same coverage as the ERG (Flickinger et al., 2000, p. 258). The following listing serves to illustrate the coverage of English constructions in MERGE.

3.1 Basic declarative sentences

- intransitive verbs:

  (2) Abrams works.

- transitive verbs:

  (3) Abrams hired Browne.

- ditransitive verbs:

  (4) a. Abrams showed the office to Browne.
      b. Abrams showed Browne the office.

- sentential complements:

  (5) Abrams bet Browne five dollars that Chiang hired Devito.

- predicatives:

  (6) a. Abrams became competent.
      b. Abrams became a manager.

- prepositional complements:

(7) a. Abrams works for Browne.
b. * Abrams works of Browne.
c. Abrams approves of Browne.
d. * Abrams approves for Browne.

- impersonal constructions

(8) a. It is time for an interview.
b. It is true that Abrams hired Browne.
c. There is a bookcase in the office.
d. There are programmers.
e. There are programmers interviewing Devito.
f. There are programmers older than Devito.
g. There stands in the office a bookcase.
h. There is Abrams, Browne, and Chiang.

- auxiliaries/modal verbs

(9) a. Abrams may hire Browne.
b. Abrams can hire Browne.
c. Abrams did hire Browne.
d. Abrams has hired Browne.
e. Abrams had better hire Browne.
f. Abrams better hire Browne.
g. Abrams could have hired Browne.
h. Abrams could be hiring Browne.
i. Abrams could have been hiring Browne.

- control verbs

(10) a. Abrams promised Browne to hire Chiang.
b. He promised us to evaluate himself.
c. * Abrams promised there to be a bookcase in the office.
d. Abrams promised Browne to be interviewed by Chiang.
e. Abrams urged Browne to hire Chiang.
f. Browne was urged to hire Chiang.
g. Abrams urged Browne to be interviewed.
h. Abrams was urged to be interviewed.
i. Abrams appealed to Browne to hire Chiang.
j. Abrams appealed to Browne to be interviewed.

- tough constructions

  (11) a. It was hard to show an office to Chiang.
  b. Chiang was hard to show an office to.
  c. Offices are hard to show to Chiang.
  d. Chiang was hard for Abrams to show an office to.
  e. Offices are hard for Abrams to show to Chiang.
  f. Chiang was hard to show an office.

- passive

  (12) a. Abrams was hired (by Browne).
  b. An office was shown to Abrams (by Chiang).
  c. Chiang was shown an office by Abrams.
  d. Abrams was urged to hire Browne by Chiang.
  e. Abrams was urged by Chiang to hire Browne.
  f. Abrams was known (by Chiang) to be interviewing Browne.
  g. Abrams was made to interview Browne.
  h. Abrams got hired (by Browne).
  i. Abrams had Browne hired (by Chiang).

- inversion

  (13) a. That office, the consultants work in.
  b. To Browne, Abrams showed an office.
  c. Competent, Abrams is.
  d. Hire Browne, Abrams did.
  e. Not one programmer did Abrams hire.
  f. At no time did Abrams hire a programmer.
  g. No more competent was Abrams.
  h. A trustworthy employee is Abrams.
  i. In the office is the bookcase.
  j. Competent is the manager who hired Abrams.
  k. Standing in the office is the bookcase.
  l. Never does Abrams work with Browne.
  m. In the office is a good location for the bookcase.

- unbounded dependencies

  (14) a. Which manager did Abrams know was interviewing programmers.
  b. Which managers did Abrams know were interviewing programmers.

c. Which department is Abrams the manager of?
d. Whose department does Abrams work in?

- comparatives

(15) a. Chiang is (two days) older than Browne.
b. Abrams is (two days) older than thirty years.
c. Abrams is more competent than Browne.
d. Abrams is as competent as Browne.
e. Abrams is the oldest manager.
f. Abrams is the most competent manager.
g. Abrams manages more programmers than Browne manages.
h. Abrams manages as many programmers as Browne manages.
i. Abrams interviewed more programmers than were hired.

- conjunction

(16) a. Chiang is a manager and Devito is a programmer.
b. Chiang and Devito work.
c. Browne, Chiang, and Devito work.
d. Browne and Chiang and Devito work.
e. Both Chiang and Devito work.
f. Either Chiang or Devito works.
g. Neither Chiang nor Devito works.
h. Chiang hired Devito and manages Browne.

- ellipsis

(17) a. Abrams was interviewing programmers, and Browne was, too.
b. Abrams was interviewing programmers, and so was Browne.
c. Abrams was interviewing programmers, or Browne was.
d. Abrams wasn't interviewing programmers, nor was Browne.
e. Abrams wasn't interviewing programmers, and neither was Browne.
f. Browne wasn't interviewed by Devito, but he was by Chiang.
g. Abrams doesn't consult for Browne, but he does for Chiang.

- binding

(18) a. Devito knew that he had hired a programmer.
b. The manager evaluated her staff.
c. Her manager interviewed Browne.
d. The person who manages him knows that Browne is competent.

e. Devito interviewed programmers. He hired Browne.
f. Browne evaluated himself.
g. The managers evaluated each other.

- adverb placement

(19) a. Evidently Chiang works.
b. Chiang evidently works.
c. Chiang works, evidently
d. Chiang has evidently been leaving.
e. Evidently Chiang showed Devito an office.
f. Chiang evidently showed Devito an office.
g. * Chiang showed evidently Devito an office.
h. * Chiang showed Devito evidently an office.

- quantifier

(20) a. Every manager who interviewed a programmer hired him.
b. If a programmer was interviewed by every manager, he was hired.
c. Every engineer who has a bookcase arrived.
d. The engineer who has a bookcase arrived.
e. Every manager interviewed one of the programmers.
f. Two programmers were interviewed by every manager.

- negation

(21) a. Devito has not hired Abrams.
b. Devito could not hire Abrams.
c. Devito could not have hired Abrams.
d. Devito could not be hiring Abrams.
e. Devito did not hire Abrams.
f. * Devito hired not Abrams
g. Devito is not a manager.

3.2 Interrogative sentences

- direct questions

(22) a. She works for whom?
b. She works for who?
c. She manages whom?

d. She manages who?
e. She showed whom an office?
f. She showed who an office?
g. Whom does she work for?
h. Who does she work for?
i. For whom does she work?
j. * For who does she work?
k. Who hired Browne?

- embedded interrogatives

(23) a. Abrams does not know who hired Browne.
b. Abrams does not know which managers interviewed Browne.
c. Abrams does not know who Browne hired.
d. Abrams does not know who showed Browne an office.
e. Abrams does not know who(m) Browne was hired by.
f. Abrams does not know by whom Browne was hired.
g. Abrams does not know which programmers Browne hired.
h. Abrams does not know where Browne works.
i. Abrams does not know when Browne was hired.
j. Abrams does not know how many consultants Browne has hired.
k. Abrams does not know how competent Browne is.

- selection of interrogatives

(24) a. Kim knows whether Sandy sleeps.
b. Kim knows whether to sleep.
c. * Kim knows whether Sandy to sleep.
d. Kim knows when to sleep.
e. Kim knows what to think Sandy likes.
f. Kim knows who to think Sandy was hired by.
g. Kim knows if Sandy sleeps.
h. Kim wonders whether Sandy sleeps.
i. Kim wonders whether to sleep.

- tag questions

(25) a. It is raining, isn't it?
b. There is a meeting, isn't there?
c. * It is raining, isn't there?
d. Sara is sleeping, isn't she?

e. * Sara is sleeping, isn't there?
f. Sara will sleep, won't she?
g. * Sara will sleep, can't she?
h. Sara sleeps, doesn't she?
i. Everyone slept, didn't they?

3.3 Imperative sentences

(26) a. Hire a programmer!
b. Be trustworthy!
c. * Are trustworthy!
d. Do not hire a programmer!
e. Don't hire a programmer!
f. You hire a programmer!
g. You be trustworthy!
h. Don't you hire a programmer!
i. Don't you be trustworthy!
j. Everyone hire a programmer!
k. Everyone be trustworthy!
l. Don't anyone hire a programmer!
m. Don't anyone be trustworthy!

3.4 Noun phrases

3.4.1 Pronouns

- Personal pronouns: Case

(27) a. He hired her.
b. * She hired he.

- Coordination

(28) a. She and I interviewed Abrams.
b. I and she interviewed Abrams.
c. Me and her interviewed Abrams.
d. Her and me interviewed Abrams.
e. * Her and I interviewed Abrams.
f. * She and me interviewed Abrams.

- Reflexives

(29) The woman evaluated herself.

- Generic pronoun

(30) Anyone with an office works.

3.4.2 Head-Specifier constructions

(31) a. The programmer was hired.
b. This programmer was hired.
c. These programmers were hired.
d. That programmer was hired.
e. Those programmers were hired.
f. Some programmer was hired.
g. Some programmers were hired.
h. * Any programmer was hired.
i. * Any programmers were hired.
j. Most programmers were hired.
k. Few programmers were hired.
l. No programmers were hired.
m. Many programmers were hired.
n. A dozen programmers were hired.
o. One programmer was hired.
p. More managers were interviewed.
q. Only the more competent programmers were interviewed.
r. Every manager who interviewed a programmer hired him.
s. Not one programmer did Abrams hire.

3.4.3 Modification

- Premodifiers

(32) a. Abrams sees Browne as a competent manager.
b. Abrams bet Browne five dollars that Chiang hired Devito.

- Postmodifiers
  - Prepositional phrases

(33) a. It is time for an interview.
b. Abrams has an office with a bookcase.

  - Relative clauses

(34) a. Abrams has an office that Browne showed Chiang.
b. Abrams has an office which Browne showed Chiang.
c. Abrams has an office Browne showed Chiang.
d. Abrams hired a woman that Browne interviewed.
e. Abrams hired a woman Browne approved of.

– Verb phrases

(35) a. Abrams hired a woman interviewed by Chiang.
b. Abrams hired a woman working for Chiang.

3.4.4 Other kinds of noun phrases

- Partitive constructions

(36) a. None of the consultants work for Abrams.
b. Five of the seven consultants work for Abrams.
c. Any five of the seven consultants can work for Abrams.
d. Most of the staff is competent.
e. Most of the program works.
f. Almost all of the program works.
g. Five teams of programmers were hired.
h. Half of the programmers were interviewed by Abrams.
i. Half the programmers were interviewed by Abrams.
j. Fewer than half the programmers were hired. (NP)
k. Fewer than half of the programmers were hired. (NP)
l. An number of programmers were interviewed. (NP)
m. An number of programmers was interviewed.

- Head-complement NPs

(37) a. All but seven programmers were hired.

- Titles

(38) a. Browne is the manager.
b. Lee Browne is the manager.
c. Mr. Browne is the manager.
d. Mr. Lee Browne is the manager.
e. A Mr. Lee Browne is the manager.
f. A Lee Browne is the manager.

g. Lee Browne, Esq is the manager.

- Dates

(39) a. Browne was hired on 1/3/84.
b. Browne was hired on the 1st of January, 1984.
c. Browne was hired on January 1st, 1984.

Chapter 4

Description of the components of the grammar

In this section all the components of the MERGE are described in detail. These include: the signature, the phrase structure rules, classes of lexical entries, the lexical rules and the principles (implicational constraints). We will give a brief description of every single instantiation of these components that can be found in the MERGE. How these different components of the grammar finally interact in order to license the different phenomena covered by the MERGE grammar is described in section 5.

4.1 The Signature

Although we reduced to the size of the MERGE signature compared to the size of the original ERG signature significantly it still contains more than a thousand types. Describing each of these types and its appropriateness conditions would probably be far beyond the scope of this grammar documentation. We will therefore here only discuss those parts of the signature which are important in order to understand the description of the linguistic theories in the following sections.

4.1.1 Signs

The basic architecture of signs in the MERGE is very similar to the one generally assumed within HPSG theories following Pollard and Sag (1994). This basic architecture is shown in figure 4.1.

In addition to the known feature PHON and SYNSEM, signs in the MERGE have the appropriate features KEY_ARG, INFLECTED, ROOT, ROBUST, POSSCL, and C_CONT. The feature KEY_ARG seems to be used to indicate which element on the daughters list of a phrase is the head, namely the one which is [KEY_ARG *plus*]. The feature INFLECTED indicates whether a sign is the output of an inflectional lexical rule. The feature ROOT seems to be used in the LKB grammar to mark which signs the parser should provide as acceptable parses, i.e., it marks the start symbols of the grammar (we can probably get rid of this feature). The

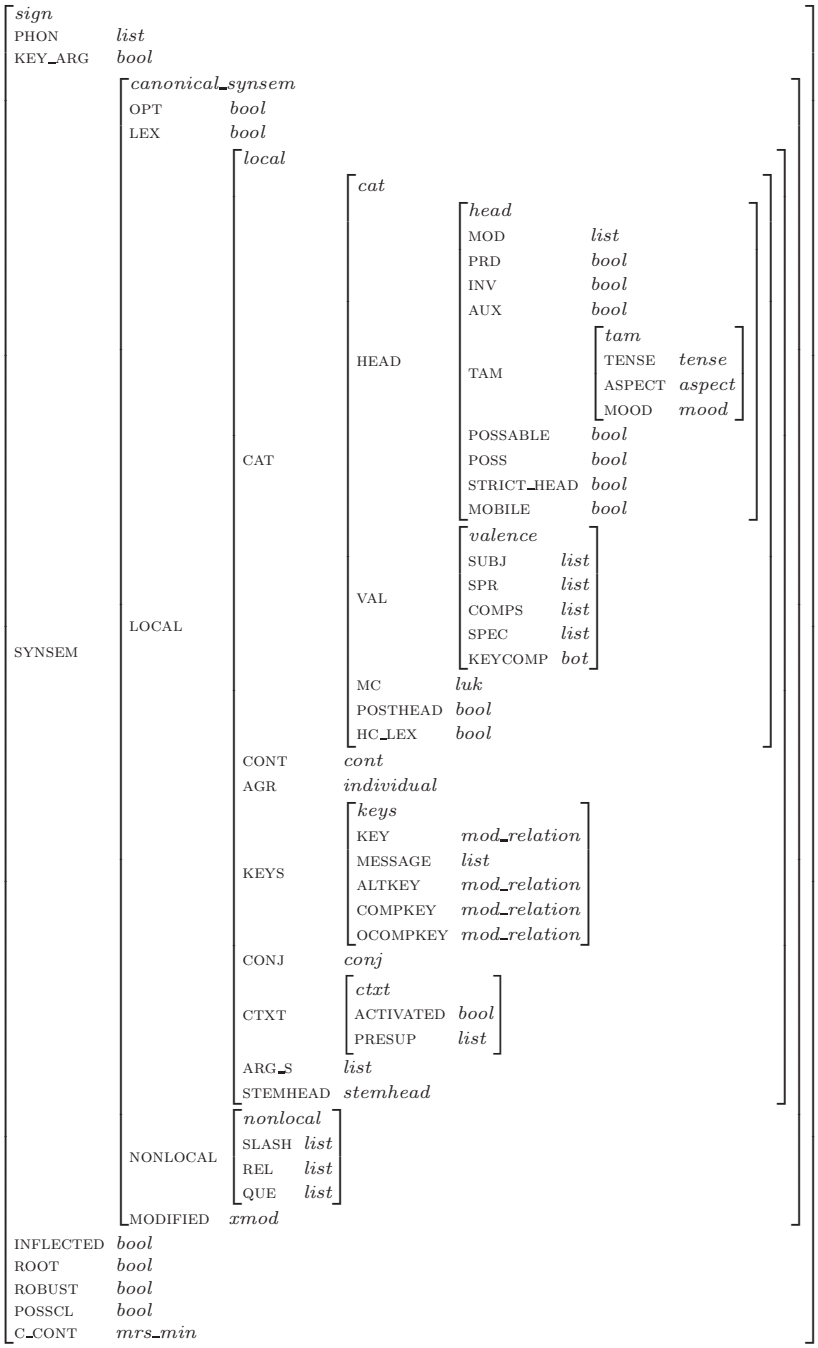


Figure 4.1: The basic architecture of signs in the MERGE

feature ROBUST is used to enable ungrammatical phrases such as missing determiners for singular nouns in the LKB grammar (this feature is currently of no use for the MERGE). The feature POSSCL marks whether sign contains a possessive clitic. The feature C_CONT specifies the "construction content" of a particular sign. For phrases, it is always structure shared in a certain way with the phrase's CONTENT values, as the principle in figure 4.2 shows.

```
phrase *>
  ( synsem:( canonical_synsem,
             local:cont:( top:Hand,
                          index:Index,
                          e_index:Event)),
    c_cont:( top:Hand,
             index:Index,
             e_index:Event)).
```

Figure 4.2: The C_CONT principle

The additional appropriateness conditions for the two only subtypes of *sign*, *word* and *phrase*, are shown in figure 4.3.

$$\begin{bmatrix} word \\ \text{ALTS} \;\; alts\_min \end{bmatrix} \begin{bmatrix} phrase \\ \text{DTRS} \;\; list \end{bmatrix}$$

Figure 4.3: Appropriateness conditions for *word* and *phrase*

The feature ALTS appropriate for words allows lexical entries to block lexical rule application. The DTRS feature appropriate for phrases is necessary in the MERGE in connection with the grisu interface to enable the proper display of tree structures (we might be able to get rid of this at some point). The subtypes of *word* and *phrase* and their appropriateness conditions are discussed in sections 4.3 and 4.2.

4.1.2 Synsem objects

The appropriate features for *canonical_synsem*, the value of the feature SYNSEM, are OPT, LEX, LOCAL, NONLOCAL and MODIFIED. Besides the known features LOCAL, NONLOCAL and LEX, the feature OPT is used to indicate whether the synsem elements on a COMPS list are optional complements or not and the feature MODIFIED indicated whether a sign has been modified or not.

The type *synsem* has the subtypes shown in figure 4.4.

4.1.3 Local objects

Local objects have the appropriate features CAT, CONT, AGR, CTXT, ARG_S, KEYS, CONJ, and STEMHEAD, of which the first five are traditional HPSG features and the last three are ERG (and MERGE) specific features.

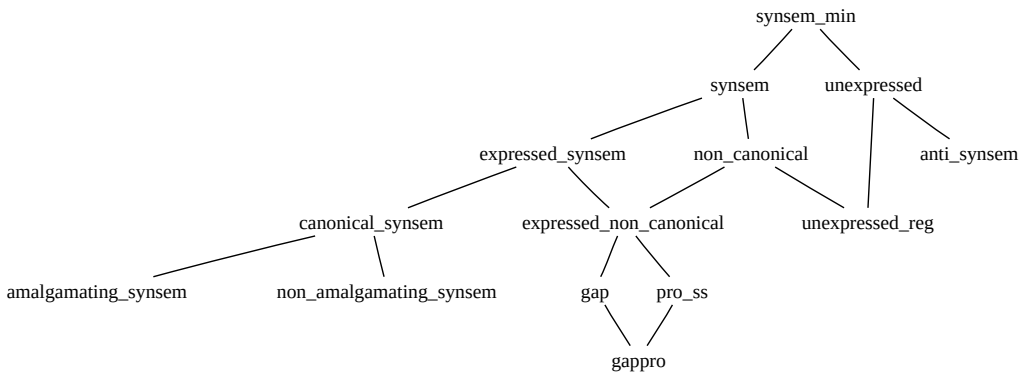


Figure 4.4: Subtypes of *synsem*

The feature AGR is introduced on LOCAL rather than on CAT (or on HEAD) since coordination schema unify the CAT value of the daughters with that of then mother, but need to be able to change AGR on the mother (to get plural agreement on verb when subject is a coordinated NP with "and" vs. "or").

The occurrence of ARG_S as a local feature is a bit unexpected and it also means, that it is present on both, words and phrases. Words always specify in the lexicon what the value of their ARG_S list is. For phrases, the constraint shown in figure 4.5 is necessary to ensure that the ARG_S feature also has some value that is more specific than *list*, namely *e_list*.

```
phrase *> (synsem:local:arg_s:[]).
```

Figure 4.5: The value of ARG_S on phrases

The feature KEY is used for semantic selection and the five features appropriate for *keys* objects have the following functions: KEY functions as a pointer to the main relation in LISZT (a *content* feature which will be further introduced in section 4.1.5). ALTKEY functions as a pointer to an alternate relation in the LISZT feature. COMPKEY is a pointer to the complement's main relation and OCOMPKEY a pointer to the oblique complement's main relation. MESSAGE specifies the message type for propositions.

The value of the feature CONJ has two subtypes, *cnil* and *strict-conj* (i.e., non-cnil). *cnil* is always the value of non-coordinated structures, in a coordinated structure the left and right conjunct receive the appropriate subtype of *strict-conj* as their CONJ value. For more info about how this works, see section 4.2.13 below.

The feature STEMHEAD of a lexical entry is used by inflectional affixes to ensure the desired stem without overly restricting the value of HEAD itself, since this needs to remain non-specific to allow conjunction of non-similar heads. This feature is only given a more specific value for lexical entries which get inflected (verbs, nouns, and adjectives).

4.1.4 Cat objects

Objects of type *cat*, the value of the feature CAT, have the well-known appropriate features HEAD and VAL, and the MERGE specific features MC, POSTHEAD and HC_LEX.

The feature MC (standing for main clause) has three possible values: +, -, and *na* (non-applicable). Non-clauses are always [MC *na*], since they can't really be said to be root or non-root. All clauses have the MC value *bool*, and if they have a restricted distribution then they are [MC +] or [MC -].

The feature POSTHEAD value is used to encode the relative position of a modifier with respect to the head.

The feature HC_LEX is used to identify the LEX value of *synsem* objects that results from combining the word with its complements. This is done in the head-complement rule which structure-shares the LEX value of the mother with the HC_LEX of its head dtr (see section 4.2.4). Most words are specified as being [HC_LEX *minus*] in the lexicon, i.e., if they combine with their complements in a head-complement phrase, the resulting phrase is [LEX *minus*]. But the head-complement structures "thirty-two" and "two o'clock", for example, are still lexical signs ([LEX *plus*]), not phrasal ones, since they can appear as prenominal adjectives and in noun-noun compound constructions, respectively.

4.1.5 The content

4.1.6 Head objects

Objects of type *head*, the value of the feature HEAD, have the well-known appropriate feature MOD, PRD,INV, and AUX, and the MERGE specific features TAM, POSSABLE, POSS, STRICT_HEAD, MOBILE.

The feature TAM encodes the tense, aspect and mood values of signs. The three features POSSABLE, STRICT_HEAD and MOBILE are encoded as subtypes of head in the ERG grammar and all other subtypes of *head*, like verb, noun, adj, etc, are cross-classified with these three types according to whether they are possable, a strict type or mobile. Since this results in huge type hierarchy below head we decided to encode these three properties as boolean features. The feature POSSABLE encodes whether whether a sign with a certain *head* type can in principle be a possessive (?) and the feature POSS then says whether it really is a possessive. This means only *head* types which are [POSSABLE *plus*] can be [POSS *plus*].

The feature STRICT_HEAD is needed for coordination (see ERG file syntax.tdl, line 2320 for more information).

The feature MOBILE marks those things that can be extracted in the complement-extraction rule, from which, for example, nominative-case NPs are excluded by not marking all nouns as [MOBILE *plus*].

4.1.7 Lists

The type hierarchy below *list* implemented in the MERGE grammar should also be briefly discussed here because it is more complex than the usual distinction between empty and non-empty lists. This hierarchy is shown in figure 4.6.

The type *ne_list* introduces the features HD and TL and the principles shown in figure 4.7 further constrain the *list* subtypes. One of the types that occurs in many parts of the MERGE

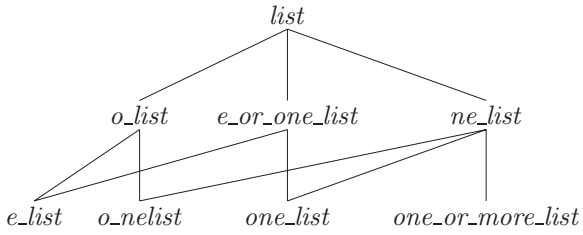


Figure 4.6: The type hierarchy below *list*

```
o_nelist *> (hd:(unexpressed,
                opt:plus),
             tl:o_list).

one_list *> tl:e_list.

one_or_more_list *> tl:ne_list
```

Figure 4.7: Constraints on *list* subtypes

is the type *o_list* which is used within the valence features to require the saturation of a phrase. A phrase is thus saturated if the value of its COMPS list is *o_list*, i.e., if the COMPS list is either empty or only consists of elements that are of type *unexpressed* (a subtype of *synsem*), as required by the constraint on *o_nelist* shown in figure 4.7.

4.2 Phrase Structure Rules

4.2.1 Phrasal types and phrase structure rules

Every phrase structure rule of the MERGE licenses a phrase corresponding to one of the atomic types in the type hierarchy below *phrase*. To get an overview of the different phrase structure rules so far implemented in the grammar it is therefore helpful to look at the general set up of the type hierarchy below *phrase* which we already saw in figure 2.15.

Phrases are further divided into headed and non-headed phrases and binary and unary phrases. All the atomic types corresponding to the phrase structure rules are subtypes of the five types *head_initial*, *head_final*, *head_only*, *non_head_only*, and *non_headed2_phrase*.

In the following, each phrase structure rule belonging to an atomic type in the hierarchy below *phrase* is described in detail. The purpose of the more general phrasal types and especially the constraints on these types, like the head feature principle applying to *headed-phrase*, are described in section 4.5.

4.2.2 General phrasal macros

There is a number of general macros which occur repeatedly in the specification of the phrase structure rules. These are briefly described here.

```
phr_synsem macro (canonical_synsem,
                  lex:minus).

phrasal macro (phrase,
               synsem: @phr_synsem).
```

Figure 4.8: Phrasal macro

```
head_nexus_rel_phrase macro (headed_phrase,
    synsem:nonlocal:rel:Rel,
    head_dtr:synsem:nonlocal:rel:Rel).

head_nexus_que_phrase macro (headed_phrase,
    synsem:nonlocal:que:Que,
    head_dtr:synsem:nonlocal:que:Que).

head_nexus_phrase macro (@head_nexus_rel_phrase,@head_nexus_que_phrase).
```

Figure 4.9: head_nexus_phrase macro

The macro `head_nexus_phrase`, which calls the two macros `head_nexus_que_phrase` and `head_nexus_rel_phrase`, as shown in figure 4.9, ensures that in headed phrases the REL and QUE values are shared between the head daughter and the phrase itself. The macro `head_valence_phrase`, shown in figure 4.10 ensure, that also the SLASH value of a headed phrase is identical to its head daughter's SLASH value. The name of this macro might suggest

```
head_valence_phrase macro (@head_nexus_phrase,
           synsem:nonlocal:slash:Slash,
           head_dtr:synsem:nonlocal:slash:Slash).
```

Figure 4.10: head_valence_phrase macro

that it also functions as some kind of valence principle. But it should be noted here that there is no general valence principle in the entire MERGE. Instead, the saturation of the valence features COMPS, SUBJ and SPR is specified in the respective phrase structure rules directly and will be described in the following sections.

Another macro whose name is a bit misleading is the macro `head_compositional` shown in figure 4.11. It does not act as a kind of semantics principle for headed phrases. Instead it specifies that certain parts of the head daughter's CONT value, namely the TOP, INDEX, and E_INDEX, are identical to these feature values in the phrase's C_CONT value.

```
head_compositional macro (headed_phrase,
   c_cont:( top:Hand,
            index:Index,
            e_index:Event),
   head_dtr:synsem:local:cont:( top:Hand,
                                index:Index,
                                e_index:Event)).
```

Figure 4.11: head_compositional macro

```
non_clause macro (@head_nexus_phrase,
   synsem:local:(cat:mc:na,
                 keys:message:e_list)).


clause macro (@phrasal,
    synsem:( local:(cat:( head:v_or_g,
                          val:comps:o_list),
                    conj:cnil),
             nonlocal:que:e_list)).
```

Figure 4.12: clause and non_clause macro

The macro `clause` specifies what description a phrase needs to satisfy in order to be a clause, i.e., it must be [LEX *minus*], its head type must be verb or gerund, it must be saturated ([COMPS *o_list*]), not a conjunct ([CONJ *cnil*]) and have an empty QUE list. For a non-clause, the feature MC (main clause) is not applicable and the message must be an empty list.

The macro `clause` in connection with the macro `decl_phrase` then specifies what requirements a declarative phrase needs to fulfill as shown in figure 4.13. A declarative phrase is thus a phrase that is not inverted, can have a boolean value for MC and meets certain semantic requirements.

4.2.3 Head-subject rule

The phrase structure rule *subjh* licensing phrases of type *head_subj_phrase* in the MERGE and the macro *@head_subj_phrase*, which contains most of the specifications used in this rule are shown in figure 4.14. As usual in a head-subject phrase, the *synsem* value of the non_head_dtr must be identical to the element on the head daughter's SUBJ list. To encode the fact that a *head_subj_phrase* has saturated its subject, the SUBJ list contains one element, an *anti_synsem*, instead of being completely empty. In addition, the COMPS list of the mother and the two daughters in a head-subject phrase must be saturated, i.e., either *o_list* or *e_list*.

```
non_rel_phrase macro (@head_nexus_rel_phrase,
   synsem: ( local:keys:message:[(message,Msg,
                                  (handel:Mhand,
                                    soa:Soahand))],
             nonlocal:rel:e_list),
    head_dtr:synsem: (local:cont: (top:Hdhand,
                                   index:Ind ),
                      nonlocal:( que:e_list,
                                 rel:e_list )),
    c_cont:(top:Mhand,
            index:Ind,
            liszt:hd:Msg,
            h_cons:hd:(sc_arg:Soahand,
                       outscpd:Hdhand))).


decl_phrase macro (@non_rel_phrase,
    synsem:local:( cat:( head:inv:minus_star,
                         mc:bool ),
                   keys:message:[prpstn_rel]),
    c_cont:(liszt:[relation],
             h_cons:[qeq])).

decl macro (@decl_phrase,@clause).
```

Figure 4.13: decl macro

```
head_subj_phrase(HeadDtr,NonheadDtr) macro (head_subj_phrase,
                                            @head_valence_phrase,
                                              @decl,
     synsem:local:( cat:( head:( v_or_g,
                                 vform: fin_star,
                                  prd:minus,
                                  tam: @ind_or_mod_subj_tam ),
                            posthead:plus,
                            mc:bool,
                            val:( subj:[anti_synsem],
                                  comps:e_list,
                                  spr:Spr)),
                    keys:altkey:Altkey,
                    conj:cnil ),
    head_dtr:(HeadDtr,synsem:local:( cat:( val:( subj:[Synsem],
                                         spr:Spr,
                                         comps:o_list),
                                    mc:na ),
                                keys:altkey:Altkey)),
    non_head_dtr:(NonheadDtr,synsem:( Synsem,canonical_synsem ,
                    local:( cat:( head:(subst,poss:minus),
                                  val:( subj:o_list,
                                        comps:o_list,
                                        spr:o_list ))),
                    nonlocal:( slash:e_list,
                               rel:e_list,
                               que:e_list
                                           )))).

subjh ## @head_subj_phrase(HeadDtr,NonheadDtr)
  ===>
cat> NonheadDtr,
cat> HeadDtr.
```

Figure 4.14: The head-subject rule

4.2.4 Head-complement rule

The phrase structure rule `hcomp` and the macro `head_comp_phrase` licensing phrases of type *head_comp_phrase* are shown in figure 4.15. The only important information that is directly

```
head_comp_phrase(HeadDtr,NonheadDtr) macro (head_comp_phrase,
                                            @head_comp_or_marker_phrase(NonheadDtr),
    synsem: (local:( cat:posthead:Ph,
                     conj:cnil),
             lex:Lex),
    head_dtr: (HeadDtr, synsem:local:cat:( posthead:Ph,
                                           hc_lex:Lex )) ).

hcomp ## @head_comp_phrase(HeadDtr,NonheadDtr)
  ===>
cat> HeadDtr,
cat> NonheadDtr.
```

Figure 4.15: The head-complement rule

specified in this macro is the fact that the mothers LEX value is identical to the head daughter's HC_LEX value. This means that a head-complement phrase is only [LEX *minus*] if this is required by the head daughter via its HC_LEX value. As already mentioned in section 4.1.4 above, this also allows head complement phrase to also be [LEX *plus*] and thus to function as lexical signs which is necessary for expression like "thirty-two".

The head of a head-complement phrase must be specified as conj:cnil (which distinguishes it from the heads of head-marker phrases).

The rest of the specifications necessary for head-complement phrases are part of the macro `head_comp_or_marker_phrase` shown in figure 4.16. It should be noted here that, although this macro mentions the term "marker", the MERGE grammar does not contain a marker phrase which is significantly different from the head-complement construction. Instead, those elements which were traditionally markers in HPSG theories are lexical heads selecting the phrase they mark via the COMPS list. Thus, traditional markers, such as complementizers, occur as the heads of head-complement phrases, while other elements, such as conjunctions, occur as the heads of the closely related head-marker phrases (see section 4.2.6).

The macro `head_comp_or_marker_phrase` basically functions as a valence principle specifying that the *synsem* value of the non-head daughter must be identical to the first element on the head daughter's COMPS list and that the rest of this COMPS list is passed up to the mother.

In addition to HEAD, feature values passed from head daughter to mother include AGR, MC, ALTKEY, KEY, MESSAGE, CONJ and the *nonlocal* features. The non-head daughter must be specified as conj:cnil_or_numconj.

```
head_comp_or_marker_phrase(NonheadDtr) macro (@head_valence_phrase, @head_compositional,
                                              head_initial,
    synsem:( canonical_synsem ,
             local: (cat:( mc:Mc,
                           val:( subj:Subj,
                                 comps:Comps,
                                 spr:Spr ) ),
                     keys:( altkey:Altkey,
                            message:Hmsg ) ) ),
    head_dtr:synsem:local:( cat:( mc:Mc,
                                  val:( subj:Subj,
                                        comps:[Synsem|Comps],
                                        spr:Spr ) ),
                            keys:( altkey:Altkey,
                                   message:Hmsg ) ),
    non_head_dtr:(NonheadDtr,synsem:(Synsem,canonical_synsem)),
    c_cont:(liszt:e_list,
            h_cons:e_list)).
```

Figure 4.16: `head_comp_or_marker_phrase`

4.2.5 Optional complement rules

4.2.5.1 Head-optional-complement rule

A number of lexical heads in the MERGE have optional complements. To ensure that a phrase can be saturated even if these optional complements are not realized there were originally two head-optional-complement rules (hoptcomp and hopt2comp) as shown in figure 4.17.

The hoptcomp rule removes one optional complement from the COMPS list, whereas the hopt2comp rule removes two optional complements from the COMPS list.

As discussed in section 2.1.1.3, we eliminated these two optional complement rules and instead introduced a more general head complement schema, which realizes the first element on a COMPS list which is not an optional, unrealized complement.

4.2.5.2 Noun-optional-complement rule

Noun-optional-complement phrases (phrases of type *noun_opt_comp_phrase* licensed by the *noptcomp* rule), are head-only phrases that remove the head of the daughter's COMPS list, provided it is specified as opt:plus and has empty NONLOCAL lists. The tail of that list, as well as the LEX specification, are inherited by the mother. The daughter has a HEAD value of type *n_or_p*; that is, nominal or prepositional.

4.2.6 Head-marker rules

Rather misleadingly named, the "marker" in a so-called head-marker phrase in MERGE is always a conjunction, and it is, in fact, the head daughter. Head-marker phrases are only used

```
head_opt_comp_phrase(Dtr-sign) := (head_opt_comp_phrase,@basic_head_opt_comp_phrase,
    synsem:local:cat:val:comps:Comps,
    head_dtr: (Dtr, inflected:plus,
               synsem:local: (cat:val:comps:
                                    [(unexpressed,opt:plus)|
                                    (Comps,[expressed_synsem|_])],
                              keys:key:event_rel ))).

    hoptcomp ## @head_opt_comp_phrase(Dtr)
  ===>
cat> Dtr.

head_opt_two_comp_phrase(Dtr) := (head_opt_two_comp_phrase, @basic_head_opt_comp_phrase,
    synsem:local:cat:val:comps:Comps,
    head_dtr: (Dtr, inflected:plus,
               synsem:local:( cat:val:comps:
                                [ (unexpressed,opt:plus)|
                                  [ (unexpressed,opt:plus)|
                                      (Comps,[expressed_synsem|_])]],
                              keys:key:event_rel))).

hopt2comp ## @head_opt_two_comp_phrase(Dtr) ===>
cat> Dtr.
```

Figure 4.17: The head-optional-complement rules

```
basic_head_opt_comp_phrase macro (basic_head_opt_comp_phrase,
                                  @head_valence_phrase,
                                  @head_compositional,
    inflected:Infl,
    posscl:Posscl,
    synsem:(canonical_synsem,
            local:(cat:(val:(subj:Subj,
                             spr:Spr,
                             spec:Spec),
                        hc_lex:Lex,
                        mc:Mc,
                        posthead:Ph),
                   conj:cnil,
                   keys:(altkey:Altkey,
                         message:Msg )),
            modified:Mod),
    head_dtr:(inflected:Infl,
              posscl:Posscl,
              synsem:(local:(cat:(val:(subj:Subj,
                                       spr:Spr,
                                       spec:Spec),
                                  mc:Mc,
                                  hc_lex:Lex,
                                  posthead:Ph),
                             keys:( message:Msg,
                                    altkey:Altkey)),
                      modified:Mod)),
    c_cont:(liszt:e_list,
            h_cons:e_list)).
```

Figure 4.18: The macro `basic_head_opt_comp_phrase`

for coordination in the ERG and MERGE, not for complement clauses with a complementizer, as usually in HPSG.

Head-marker phrases are head-initial, and like head-complement phrases in that the non-head daughter has a SYNSEM value structure-shared with the head of the head daughter's COMPS list (see the discussion of @head_comp_or_marker_phrase in section 4.2.4). The tail of that list, as well as the SUBJ and SPR specifications (and others), are inherited by the mother. The head daughter must be an (inflected) word, specified as conj:strict_conj. In fact, this is the only property distinguishing the heads of head-marker phrases from the heads of head-complement phrases; the head of a head-complement phrase must be specified as conj:cnil.

NONLOCAL feature specifications are structure-shared between both daughters and mother. The mother's LEX specifications are inherited from the non-head-daughter.

4.2.6.1 Nominal head-marker rule

In phrases of type *head_marker_phrase_nom*, licensed by the hmarknom rule and described by the macro @head_marker_phrase_nom, the non-head daughter has a KEY value of type *non_event_rel*.

4.2.7 Head-specifier rule

Phrases of type *head_spec_phrase*, licensed by the rule *hspec* and described by the macro @head_spec_phrase, are head-final, and are characterized by mutual selection between the head and the specifier. The COMPS value is inherited from the non-head daughter.

In a head-specifier phrase, the non-head daughter's SYNSEM value is structure-shared with the the single member of the head daughter's SPR list. The tail of the list is inherited by the mother, as are SPEC and SUBJ. COMPS is specified as *o_list*.

Furthermore, the non-head daughter's SPEC list contains a *synsem* which must be consistent with the head daughter's in a variety of respects (as enforced by unification): HEAD, COMPS, INDEX, TOP, KEY, and MODIFIED.

Both daughters are specified as inflected:plus.

4.2.8 Modification

In MERGE, modification is licensed via the list-valued MOD feature, which is either empty or contains a *synsem*.

Phrases involving modification are described by the macro @basic_head_mod_phrase_simple. The modifier is the non-head daughter, and the *synsem* on its MOD list is structure-shared (or certain paths are structure-shared) with the *synsem* of the head-daughter. Currently, the structure-sharing of paths is specified as it is in ERG, though structure-sharing at the *synsem* level would be preferable.

In addition to HEAD, the features with values structure-shared between mother and head daughter include AGR, SPR, SUBJ, CONJ, ALTKEY, and KEY. A head-modifier phrase is specified as modified:hasmod, and both its COMPS and REL lists must be empty. The mother inherits the values of both MC and MESSAGE from the non-head daughter.

The non-head daughter must have *o_list*-valued COMPS and SPR lists, a KEY value of type *independent_rel*, an empty QUE list, and be specified as conj:cnil.

Specific constructions that involve modification are either a kind of adjunct-head phrase (pre-modification), or a kind of head-adjunct phrase (post-modification). Properties of both classes are discussed below, and descriptions of specific construction types follow.

Adjunct-head phrases are head-final, described by the macro @adj_head_phrase. An adjunct-head phrase is specified as modified:lmod_star, and the value of the *xmod* feature PERIPH is structure-shared with that of the non-head daughter. The values of POSTHEAD and LEX are inherited from the head daughter, and the non-head daughter must be specified as prd:minus, and have empty COMPS, REL and QUE lists.

Intersective adjunct-head phrases have a head daughter specified as tense:no_tense and posthead:minus.

Head-adjunct phrases are head-initial, described by the macro @head_adj_phrase. A head-adjunct phrase is specified as posthead:plus and modified:rmod_star. QUE and REL are inherited from the head-daughter. The head daughter's value of the feature MODIFIED must be *notmod_or_rmod*. The non-head daughter must be specified as posthead:plus, and QUE must be empty.

4.2.8.1 Intersective adjunct-noun rule

A phrase of type *adj_n_int_phrase*, described by the macro @adj_n_int_phrase and licensed by the rule *adjn_i*, is an intersective adjunct-head phrase with a HEAD value of type *noun*. SPR specifies something with a KEY value of type *quant_rel*.

4.2.8.2 Noun-adjunct rules

Noun-adjunct rules license head-adjunct phrases in which the head daughter is a so-called N'. Postnominal modifiers are grouped into two classes: reduced relatives, and relative clauses. Separate rules license modification by each, but all rules licensing postnominal modification are described by @n_adj_int_phrase. A noun-adjunct phrase has a HEAD value of type *noun*, an empty COMPS list, and is specified as modified:periph:na_or_minus. The non-head daughter must be a phrase specified as prd:plus, mc:na, and it must have an empty SLASH list.

Reduced relative noun-adjunct rules Both rules licensing postnominal modification by a so-called reduced relative are described by @n_adj_redrel_phrase. In this kind of a phrase, the non-head daughter has a MOD list specifying something with a LOCAL value of type intersective_mod. The non-head daughter itself is specified as tense:no_tense, its REL list is empty, and its subject is of type *unexpressed*.

Reduced relatives are not a phrasal type *per se*, and unlike other relatives they have an empty REL list. Qualified candidates include prepositional phrases and present participial verb phrases.

Phrases of type *n_adj_redrel_temp* or *n_adj_redrel_nontemp* (involving a so-called reduced relative), are licensed by the rules *nadj_rr_t*, and *nadj_rr_nt* respectively.

Nontemporal reduced relative noun-adjunct rule The KEY value is of type *nontemp_rel* and an ALTKEY value of type *nontemp_or_conj_rel*.

Temporal reduced relative noun-adjunct rule The value of the feature KEY is of type *temp_loc_abstr_rel* and an ALTKEY value of type *temp_abstr_or_conj_rel*.

Relative clause noun-adjunct rule Phrases of type *n_adj_relcl_phrase* are licensed by the *nadj_rc* rule, described by the macro @n_adj_relcl_phrase(HeadDtr,NonheadDtr). The non-head daughter must have a HEAD value of type *verbal*, a VFORM value of type *fin_or_inf*, and *o_list*-valued SUBJ and COMPS lists. It is not required to have a non-empty REL list.

This kind of phrase is specified as periph:plus, which blocks iterative modification by relative clauses, since the MOD value of a relative clause specifies something marked as modified:periph:na_or_minus. SPR specifies something with a KEY value of type i_or_e_quant_deg_rel.

4.2.9 Filler-head rules

4.2.9.1 Filler-head relative rule

Phrases of type *filler_head_rule_rel* are non-headed binary phrases licensed by the fillhead_rel rule. Apparently, they are non-headed in order to allow a prd:plus specification on the mother and a prd:minus specification on the would-be head daughter. The macro @n_adj_int_phrase (invoked by all rules licensing postnominal modification) requires that the non-head daughter be specified prd:plus. This property distinguishes relative clauses from all others, and thus the specification acts as a filter.

The mother inherits *head* feature AUX, INV, MOD, TAM, and VFORM from the non-head2 daughter. The HEAD value is of type *verb*, specified as finite (vform:fin), noninverted (inv:minus), predicative (prd:plus), mood:strict_ind_or_mod_subj, with a nonempty MOD value specified as a *synsem_min* with a LOCAL value of type *intersective_min*.

Wh-relatives are specified as posthead:plus; they have an empty COMPS list, an *o_list*-valued SUBJ value inherited from the non-head2 daughter, and a SPR list containing an *anti_synsem* with empty NONLOCAL lists. They have empty QUE and SLASH lists, but a non-empty REL list.

The non-head2 daughter has a nonempty SLASH list, whose member is structure-shared with the SYNSEM value of the non-head daughter. This *synsem* is specified as having *o_list*-valued SUBJ and COMPS lists. Furthermore, the non-head2 daughter is inflected, has an *o_list*-valued COMPS list, and empty QUE and REL lists.

The non-head (filler) daughter has a non-empty REL list, empty QUE and SLASH lists, and is inflected.

4.2.10 Non-*wh*-relative rules

Non-*wh*-relatives include the maximally specific phrase types *fin_non_wh_rel_cl* (finite non-*wh*-relatives, or bare relatives) , and *inf_non_wh_rel_cl* (infinitive non-*wh*-relatives, or simple infinitive relatives). These phrases are licensed by the fin_non_wh_rel and inf_non_wh_rel rules, respectively, and are distinguished primarily by their VFORM and SUBJ values.

The mother has empty SUBJ and COMPS lists, regardless of what the daughter's values were (although the daughter's COMPS list must be of type *o_list*). Thus, the daughter may be phrasal, as is the case with simple infinitive relatives with complement gaps.

4.2.11 Extracted argument rules

Extracted-argument phrases are *head_only* phrases, so the HFP applies, and are required to satisfy the description @head_valence_phrase, so nonlocal features are passed from daughter to mother.

In every case, some valence item is specified as an object of type *gap*, a *synsem* object whose LOCAL value is structure-shared with the element on its SLASH list. Previously discussed constraints and specifications ensure that this SLASH value ends up on the mother's SLASH list.

4.2.11.1 Extracted complement rule

In a phrase of type *extracted_comp_phrase*, licensed by the rule *extracomp*, the first element on the COMPS list of the daughter is a gap synsem. The rest of the COMPS list, as well as the SUBJ and SPR values, are passed up to the mother. A simplified version of the macro describing a phrase of type *extracted_comp_phrase* is shown in figure 5.10.

4.2.11.2 Extracted subject rules

Phrases of type *extracted_subj_phrase_fin* or *extracted_subj_phrase_inf*l licensed by the rules extrasubj_f and extrasubj_i respectively, have empty valence lists. The head daughter has a gap specified as case:nom on its SUBJ list. A simplified version of the macro describing a phrase of type *extracted_subj_phrase_fin* is shown in figure 5.11.

4.2.12 Extracted adjunct rules

Since adjuncts do not appear on any ARG_S list, adjunct "extraction" must be handled differently than those cases involving arguments. Phrases of type *extracted_adj_int_phrase* are a subtype of *head_only_phrase*, so the HFP applies, but are not head-valence phrases. Thus, a SLASH value may be specified on the mother which is not present on the daughter. That SLASH value has a MOD value consistent in a variety of respects (HEAD, VALENCE, etc.) with the daughter. The daughter must have an empty SUBJ list.

4.2.13 Coordination rules

In a coordinate structure, described by @coord_phr (a phrase of type *basic_coord_phr*, which is a subtype of non_headed2_phrase, so that it is binary and non-headed), both daughters and the mother structure-share their LEX, CAT, MODIFIED and NONLOCAL feature values. Furthermore, the phrase's PNG value is inherited from the NON_HEAD_DTR2 (the right daughter), while the KEY value is inherited from the NON_HEAD_DTR (the left daughter). The phrase is specified as divisible:minus_star.

The grammar distinguishes between top- and mid- coordination rules. Top-coordination rules specify that the ALTKEY value of the phrase is structure-shared with both daughters, but the INDEX and EINDEX values are inherited from the non-head2 daughter.

Mid-coordination rules specify that the ALTKEY value of the phrase is of type *conj_rel*, and is the sole member of the C_CONT LISZT.

In a top-coordination phrase, the non-head daughter has a CONJ value of type *conj*, while the non-head2 daughter has a CONJ value of type *complex_conj*. The value of the non-head daughter's CONJ feature is the head of the non-head2 daughter's conj:chead:left list. In a mid-coordination phrase, the phrase has a CONJ value of type *phr_conj*, and its CHEAD value is inherited from the non-head2 daughter. The non-head daughter has a CONJ value of type *conj*, while the non-head2 daughter has a CONJ value of type *complex_conj*. The value of conj:chead:left is a list with two members, the second of which is the CONJ value of the non-head daughter.

Event coordination rules Phrases of type *top_coord_event* are licensed by the *top_coord_e* rule. Both daughters have a KEY value of type *event_or_degree_or_no_rel*, and the non-head daughter has a nonempty SPR list containing a *synsem* specified as key:relation. The mother has an *o_list*-valued COMPS list, and an INDEX value of type *conj_event*.

Phrases of type *mid_coord_event* are licensed by the rule *mid_coord_e*. Both daughters have a KEY value of type *event_rel*, and the mother has an INDEX value of type *conj_rel*.

Nominal coordination rules Nominal coordination rules specify that both daughters have an INDEX value of type *ref_ind*, that their DIVISIBLE values are structure-shared, and that the non-head daughter has a HEAD value of type *n_or_a*, and an empty COMPS list.

Phrases of type *top_coord_nom* are licensed by the *top_coord_nom* rule.

Phrases of type *mid_coord_nom* are licensed by the *mid_coord_nom* rule. Both daughters have a KEY value of type *basicNom*, and the C_CONT INDEX value is specified as pn:nonfirstsg.

4.2.14 Special NP rules

4.2.14.1 Specifier-less noun phrases

These head-only phrases, which are all of type *generic_bare_np_phrase*, remove the daughter's SPR specification, while still incorporating the (unrealized) specifier's KEY value into the semantics of the phrase (via C_CONT).

The daughter has a HEAD value of type *noun*, is inflected, and has an empty COMPS list, an *o_list*-valued SUBJ list, and an empty SLASH list. Its KEY value is of type *nonpro_rel*.

The mother has an ALTKEY value of type *implicit_quant_rel*, and empty valence lists.

Proper noun phrase rule Phrases of type *proper_np_phrase* are licensed by the rule *proper_np*. Eligibility to be the daughter of a proper noun phrase is primarily semantic; the daughter's KEY value must be of type *abstr_named_np_rel*, with an ALTKEY value of *basic_nom_rel*. Further, the daughter is specified as divisible:minus_star (in other words, it is a singular count noun). The specifier's KEY value is of type *def_np_rel*.

Bare noun phrase rule Phrases of type *bare_np_phrase* are licensed by the rule *bare_np*. Bare noun phrases involve common nouns that do not need specifiers to be licensed as NPs – like mass nouns and plurals. Eligibility depends upon the specification of the value of the feature DIVISIBLE as *plus_star*. The specifier's KEY value is of type *udef_rel*.

Bare verbal gerund phrase rule Phrases of type *bare_vger_phrase* are licensed by the rule *bare_vger*. The daughter of a bare gerund phrase is specified as head:gerund, with an *o_list*-valued COMPS list, a KEY value of type *gerund_rel*, and an ALTKEY value of type *no_rel*, which is not only inherited by the mother, but contributed to the semantics of the phrase via structure-sharing with a C_CONT specification, which further specified as being of type *udef_rel*. Thus these gerund phrases are implicitly quantified in a way similar to bare noun phrases. Another similarity to bare noun phrases is that the daughter is specified as synsem:local:cont:index:divisible:plus, and the valence lists of the mother are all empty.

4.2.14.2 Compound noun phrases

Compound noun phrases are head-final, inheriting the AGR, VAL, and NONLOCAL feature values from the head-daughter, which must be specified as lex:plus_star, and the feature MODIFIED must have the value *notmod_or_lmod*. Further, the head daughter has a non-empty SPR list, specifying a non-optional specifier with KEY value *quant_rel*. The non-head daughter has empty *nonlocal* lists. Both daughters have an empty COMPS list, are specified as inflected:plus, and have a KEY value of type *non_pro_rel*. Noun compounds are specified as lex:plus_star, with a *head* value *noun_star* and an empty MOD list.

Noun-noun compound rule Phrases of type *noun_n_cmpnd_phr* are licensed by the *noun_n_cmpnd* rule. The head daughter of a noun-noun compound phrase is specified as periph:minus. The non-head daughter is specified as lex:plus_star, with a HEAD value *noun_star*, specified further as prd:minus. Its non-optional specifier *synsem* specification is of type *unexpressed*. Both daughters have an ALTKEY value of type *no_rel*. The mother inherits its modified:periph specification from the non-head daughter.

NP-noun compound rule Phrases of type *np_n_cmpnd_phr* are licensed by the *np_n_cmpnd* rule. NP-noun compound phrases are specified as modified:periph:na. The head daughter is specified as lex:plus, and modified:periph:na_or_minus, and has an ALTKEY value of type *no_rel*. The non-head daughter has an empty SPR list, a KEY value of type *abstr_named_rel*, and an ALTKEY value of type *implicit_quant_rel*. It is specified as lex:plus.

NP-name compound rule Phrases of type *np_name_cmpnd_phr* are licensed by the rule *np_name_cmpnd*. NP-name compound phrases are specified as modified:periph:na. The head daughter has a KEY value of type *named_np_rel*, while the non-head daughter has a KEY value of type *named_np_rel* and an ALTKEY value of type *implicit_quant_rel*. Both daughters are specified as modified:periph_plus and lex:plus.

4.2.14.3 Temporal modifier rule

Phrases of type *temp_mod_nonwh_phrase* are licensed by the *temp_nwh* rule. Basically, these are derivational non-head-only phrases, taking temporal noun phrases and making them modifiers.

The daughter is an ordinary NP (head:noun, inflected:plus, spr:o_list, comps:⟨ ⟩), with a KEY value of type *modable_rel*, and an empty REL list.

Inherited from the daughter are PRD, COMPS, INDEX, and NONLOCAL feature specifications. The daughter's KEY value is structure-shared with the mother's ALTKEY value.

The phrase's own HEAD value is of type *modnp_star*, and it has a nonempty MOD list containing a *synsem_min* with HEAD value of type *nominal_or_verbal*, a nonempty SPR list (a *synsem_min* with KEY value of type *quant_or_deg_rel*), and an *o_list*-valued COMPS list.

Further, this phrase subcategorizes for both a non-optional subject, and an optional specifier.

The subject is specified as *nominal*, with *head* specifications poss:minus, strict_head:plus, and tam:mood:ind_or_mod_subj. The subject has an empty COMPS list, *o_list*-valued SPR and SUBJ lists, and empty NONLOCAL lists.

The specifier has a HEAD value of type *adv*, an *o_list*-valued SPR list, and empty QUE list, and has a KEY value of type *degree_rel*.

The KEY value of the phrase is of type *unspec_loc_rel*, and is contributed to the semantics of the phrase via C_CONT.

A phrase of type *temp_mod_nonwh_phrase* has the additional specifications that the specifier is an *anti_synsem*, and the daughter has empty SLASH and QUE lists.

4.2.14.4 Measure NP rule

Phrases of type *measure_np_phrase* are licensed by the measure_np rule.

A measure-NP phrase is a binary non-headed phrase with a HEAD value *adv_or_partn*, with empty MOD, SUBJ, COMPS, and NONLOCAL feature lists. Their SPEC list specifies a predicative adjective (head:adj, prd:plus) whose ALTKEY value is of type *non_conj_rel*, and whose KEY value is of type *mod_relation*. They select an optional specifier (via SPR) with a HEAD value of type *adv*, a KEY value of type *degree_rel*, an *o_list*-valued SPR list and an empty QUE list. Its own KEY value, which is the first element on the C_CONT LISZT, is of type *deg_rel*. AGR and INFLECTED specifications are inherited from that daughter. MODIFIED value is inherited from the non-head daughter.

The non-head2 daughter is a lexical item (lex:plus_star) with HEAD value of type *noun*, empty SUBJ, COMPS, SLASH lists. It selects an non-optional specifier with empty *nonlocal* feature lists, whose KEY value is of type *udef_rel*. That *rel* binds the index of the non-head2 daughter, which is specified as gen:real_gender and pn:strict_nonthirdsg. Its own KEY value is of type *nonpro_rel*, and its ALTKEY value is of type *no_rel*.

The non-head daughter has a HEAD value of type *intadj*, has an empty COMPS list, *o_list*-valued SPEC and SPR lists, and is specified as inflected:plus. Its KEY value is of type *const_rel*, and its ALTKEY value is of type *mod_relation*.

4.2.14.5 Free relative rules

Free relatives are head-initial phrases, with a nominal or prepositional head value (*n_or_p*), empty valence lists, and empty nonlocal feature lists. The head daughter of a free relative phrase has a nonempty QUE list, and a SLASH value structure-shared with the SLASH value of the non-head daughter. The head daughter is further specified as having an empty COMPS list, and *o_list*-valued SPR and SUBJ lists.

The non-head daughter is a finite or infinitive uninverted clause or VP; that is, it has a *verbal* HEAD value, and is specified as inv:minus and vform:fin_or_inf, and *o_list*-valued COMPS and SPR lists. Its SLASH value is specified as structure sharing its KEY, ALTKEY and INDEX values with the head daughter. Thus, the relationship between the gap and the wh-phrase is semantic, while the purely syntactic specifications of the non-head daughter's SLASH value are tied to the SLASH value of the free relative pronoun, which are specified in their lexical entries. This allows for a syntactic mismatch between the gap in the non-head daughter and the properties of the free relative pronoun which serves as filler, while ensuring a semantic fit.

The free relative phrase itself inherits its AGR specifications from the head daughter, and has empty VAL and NONLOCAL lists.

Infinitive free relative rule These are phrases of type *free_rel_inf_phrase*, licensed by the freerel_inf rule. The non-head daughter of an infinitive free relative phrase is an infinitive VP, whose unsaturated SUBJ list contains a *synsem* with an empty SLASH list.

Finite free relative rule These are phrases of type *free_rel_fin_phrase*, licensed by the freerel_fin rule. The non-head daughter in a finite free relative phrase is finite, with an *o_list*-valued SUBJ list.

4.3 Lexical Entries

4.3.1 Common nouns

Head feature properties: The HEAD value is of type *noun*, and MOD list is empty.

Selectional properties: Empty SUBJ list. SPR specifies a non-optional specifier with HEAD value *det*, an empty SUBJ list, and *o_list*-valued COMPS and SPR lists.

Semantic properties: KEY is of type *basic_nom_rel*, and is a member of LISZT.

Other properties: AGR feature value is structure-shared with the INDEX feature value (except where noted). The STEMHD value is *countnstem*.

Intransitive nouns (@n_intr_le)

Selectional properties: Empty COMPS list. The specifier has a KEY value of type *quant_or_wh_rel*.

Examples: dog and *bookcase*

Nouns taking PP complements (@n_ppcomp_le)

Selectional properties: These nouns select a complement with HEAD value of type *prep*, specified as [PRD -], whose semantic argument is its own. Specific lexical entries restrict the relational type of their PP complement.

Semantic properties: The KEY value is of type *diadic_nom_rel*.

Examples: belief and *month*

Plural nouns taking PP complements (@n_plur_ppcomp_le)

Other properties: These nouns have the AGR specification pn:thirdpl_star in the lexicon, and thus show a pattern of plural agreement with singular morphology.

Examples: staff

Nouns taking PP[of] complements (@n_ppof_le) These nouns look much like other nouns selecting PP complements, with the following exception:

Selectional properties: The KEY value of the PP complement is specified as *x_of_rel*.

Examples: manager and *apartment*

Nouns taking CP complements (@n_cpcomp_fin_le)

Selectional properties: These nouns select a non-optional uninverted finite complement clause (*head* value *comp*, all valence lists specified as *o_list*). This CP has the MOOD specification *strict_ind_or_mod_subj*.

Semantic properties: The KEY value is of type *basic_hcomp_nom_rel*, and the value of its (handle-valued) argument is structure-shared with the TOP specification of the CP.

Examples: fact and *belief*

Mass nouns (@n_mass_le)

Selectional properties: The specifier has a KEY value of type *quant_or_wh_rel*.

Other properties: Nouns in the class differ from intransitive nouns only in their STEMHD specification, which is *massnstem*.

Examples: time

4.3.2 Time and date expressions

Hour nouns (@n_hour_le)

Head feature properties: Hour nouns have a LOCAL value of type *nomod_local*, a HEAD value of type *noun_star*, an empty MOD value, and are specified as poss:minus.

Selectional properties: Empty SUBJ list. They select an optional specifier with a HEAD value of type *adv* with *o_list*-valued SPR and COMPS lists, a non-empty SPEC value, and a KEY value of type *degree_rel*. Hour nouns select three optional complements (the last implicitly so). The first complement has a HEAD value of type *intadj*, empty COMPS and SLASH lists, and a KEY value of type *minute_rel*. The second complement has a HEAD value of type *no_head*, has an empty COMPS list, and a KEY value of type *am_pm_rel*. The third complement has a HEAD value of type *prep*, a KEY value of type *temp_loc_rel*, and a MOD value specifying a KEY value structure-shared with the noun's own. This third complement is predicative, and is specified as strict_head:plus, tense:no_tense, and has *o_list*-valued SPR and COMPS lists and a non-empty SUBJ list. All three complements are specified as conj:cnil.

Nonlocal feature properties: Empty NONLOCAL feature values.

Semantic properties: The KEY value is of type *numbered_hour_rel*.

Other properties: The AGR feature value specifies sort:time. The base lexical entries of hour nouns are specified as non-lexical and inflected.

Examples: noon

Temporal PP-complement nouns (@n_temp_ppcomp_le)

Head feature properties: Temporal PP-complement nouns have a HEAD value of type *noun*, an empty MOD value, and are specified as poss:minus.

Selectional properties: They have an empty SUBJ list. They select a non-optional specifier with a HEAD value of type *det*, specified as mobile:minus, with *o_list*-valued SPR and COMPS lists and an empty SUBJ. They select an optional complement with a HEAD value of type *prep*, specified as prd:minus, with an *o_list*-valued COMPS list, and a KEY value of type *x_of_rel*.

Semantic properties: The KEY value is of type *temp_abstr_rel*.

Other properties: The AGR feature value specifies sort:time. Their STEMHD value is of type *countnstem*, and their base lexical entries are specified as lexical and uninflected.

Examples: hour and *day*.

Definite partitive *day* (@n_def_day_part_le)

Head feature properties: This lexical class containing *day* has a HEAD value of type *noun_star*, and an empty MOD value.

Selectional properties: It has empty SUBJ and COMPS lists. It selects a non-optional specifier with a HEAD value of type *det*, specified as mobile:minus, with *o_list*-valued an COMPS list, an empty SUBJ list, and a KEY value of type *abstr_def_rel*.

Nonlocal feature properties: It has empty *nonlocal* feature values.

Semantic properties: Its KEY value is of type *def_day_part_rel*.

Other properties: Specified as posthead:plus. Its AGR feature value specifies sort:time. The base lexical entry is specified as lexical and uninflected. Index/agreement specifications are given in the lexical entry.

Day of week nouns (@n_day_of_week_le) Day-of-week nouns look like temporal PP complement nouns, with these additional specifications:

Selectional properties: The COMPKEY value of the complement is *non_day_diadic_modable_rel*, whose INST value specifies sort:time.

Semantic properties: Furthermore, day-of-week nouns have a KEY value of type *dofw_rel*, and an ALTKEY value of type *basic_nom_rel*.

Examples: tuesday

Day of month nouns (@n_day_of_month_le) Day-of-month nouns look like temporal PP complement nouns, with these additional specifications:

Head feature properties: Specified as prd:plus.

Selectional properties: The specifier has a KEY value of type *abstr_def_rel*, and complement has a COMPKEY value of type *mofy_rel*, whose INST value specifies sort:time. The complement must also have an empty SLASH list.

Nonlocal feature properties: Empty *nonlocal* feature values.

Semantic properties: Day-of-month nouns have a KEY value of type *dofm_rel*, and an ALTKEY value of type *basic_nom_rel*.

Other properties: Specified as inflected:plus. Their AGR feature value (structure-shared with INDEX) specifies that they are non-divisible, neuter, and third-singular.

Examples: first

Cardinal day of month nouns (@n_day_of_month_card_le) Cardinal day-of-month nouns look like ordinary day-of-month nouns in some respects, but their selectional properties are different, they are not predicative, and some semantic feature specifications are different.

Selectional properties: The specifier of a cardinal day-of-month nouns has different semantic feature specifications than the specifier of an ordinary day-of-month noun: the KEY value is of type *implicit_quant_rel*, and the ALTKEY value is of type *mofy_rel*. Cardinal day-of-month nouns also select an accusative *nominal* complement, specified as mobile:plus, poss:minus, strict_head:plus and mood:ind_or_mod_subj. This complement must have *o_list*-valued SUBJ and SPR lists, an empty COMPS list, an empty MESSAGE list, be specified as conj:cnil, and have a KEY value of type *yofc_rel*. (That is, these nouns take a specifier corresponding to the month, and a complement corresponding to the year.)

Semantic properties: The LISZT of a cardinal day-of-month noun has an additional member, of type *unspec_rel*, and the ALTKEY value is unspecified.

Examples: three

Month nouns (@n_month_le) Month nouns nouns look like temporal PP complement nouns, with these additional specifications:

Selectional properties: The complement has a COMPKEY value of type *yofc_rel*, whose INST value specifies sort:time. The complement must also have an empty SLASH list.

Semantic properties: Month nouns have a KEY value of type *mofy_rel*, and an ALTKEY value of type *basic_nom_rel*.

Examples: january

Month-year nouns (@n_month_year_le)

Head feature properties: Month-year nouns have a HEAD value of type *noun_star*, have an empty MOD value and are specified as poss:minus.

Selectional properties: Month-year nouns select a non-optional specifier with a KEY value of type *quant_or_wh_rel*, and a non-optional accusative *nominal* complement, specified as mobile:plus, poss:minus, strict_head:plus, and mood:ind_or_mod_subj. This complement should have an *o_list*-valued SPR list, and empty SUBJ, COMPS and SLASH lists. Its KEY value must be of type *yofc_rel*.

Nonlocal feature properties: Their NONLOCAL lists are all specified as empty.

Semantic properties: The KEY value of a month-year noun is of type *mofy_rel*; also on LISZT is an *x_of_rel* taking the indices of the month-year noun and its complement as arguments.

Other properties: They have the AGR specifications divisible:minus, pn:thirdsg_star, and sort:time. The base lexical entries are specified as lexical and inflected.

Examples: january

Year nouns (@n_year_le) Year nouns look like intransitive nouns, with the following exceptions:

Nonlocal feature properties: Their NONLOCAL lists are all specified as empty.

Semantic properties: The KEY value is of type *yofc_rel*, and the ALTKEY value is of type *basic_nom_rel*.

Other properties: They have the AGR specifications divisible:minus, pn:thirdsg_star, and sort:time.

Examples: nineteen_eightyfour (1984)

4.3.3 Proper nouns (@n_proper_le)

Proper nouns are intransitive nouns, with some additional specifications.

Selectional properties: The agreement feature DIVISIBLE of the specifier is specified as *minus* and the agreement feature PN as *thirdsg*.

Nonlocal feature properties: All NONLOCAL feature lists are specified as empty.

Semantic properties: The KEY value is of type *named_rel*, and the ALTKEY value is of type *basic_nom_rel*.

Other properties: The base lexical entries are marked as inflected:plus, and the agreement features specify divisible:minus and pn:thirdsg. Thus, these lexical entries are not eligible as input to the plural noun lexical rule, and so "Kims", for example, is ungrammatical.

Examples: kim and *csli*

4.3.4 Partitive nouns

Head feature properties: Partitive nouns have the HEAD value *partn*, and an empty MOD list.

Selectional properties: They select an optional specifier with KEY value of type *degree_rel*, with *o_list*-valued SPR and COMPS lists, and empty QUE and REL lists.

Partitive nouns have empty SUBJ lists, but they vary in their COMPS specifications; the complement is either a PP[of], an NP, or nothing, but the semantics of the phrase are the same in any case. That is, "all of the dogs" and "all the dogs" mean the same thing, and "all" alone is implicitly partitive.

Nonlocal feature properties: Partitives have empty *nonlocal* feature values.

Semantic properties: The KEY value of the partitive noun is of type *part_of_rel*, while its ALTKEY value is of type *quant_or_wh_rel*, which are the first two elements of LISZT, respectively. Semantic similarity regardless of selectional difference is encoded via the

synsem-valued KEYCOMP feature specification. The value of KEYCOMP is a phrasal non-optional non-predicative *synsem* with HEAD value *prep* and a KEY value of type *x_of_rel*, which is structure-shared with the partitive noun's COMPKEY value (rather than the KEY value of the complement itself).

Other properties: Marked as lexical and inflected.

PP[of] no-agreement partitives (@n_part_ppof_noagr_le)

Selectional properties: These nouns select a non-optional complement with a HEAD value of type *prep* specified as prd:minus, with an *o_list*-valued COMPS list, an empty SLASH list, and a KEY value of type *x_of_rel*. These specifications are structure-shared with KEYCOMP.

Examples: none and *many*

PP[of] agreement partitives (@n_part_ppof_agr_le) These nouns look like PP[of] non-agreement partitives, with this additional requirement:

Other properties: The AGR feature values for DIVISIBLE and PN be structure-shared with those of the complement.

Examples: half and *most*

NP-complement agreement partitives (@n_part_npcomp_agr_le) These nouns look like PP[of] agreement partitives, with the following exceptions:

Selectional properties: They select an accusative *nominal* complement, with *o_list*-valued SUBJ and SPR lists and an empty COMPS list. This complement must be specified as mobile:plus, poss:minus, strict_head:plus, mood:ind_or_mod_subj, conj:cnil, and its ALTKEY value must be of type *explicit_quant_rel*. This complement is not syntactically related to the KEYCOMP value.

Semantic properties: Since an *x_of_rel* will not be included by the complement, it is included in the LISZT of the partitive noun.

Examples: half

No-complement partitive nouns (@n_part_nocomp_le) These nouns look like other no-agreement partitives, with the following exception:

Selectional properties: The COMPS list is empty.

Examples: half and *many*

4.3.5 Pronouns

The base lexical entries for pronouns are marked as inflected. Pronouns have a HEAD value of type *noun_star*, an empty MOD list, and have empty valence lists, except where noted.

Personal pronouns (@n_pers_pro_le)

Selectional properties: Empty valence lists.

Nonlocal feature properties: All NONLOCAL lists are empty.

Semantic properties: Their KEY value is of type *pron_rel*; LISZT contains both this *rel* and one of type *def_rel* (that is, personal pronouns are implicitly quantified).

Other properties: Personal pronouns have agreement features specified as gen:real_gender and prontype:std_pron. AGR is structure-shared with INDEX.

Examples: she and *them*

Singular *they* (@n_pers_pro_noagr_le) This is the lexical class of so-called "singular they," which occurs in tag questions co-indexed with syntactically singular generic pronouns like *everyone*, while showing a pattern of syntactically plural agreement itself. This lexical entry is just like other personal pronouns, with the following exceptions:

Other properties: AGR and INDEX values are not structure shared. The class itself has no AGR specifications, but the INDEX value is specified as gen:real_gender. Singular *they* has the AGR specification pn:thirdpl_star, and the INDEX specifications gen:andro_star, pn:thirdsg_star, and prontype:std_3.

Expletive pronouns look like personal pronouns, with the following exceptions:

Nonlocal feature properties: Empty NONLOCAL lists.

Semantic properties: KEY value of type *no_rel*, and an empty LISZT value. Thus, they make no semantic contribution to a phrase.

Other properties: Structure-share AGR and INDEX values. Their gender types are subtypes of *strict_gender*, but not *real_gender*.

Expletive it (@n_expl_it_le)

Other properties: An AGR value of type *it_ind*, with the specifications gen:no_gend_it and pn:thirdsg_star.

Expletive there (@n_expl_there_le)

Other properties: An AGR value of type *there_ind*, with the specification gen:no_gend_there.

Possessive pronouns (@n_poss_pro_le)

Selectional properties: These possessive pronouns have a nonempty SPEC list, the only member of which is an anti-synsem with empty NONLOCAL lists.

Nonlocal feature properties: Like personal pronouns, all *nonlocal* feature lists are empty.

Semantic properties: The semantic content is complex – in addition to the LISZT and KEY specifications described for personal pronouns, possessive pronouns have an ALTKEY value of type *basic_poss_rel*. This reln is included on LISZT along with a *def_np_rel*.

Examples: mine

Deictic pronouns (@n_deictic_pro_le)

Nonlocal feature properties: Like personal pronouns, all *nonlocal* lists are empty.

Semantic properties: Deictic pronouns have a KEY value of type *generic_nom_rel* and an ALTKEY value of type *demonstrative_rel*, both of which are included on LISZT. The index of the first is bound by the second (is its BV value). Semantically, then, the demonstrative acts as a quantifier binding a generic (implicit) nominal.

Examples: that and *these*

Generic pronouns (@n_generic_pro_le)

Head feature properties: Generic pronouns have HEAD value of type *noun*.

Selectional properties: They select an optional predicative adjectival complement, with *o_list*-valued COMPS and SPR lists, a non-empty SUBJ list, and empty *nonlocal* lists. They also select an optional specifier whose KEY value is of type *degree_rel*, and which has empty SLASH and REL lists.

Nonlocal feature properties: Their *nonlocal* lists are empty.

Semantic properties: Like deictic pronouns, their KEY value is of type *basic_nom_rel*, but their ALTKEY value is of type *quant_rel* (and specified further in the individual lexical entries). Both are included on LISZT, with the *quant_rel* binding the index of the *basic_nom_rel*.

Other properties: The agreement features are specified as gen:real_gender and pn:thirdsg_star. AGR and INDEX are structure-shared.

Examples: anyone and *everyone*

Reflexive pronouns (@n_refl_pro_le) Reflexive pronouns have the same specifications as personal pronouns with the following exceptions:

Head feature properties: CASE value is always *acc*.

Other properties: The (AGR feature) PRONTYPE value is *refl*.

Examples: *myself* and *herself*

***Wh*-pronouns (@n_wh_pro_le)**

Nonlocal feature properties: Wh-pronouns have a nonempty QUE list, but empty REL and SLASH lists.

Semantic properties: Their KEY value is of type *basic_nom_rel*, their ALTKEY value is of type *which_rel*, and both are included on LISZT with the second binding the index of the first.

Other properties: Otherwise, they look like personal pronouns in terms of syntactic specifications, with the additional agreement feature specification that the value of PN is always *thirdsg_star*.

Examples: *what* and *who*

Free relative pronouns (@n_freerel_pro_le)

Nonlocal feature properties: Free relative pronouns have a nonempty QUE list, and the single element on that list is the value of the pronouns INDEX feature. Furthermore, free relative pronouns have nonempty SLASH lists, containing a *local* object specified as a nonpredicative noun with an empty MOD list, empty SUBJ and COMPS lists, and the agreement feature specification pn:thirdsg. The REL list of a free relative pronoun is empty.

Semantic properties: Their KEY value is of type *instance_rel*, their ALTKEY value is of type *free_relative_ever_rel*, and both are included on LISZT with the first binding the index of the second.

Other properties: Otherwise, they look like wh-pronouns in terms of syntactic specifications, with the additional agreement feature specification that the value of PN is always *thirdsg*.

Examples: *what* and *whatever*

Relative pronouns (@n_rel_pro_le)

Nonlocal feature properties: In terms of syntactic specifications, relative pronouns look like personal pronouns, except that they have a nonempty REL list.

Semantic properties: Their KEY value, which is structure-shared with TOPKEY, is of type *reg_nom_rel*, but LISZT is empty.

Examples: *who* and *which*

Non-*wh* relative pronoun *that* (@n_rel_pro_nonwh_le)

Semantic properties: The non-wh relative pronoun looks like other relative pronouns, except that the KEY value is of type *modable_nom_rel*.

4.3.6 Adverbial nouns

***there* (@n_adv_le)**

Selectional properties: This noun looks like a pronoun, but has a nonempty SPR list, whose single member is an *anti_synsem* with empty *nonlocal* feature values.

Nonlocal feature properties: Empty *nonlocal* lists.

Semantic properties: Their KEY value is of type *basic_nom_rel*, their ALTKEY value is of type *def_explicit_rel*, and both appear on LISZT, with the second binding the index of the first.

Other properties: Specified as agr:pn:thirdsg_star

Adverbial *wh*-nouns (@n_wh_adv_le)

Head feature properties: Specified as case:acc.

Selectional properties: Adverbial *wh*-pronouns look like *wh*-pronouns, except that they select an optional complement with HEAD value *wh_adv* and KEY value *wh_the_hell_rel*, and their SPR list specifies an *anti_synsem* with empty *nonlocal* lists.

Examples: when

Adverbial free relative pronouns (@n_freerel_pro_adv_le) Adverbial free relative pronouns look like free relative pronouns, with the following exceptions:

Nonlocal feature properties: The SLASH value specifications differ from other free relative pronouns. Here, the *local* object is specified as having HEAD value of type *prep*, and a nonempty MOD list whose member has an INDEX value identified with another argument position of the *prep_mod_rel*.

Semantic properties: LISZT contains an additional reln of type *prep_mod_rel* which takes the index of the *instance_rel* as an argument. ALTKEY value is the same as the KEY value.

Examples: whenever and *how*

4.3.7 Determiners (@basic_det_synsem)

Head feature properties: As a class, determiners have the HEAD value *det*, but specific lexical entries are specified with the subtypes *non_part_det* or *part_det*. We introduced this distinction, not present in the ERG, in place of the types *part_det_synsem* and

non_part_det_synsem, which we have implemented as types. This distinction is used to determine which determiners are eligible as input to the partitive lexical rules, which derive partitive nouns (taking NP or PP complements) from partitive determiners. Determiners have an empty MOD value, and are specified as non-mobile.

Selectional properties: Determiners have empty SUBJ and COMPS lists, but select an optional specifier, which has empty QUE and REL values and a KEY value of type *degree_rel*. All determiners have a non-empty SPEC value, used to select the kind of thing for which a particular class of determiner may act as specifier. In every case, the index of the SPEC value is bound by the KEY of the determiner, via the BV feature.

Semantic properties: KEY value is of type *quant_or_wh_rel*.

Other properties: Marked as lex:plus.

Non-partitive determiners

(Ordinary) determiners (@det_le) Except where noted, the other non-partitive (non-possessive) determiners share these specifications, but have more specific selectional requirements.

Head feature properties: Marked as poss:minus.

Selectional properties: The SPEC value is of type *canonical_synsem*.

Nonlocal feature properties: Empty *nonlocal* feature values.

Semantic properties: ALTKEY value of type *no_rel*.

Other properties: Marked as inflected:plus. Unlike other non-partitive determiners, the value of the feature MODIFIED inherited from SPEC.

Examples: the and *no*

Singular mass determiners (@det_sm_le)

Selectional properties: SPEC specifies something whose value of the index feature PN is specified as *thirdsg_star*, that is, singular mass or count nouns.

Semantic properties: ALTKEY is unspecified.

Other properties: Agreement features specify pn:thirdsg. Value of the feature MODIFIED is *hasmod*.

Examples: this, *that*

Singular determiners (@det_sg_le)

Selectional properties: SPEC specifies something whose INDEX value is specified as divisible:minus_star and pn:thirdsg_star, that is, a singular count noun.

Other properties: Agreement features specify divisible:minus and pn:thirdsg. Value of the feature MODIFIED is *hasmod*.

Examples: every

Singular no-modifier determiners (@det_sg_nomod_le)

Selectional properties: Same SPEC specifications as a singular determiner. SPR value is an *anti-synsem*.

Other properties: Same AGR specifications as a singular determiner. Value of the feature MODIFIED is *notmod*.

Examples: a

Plural determiners (@det_pl_le)

Selectional properties: SPEC specifies something whose value of the index feature PN is specified as *thirdpl_star*, that is, plural nouns.

Other properties: Agreement features specify pn:thirdpl. Value of the feature MODIFIED is *notmod*.

Examples: these, *those*

Possessive determiners (@det_poss_le) This class differs from other non-partitive determiners in a number of respects.

Head feature properties: Marked as head:poss:plus.

Nonlocal feature properties: Empty *nonlocal* feature values.

Semantic properties: A complex semantic content, contributing the semantics of a personal pronoun (the possessor), and information about the possessor-possessed relationship via its LISZT specifications. The KEY value is of type *abstr_def_rel*. The ALTKEY value is of type *pro_poss_rel*; this REL has both the index of the SPEC value (the possessed) and the INST value of the determiner's own pronominal content as argument values.

Other properties: Specified as inflected in the lexicon.

Examples: their, *its*

Partitive determiners

(Ordinary) partitive determiners (@det_part_le) Similar to ordinary non-partitive determiners @det_le, with the following exceptions (Other partitive determiners are further specified, as described below.):

Head feature properties: HEAD value of type *part_det*. Unspecified POSS value.

Selectional properties: The SPEC list member is of type *synsem_min*; its ALTKEY value must be of type *basic_nom_rel*.

Semantic properties: ALTKEY value type is unspecified.

Examples: some, *either*, *any*

Singular partitive determiners (@det_part_sg_le) have the AGR and SPEC specifications of non-partitive singular determiners.

Examples: each, *neither*

Plural partitive determiners (@det_part_pl_le) have the AGR and SPEC specifications of non-partitive plural determiners.

Examples: many, *several*, *both*

Plural-mass partitive determiners (@det_part_pl_mass_le)

Selectional properties: SPEC list specifies something for which the index is specified as divisible:plus_star, that is, mass or plural nouns.

Other properties: Agreement features are specified as divisible:plus.

Examples: most, *all*, *more*

The determiner one (@det_part_one_le) has the selectional and agreement properties of the other singular partitive determiners, but its semantic content is different.

Semantic properties: KEY value is of type *udef_rel*; also on LISZT is a REL of type *card_rel* whose CONST_REL argument has the (string-) value *one*.

Wh-determiners

The determiner what (@det_wh_le) This *wh*-determiner has the specifications of @det_le, with the following exceptions:

Selectional properties: The SPEC value is of type *synsem_min*, and it has a non-empty QUE value, specified in the lexical entry itself.

Nonlocal feature properties: QUE value is non-empty.

The determiner which (@det_part_unsp_le) The unspecified partitive determiner has the specifications of @det_part_le, with the following exceptions:

Head feature properties: Specified as poss:minus.

Selectional properties: It lacks the requirement of other partitives that it specify something whose ALTKEY value is of type *basic_nom_rel*.

Nonlocal feature properties: QUE value is non-empty.

The determiner how_many (@det_part_pl_wh_le_many) The plural partitive *wh*-determiner has the specifications of @det_part_pl_le, with the following exceptions:

Head feature properties: Specified as poss:minus.

Selectional properties: It lacks the requirement of other partitives that it specify something whose ALTKEY value is of type *basic_nom_rel*.

Nonlocal feature properties: QUE value is non-empty.

Semantic properties: Semantic content is more complex.

The determiner whichever (@n_freerel_part_le) The free-relative partitive noun is a misnomer, since its HEAD value is of type *part_det*. Its specifications are similar to @det_part_unsp_le, with the following exceptions:

Nonlocal feature properties: It has a non-empty SLASH value (in addition to the non-empty QUE value), specified as having a HEAD value of type *n_or_p*.

Semantic properties: KEY value is of type *free_relative_ever_rel*.

The free-relative determiner what (@det_freerel_le) has specifications similar to the macro @det_wh_le, with the following exceptions:

Nonlocal feature properties: It has a non-empty SLASH value (in addition to the non-empty QUE value), specified as having a HEAD value of type *n_or_modnp* and accusative case.

Semantic properties: Its KEY value is of type *free_relative_ever_rel*.

The possessive relative determiner whose (@det_rel_poss_le) has specifications similar to the possessive determiners (@det_poss_le), with the following exceptions:

Nonlocal feature properties: It has a non-empty REL value.

Semantic properties: It lacks the personal pronoun content in its LISZT value.

4.3.8 Prepositions

Prepositions have the HEAD value *prep_star*. They are specified as lex:plus_star, inflected:plus, and posthead:plus.

Regular prepositions (@p_reg_le)

Head feature properties: Regular prepositions have a MOD value which specifies that they modify something with a LOCAL value of type *intersective_mod*, a HEAD value of type *n_or_v*, strict_head:plus, with an *o_list*-valued COMPS value, and a non-empty SPR value, of type *synsem_min*, and a KEY value of type *quant_or_deg_rel*.

Selectional properties: Regular prepositions select an *nominal* subject, specified as poss:minus, strict_head:plus, mood:ind_or_mod_subj, and mc:na_or_minus. The subject must have *o_list*-valued SUBJ and SPR lists, an empty COMPS list, and empty *nonlocal* feature values.

They select an optional *adv* specifier, with *o_list*-valued SPR and COMPS lists, an empty QUE list, and a KEY value of type *very_deg_rel*.

They select a non-optional accusative *nominal* complement, specified as mobile:plus, poss:minus, strict_head:plus, mood:ind_or_mod_subj, and mc:na_or_minus. The subject must have *o_list*-valued SUBJ and SPR lists, an empty COMPS list, and empty *nonlocal* feature values. This complement must have a KEY value of type *non_temp_nom_rel*, which is structure-shared with the COMPKEY value of the preposition.

Semantic properties: The KEY value of a regular preposition is of type *basic_prep_mod_rel*.

Examples: on, *to*

(Ordinary) prepositions (@p_le) Prepositions in this class are identical to regular prepositions, except there is no restriction on the KEY value of the (non-optional) complement.

Examples: for, *with*, *before*

No-specifier prepositions (@p_nospec_le) are similar to @p_le, with the following exceptions:

Head feature properties: MOD list specifies something with a with a KEY value of type *never_unify_rel*. This effectively prohibits a PP headed by such a preposition from functioning as an adjunct. Specified as aspect:noasp_and_progr and tense:no_tense.

Selectional properties: The SPR list member is of type *unexpressed*.

Other properties: Values of the agreement features DIVISIBLE and PN are structure-shared with those of the complement.

Examples: of

No-specifier no-gap prepositions (@p_nospec_nogap_le) look like no-specifier prepositions (@p_nospec_le) with the following exceptions:

Head feature properties: No restriction on KEY value of the MOD list item (thus, these prepositions can act as modifiers).

Selectional properties: Complement must have an empty SLASH list.

Other properties: No agreement feature structure-sharing with complement.

Examples: but

Idiomatic no-modifier prepositions (@p_idiom_nomod_le)

Head feature properties: Empty MOD list. Specified as prd:minus.

Selectional properties: The same SPR and COMPS specifications as @p_le. Empty SUBJ list (a nonempty list is necessary only for prepositions which can be predicative).

Semantic properties: KEY value is of type *prep_rel*.

Examples: (passive) *by*

Comparative *than* (@p_compar_than_le)

Head feature properties: Specified as prd:minus. MOD list specifies something with HEAD value of type *adj*, with a SPR specification whose KEY value is of type *much_deg_rel*, and an ALTKEY value of type *comp_rel*.

Selectional properties: Selects a *nominal* subject specified as poss:minus, strict_head:plus, mood:ind_or_mod_subj, mc:na_or_minus, and conj:cnil. The subject must have *o_list*-valued SPR and SUBJ lists, and an empty COMPS list. It selects a non-optional *noun* complement, with *o_list*-valued SPR and SUBJ lists, an empty COMPS list, and empty *nonlocal* feature lists. It has an empty SPR list.

Nonlocal feature properties: Empty *nonlocal* feature lists.

Semantic properties: KEY value of type *x_compar_than_rel*.

Temporal prepositions (@p_temp_le) look like @p_le with the following exceptions:

Selectional properties: The nominal complement has unspecified MOBILE MESSAGE, and MOOD feature values. The complement is specified as mc:na and comps:o_list.

Examples: at, *on*

The preposition *a* (@p_nbar_comp_nmod_le) In ERG, the *a* in "three times a day" is analyzed as a preposition. This preposition is similar to @p_le, with the following exceptions:

Selectional properties: Most importantly, the complement is lexical, a *thirdsg* noun with an *o_list*-valued COMPS list, an *unexpressed* specifier, an empty SLASH list, and an ALTKEY value of type *no_rel*. This complement is specified as modified:notmod, periph:na_or_minus. The SUBJ list specifies an *anti_synsem*. The specifier must have an empty SLASH list.

No-noun modifying prepositions (@p_no_n_mod_le) are identical to @p_le with the following exceptions:

Head feature properties: MOD list specifies something with a non-empty SUBJ list. This rules out nouns, which always have empty SUBJ lists.

Examples: until

Subordinating conjunctions (@p_subconj_le)

Head feature properties: HEAD value is of type *prep*. Specified as prd:minus. MOD list specifies something with a *local* value of type *scopal_mod*, a HEAD value of type *verb*, specified as mobile:minus, strict_head:plus, with *o_list*-valued COMPS and SUBJ lists, a singleton MESSAGE list, and an empty SLASH list.

Selectional properties: Empty SUBJ and SPR lists. COMPS list specifies a non-optional finite clause: a HEAD value of type *verb*, specified as non-inverted, finite, mc:plus, with *o_list*-valued SUBJ and COMPS lists, a MESSAGE list containing a *prpstn_rel*, and empty *nonlocal* feature values

Nonlocal feature properties: All *nonlocal* lists are empty.

Semantic properties: The KEY value is of type *subord_rel*, the MESSAGE list contains a *prpstn_rel*. Both of these *rel*s are members of LISZT.

Other properties: The MC value of the preposition is structure-shared with that of the MOD value.

Examples: although, *when*, *as_soon_as*

Predicative subordinating conjunctions (@p_subconj_prd_le) look like subordinating conjunctions (@p_subconj_le), with the following exceptions in terms of selectional and modifying properties.

Head feature properties: Additional specifications in MOD: a VFORM value of type *fin_or_imp*, mood:ind_or_mod_subj, and mc:plus. Thus, these prepositions modify finite or imperative clauses. The AGR feature value specified in MOD is structure-shared with the INDEX value of the complement's *unexpressed* subject.

Selectional properties: Non-optional predicative complement, specified as strict_head:plus and tense:no_tense. This complement must have *o_list*-valued SPR and COMPS lists, a non-empty SUBJ whose member is of type *unexpressed_reg*, and empty *nonlocal* feature lists.

Semantic properties: LISZT contains an additional element of type *hypo_rel*.

Examples: when

Infinitive subordinating conjunctions (@p_subconj_inf_le) look like predicative subordinating conjunctions, but their selectional properties are different. PPs headed by these "prepositions" look like infinitive VPs; these are what are often called *rationale infinitives*, or *in order* clauses, in the literature.

Selectional properties: Complement has a HEAD value of type *verb*, specified as non-inverted, non-mobile, non-predicative, indicative, and having the VFORM specification *bse_only*. The SUBJ and SPR list of this complement are non-empty, specifying something of type *unexpressed*, and the MESSAGE list must be empty.

Semantic properties: LISZT contains two additional elements, one of type *def_rel*, the other of type *pron_rel*, representing the subject of the complement.

Examples: in_order_to, *to*

Indicative *if* (@p_subconj_if_indic_le) differs from predicative subordinating conjunctions (@p_subconj_prd_le) primarily in its selectional properties.

Selectional properties: It selects three complements. The first is an optional complement, with a HEAD value of type *prep_or_modnp*, a non-empty MOD value specifying something with a LOCAL value of type *intersective_mod*, and an empty COMPS list. The second (non-optional) complement is a nominative *nominal* with *o_list*-valued SUBJ and SPR lists, an empty COMPS list, and an empty SLASH list. The third (non-optional) complement is *verbal*, finite, with an *o_list*-valued COMPS list, a non-empty SUBJ list, and an empty MESSAGE list. The AGR value of the NP complement is structure-shared with the AGR value of the subject of the VP complement, enforcing agreement between the two. Furthermore, the MOOD specification of the VP complement is structure-shared with the MOOD of the MOD value.

CP-complement prepositions (@p_cp_le) look like @p_le, with the following exceptions:

Selectional properties: Select a complement with a HEAD value of type *verb*, specified as non-inverted, non-mobile, finite, mood:strict_ind_or_mod_subj, tense:real_tense, and mc:minus, with an *o_list*-valued COMPS list and a MESSAGE list containing a *prpstn_rel*.

Semantic properties: The value of the ARG3 position in the KEY value is the (event-)index of the CP.

Examples: until, after

PrdP-complement prepositions (@p_prdp_le) These prepositions look like @p_le, except for the COMPS specifications.

Selectional properties: The HEAD value of the complement is left unspecified, but it must be specified as prd:plus and tense:no_tense, with an *o_list*-valued COMPS list, a non-empty SUBJ list, and empty *nonlocal* feature lists.

Examples: as

Phrasal prepositions are specified as lex:minus (except where noted), have empty COMPS lists, and empty *nonlocal* lists except where noted.

Ordinary phrasal prepositions (@pp_le) look like regular prepositions (@p_reg_le) with the exceptions noted above and below. This is the lexical class of particles.

Head feature properties: MOD list specifies something which is not a clause, that is, specified as mc:na and message:[], and is specified as modified:notmod_or_rmod. Otherwise the MOD specification looks like that of a regular preposition.

Semantic properties: KEY value is of type *glbtype1036*.

Examples: off

Relative prepositions (@pp_rel_le) differ from ordinary phrasal prepositions in the following respects.

Head feature properties: Specified as prd:plus. MOD list specifies a VP: something with a HEAD value of type *verb* which is non-inverted and non-mobile, with non-empty SUBJ and SPR lists. The specifier must have a KEY value of type *quant_or_deg_rel*.

Selectional properties: They select an (optional) specifier with a KEY value of type *degree_rel*.

Nonlocal feature properties: Non-empty REL value.

Semantic properties: KEY value is of type *glbtype910*.

Examples: where, when

***Wh*-prepositions (@pp_wh_le)** differ from ordinary phrasal prepositions in the following respects.

Head feature properties: MOD specifies something with a HEAD value of type *verb_or_gerund*.

Nonlocal feature properties: Non-empty QUE value.

Semantic properties: KEY value is of type *temp_abstr_rel*, ALTKEY value is of type *nom_rel*, and LISZT also contains a *which_rel*.

Other properties: Specified as lex:plus_star.

Examples: how

4.3.9 Adjectives (@basic_adj_synsem_lex_or_phrase)

Head feature properties: Adjectives have a HEAD value of type *adj_star*.

Selectional properties: They select an optional specifier with a KEY value of type *degree_rel*, an empty SLASH list, and an *o_list*-valued SPR list. They also select a *nominal* subject, specified as poss:minus, strict_head:plus, and tam:ind_or_mod_subj. The subject must have *o_list*-valued SUBJ and SPR lists, and an empty COMPS list.

Semantic properties: KEY value of type *basic_adj_rel*.

Other properties: Adjectives structure-share their AGR value with the AGR and INDEX values of their subject, and are specified as stemhead:astem.

4.3.9.1 Modifying adjectives

Intransitive adjectives (@adj_intrans_le)

Head feature properties: MOD specifies something with a HEAD value of type *noun*, specified as poss:minus and strict_head:plus, with empty SUBJ and COMPS lists, but a non-empty SPR list, containing a *synsem* with a KEY value of type *quant_or_deg_rel*.

Selectional properties: Empty COMPS list.

Semantic properties: KEY value of type *abstr_adj_rel*, and an empty MESSAGE list. INDEX value is structure-shared with the value of AGR, as well as the INDEX values of both the subject and the thing modified.

Other properties: Specified as conj:cnil and posthead:minus. Base lexical entries are marked lex:plus_star and inflected:minus.

Examples: old, *trustworthy*

Comparative adjectives (@adj_comp_le) look like intransitive adjectives with the following exceptions:

Selectional properties: The specifier must have a KEY value of type *much_deg_rel*.

Semantic properties: ALTKEY value is of type *comp_rel*, and included on LISZT.

Other properties: Base lexical entries are marked as inflected:plus.

Examples: older, *harder*

Superlative adjectives (@adj_superl_le) look like intransitive adjectives with the following exceptions:

Head feature properties: Non-predicative (prd:minus).

Selectional properties: The specifier must have a KEY value of type *very_deg_rel*.

Semantic properties: ALTKEY value is of type *superl_rel*, and is included on LISZT.

Other properties: Base lexical entries are marked as inflected:plus.

Examples: oldest

The adjectives *more* and *less* (@adj_more_less_le) look like comparative adjectives with the following exceptions:

Selectional properties: Both the subject and specifier are unexpressed (and opt:plus). COMPS list specifies a non-optional lexical adjectival complement, with an empty COMPS list, an *o_list*-valued SPR list, and a non-empty SUBJ specification which is structure-shared with the adjective's own SUBJ list.

Semantic properties: The KEY value is of type *more_less_adj_rel*.

The adjectives *most* and *least* (@adj_most_least_le) look like @adj_more_less_le with the following exceptions:

Selectional properties: The KEY value of the (unexpressed) specifier is of type *very_deg_rel*.

Semantic properties: The KEY value is of type *most_least_adj_rel*, and the ALTKEY value is of type *super_rel*.

Ordinal adjectives (@adj_bare_unspecified_ord_le)

Head feature properties: HEAD value of is type *intadj*, specified as ordinal:plus and prd:minus. MOD list specifies something with a HEAD value of type *noun*, specified as divisible:plus_star (that is, a mass or plural noun), having empty SUBJ and COMPS lists, and a non-empty SPR value, specifying something with a KEY value of type *quant_or_deg_rel*. The KEY value of the modified thing must be of type *nonpro_rel*.

Selectional properties: Empty SUBJ and COMPS lists, selecting only an optional specifier with a HEAD value of type *adv*, *o_list*-valued COMPS and SPR lists, and a KEY value of type *degree_rel*. SPEC is also non-empty.

Nonlocal feature properties: Empty *nonlocal* feature values.

Semantic properties: KEY value is of type *ord_rel*, and is structure-shared with ALTKEY.

Other properties: Base lexical entries are marked as lex:plus_star and inflected:plus.

Examples: first

Cardinal adjectives (@adj_bare_unspecified_card_le) differ from ordinal adjectives only in that they are specified as ordinal:minus, and their KEY value is of type *card_rel*.

Examples: two, thirty

4.3.9.2 Non-modifying adjectives

CP[that] adjectives (@adj_reg_atrans_that_cp_le)

Head feature properties: Specified as prd:plus. MOD list specifies something with a HEAD value of type *no_head*.

Selectional properties: SUBJ and SPR specifications like those of intransitive adjectives, except that the subject's INDEX value must be of type *it_ind*. COMPS list specifies a non-optional *verbal* complement, specified as inv:minus, mood:strict_ind_or_mod_subj, tense:real_tense, vform:fin_or_inf, mc:minus, having *o_list*-valued valence lists, and a non-empty MESSAGE list, specifying a *prpstn_rel*. The member of the COMPS list is also the value of the feature KEYCOMP.

Semantic properties: The KEY value is of type *adj_arg4_rel*. The complement's KEY value is structure-shared with COMPKEY.

Examples: true

***Wh*-adjectives (@adj_wh_le)**

Head feature properties: Empty MOD list.

Selectional properties: Empty SPR and COMPS lists. SUBJ list specifies a *nominal* subject like that of any adjective.

Nonlocal feature properties: QUE is non-empty; REL and SLASH are empty.

Semantic properties: KEY value is of type *prpstn_to_prop_rel*. Semantic content is complex. MESSAGE list is empty.

Other properties: Specified as posthead:plus and mc:na. Base lexical entry is specified as lex:plus_star and inflected:plus.

Examples: how

4.3.10 Degree specifiers (@adv_degree_spec_le)

Head feature properties: HEAD value is of type *adv*, and MOD list is empty.

Selectional properties: Empty SUBJ and COMPS lists. They select an optional specifier, also with a HEAD value of type *adv* and a KEY value of type *degree_rel*. This optional specifier also must have an *o_list*-valued SPR list, and an empty QUE list. SPEC list specifies something with an ALTKEY value of type *non_conj_rel*.

Nonlocal feature properties: *nonlocal* feature lists are empty.

Semantic properties: KEY value of type *degree_rel*. MESSAGE list is empty.

Other properties: Specified as mc:na, conj:cnil, and inflected:plus. The value of the MODIFIED feature is of type *xmod*, specified as periph:na.

Examples: *very*, *almost*, *only*

Titles (@n_title_le)

Head feature properties: HEAD value *adv*. MOD list specifies a *canonical_synsem* with a HEAD value of type *noun*, specified as poss:minus, strict_head:plus and modified:notmod_or_rmod, with empty SUBJ and COMPS lists, and a non-empty SPR list containing something with a KEY value of type *quant_or_deg_rel*. More importantly, the MOD value specifies something with a KEY value of type *abstr_named_rel*.

Selectional properties: Empty valence lists.

Nonlocal feature properties: Empty *nonlocal* feature lists.

Semantic properties: KEY value of type *instance_rel*. LISZT also contains a *udef_rel* and a *title_id_rel*.

Other properties: Specified as posthead:minus. The base lexical entries of titles are specified as lexical and inflected.

Examples: *mr*

Post titles (@n_post_title_le) differ from (pre-) titles only in their specification as post-head:plus, and hc_lex:minus_star.

Examples: *esq*

4.3.11 Conjunctions (@conj_word)

Conjunctions inherit a number of feature values from the first element on the COMPS list, as described below.

Head feature properties: HEAD feature value structure-shared with first complement.

Selectional properties: Conjunctions select one non-optional complement. The COMPS of that complement is the rest of the conjunction's COMPS list (that is, the complements are inherited). SPR and SUBJ are also inherited from the first complement.

Semantic properties: The values of KEY and MESSAGE are inherited from the first complement.

Other properties: Conjunctions are specified as inflected:plus. The values of the features LEX,DIVISIBLE, HC_LEX and MC are inherited from the first complement.

Complex conjunctions (@conj_complex_le)

Semantic properties: ALTKEY value is of type *conj_rel* and is the sole member of LISZT. The INDEX value is of type *conj_ind*. The value of H_CONS is the empty list.

Other properties: specified as conj:lex_conj.

Examples: and, *or*, *nor*, *but*

Atomic conjunctions (@conj_atomic_le)

Selectional properties: The first complement must be specified as conj:cnil.

Semantic properties: Both LISZT and H_CONS are empty.

Other properties: CONJ value is *atomic_conj*.

Examples: either, *neither*, *both*

4.4 Lexical Rules

4.4.1 Inflectional Lexical Rules

Singular nouns sing_noun_infl_rule keeps morphology constant. The output has a SPR value whose AGR feature value specifies pn:thirdsg and divisible:minus. It is specified as stemhead:countnstem, and its INDEX value specifies pn:thirdsg_star, gen:neut_star, and divisible:minus_star.

Mass nouns mass_noun_infl_rule keeps morphology constant. The output has a SPR value whose AGR feature value specifies pn:thirdsg and divisible:plus. It is specified as stemhead:massnstem, and its INDEX value specifies pn:thirdsg_star, gen:neut_star.

Plural nouns plur_noun_infl_rule adds an affix to the input phonology. The output has a SPR value whose AGR feature value specifies pn:thirdpl and divisible:plus. It is specified as stemhead:count_or_masscount_nstem, and its INDEX value specifies pn:thirdpl_star, and divisible:plus_star.

4.4.2 Derivational lexical rules

4.4.2.1 Partitive lexical rules

derive partitive nouns from so-called partitive determiners: those that have the HEAD value *part_det*, with a KEY value of type *quant_or_wh_rel*.

The partitive construction lexical rules specify that the KEY value of the input is the ALTKEY value of the output, that the derived lexical entry inherits the DIVISIBLE and NONLOCAL specifications of the input, and that the KEY value of the derived lexical entry is structure-shared with the head of C_CONT — LISZT (otherwise it would not appear on the

LISZT value of the derived partitive noun, given the way that LISZT feature amalgamation is formulated for lexical rules).

part_nocomp_constr This lexical rule derives partitive nouns with empty COMPS lists, licensing things like "many sleep." The KEY value is specified as *part_of_rel*. Since there are no agreement specifications on the noun, things like "many sleeps" are also licensed.

There are two lexical rules that derive partitive nouns selecting for a PP[of] complement. Both specify that the KEYCOMP value is structure shared with the only element on the COMPS list. The KEY value is of type *part_of_rel*, and it has an argument whose value is the index which is also the value of the complement's KEY value (that is, the index of the NP complement of the PP[of]).

part_ppof_agr_constr This lexical rule derives partitive nouns whose agreement feature specifications are determined by the NP complement of *of*. Specifically, its PN and DIVISIBLE specifications are structure-shared with those of the KEYCOMP value.

The input to this lexical rule must have a KEY value of type *explicit_quant_agr_rel*. It structure-shares its DIVISIBLE specification with the input's KEYCOMP. . . DIVISIBLE specification, and structure-shares its PN specification with its own KEYCOMP . . . PN specification.

part_ppof_noagr_constr This lexical rule derives partitive nouns whose agreement features are inherited wholesale from the input, whose feature KEY must have the value *explicit_quant_or_udef_noagr_rel*.

partitive_num was originally a non-head-only phrasal type, but we changed it to a lexical rule.

4.4.2.2 month_det

This lexical rule takes nominal month lexical entries (as in "the first of january") and derives determiners that function as specifiers for the day-of-month in phrases like "january first".

The input to this lexical rule has a KEY value of type *mofy_rel* (referring to month-of-year), selects for a single optional complement, and is specified as stemhead:nstem.

The derived lexical entry inherits its COMPKEY, OCOMPKEY, and NONLOCAL values from the input. The KEY value of the input is the ALTKEY value of the output. The KEY value of the derived lexical entry must be of type *def_rel* and is included in the semantics of the lexical entry via C_CONT (as are two additional *s*).

The derived lexical entry has a HEAD value of type *det*, and empty valence lists except for SPEC, the member of that list has a KEY value of type *dofm_rel* (day-of-month) and an INDEX value structure-shared with its own.

4.4.2.3 dofm_yofc

The input of this lexical rule is specified by the macro @dom_ord_synsem, but the only specifications shared between input and output are those dictated by a general description of lexical rules, and constraints on lexical rule types. The input to this lexical rule, then, should contribute a single *rel* of type *dofm_rel* to the semantics of the output; the only lexical

```
headed_phrase *>
   (root:minus,
    synsem:local:( cat:( head:Head,
                         hc_lex:Hclex),
                   agr:Agr,
                  conj:Conj,
                  keys:key:Key ),
    head_dtr:synsem:local:(cat: (head:Head,
                                 hc_lex:Hclex ),
                           agr:Agr,
                          conj:Conj,
                          keys:key:Key ) ).
```

Figure 4.19: The Head Feature Principle

entries meeting this condition are the nominal day-of-month lexical entries, such as "first". These base lexical entries select an optional PP[of] complement that specifies month, and optionally year, as in "the first of january 1984."

This lexical rule derives day-of-month lexical entries that select a non-optional month-of-year specifier, and a non-optional year-of-century complement, as in "january first 1984."

The derived lexical entry has a HEAD value *noun*, and is specified as prd:plus. Its specifier has a HEAD value *det*, an ALTKEY value of type *mofy_rel*, and a KEY value of type *abstr_def_rel*. The lexical entries derived via the month_det lexical rule satisfy this description. Its single complement has a HEAD value of type *nominal* and a KEY value of type *yofc_rel*.

4.5 General Principles

4.5.1 The Head Feature Principle

The Head Feature Principle (HFP) is located on a constraint on the type *headed_phrase*, as shown in figure 4.19. Basically, any kind of mother with any kind of daughter(s) may be licensed in the current incarnation of the MERGE grammar.[1] This is true of any phrase not a subtype of *headed_phrase*.

4.5.2 Slash Amalgamation

The principles necessary for the lexical amalgamation of the NONLOCAL features are discussed in section 2.1.1.2.

[1]This aspect of the grammar is inherited from the ERG grammar, and will ideally not be maintained in future MILCA grammars (except possibly in the case of coordination), as it runs counter to a fundamental idea of HPSG.

```
head_nexus_rel_phrase macro (headed_phrase,
    synsem:nonlocal:rel:Rel,
    head_dtr:synsem:nonlocal:rel:Rel).


head_nexus_phrase macro (@head_nexus_rel_phrase,
@head_nexus_que_phrase).

head_valence_phrase macro (@head_nexus_phrase,
    synsem:nonlocal:slash:Slash,
    head_dtr:synsem:nonlocal:slash:Slash).
```

Figure 4.20: Nonlocal Feature Inheritance

4.5.3 Nonlocal Feature Inheritance

The requirement that a SLASH value be shared between mother and head daughter is located in the macro @head_valence_phrase. Any phrase not specified as satisfying this description does not inherit a SLASH value from its head daughter unless the schema licensing it explicitly specifies such structure sharing. Similarly, the requirement that a *v*alue be shared between mother and head daughter is located in the macro @head_nexus_rel_phrase. Since @head_nexus_rel_phrase is a "super-macro" of @head_valence_phrase, any phrase specified as satisfying the latter must also satisfy the former. The MERGE does not require any special mechanism (such as TO_BIND) to "cancel" something off a SLASH list; instead, the relevant phrase must simply not be specified as satisfying the description @head_valence_phrase, and the nonlocal feature values on mother and daughter(s) should be specified explicitly.

Chapter 5

Phenomena and how they are licensed

5.1 The nominal domain

5.1.1 Simple noun phrases

Noun phrases consisting of a determiner and a noun (or N') are instances of head-specifier phrases, wherein the determiner and the noun co-select each other via the SPEC and SPR features.

Both daughters in the head-specifier phrase must be inflected. While the base lexical entries of determiners are marked as such, the base lexical entries of (most) nouns are not. This means only derived lexical entries, the output of the singular-, mass-, or plural- noun lexical rules, are eligible to combine with a determiner to form an NP. Base lexical entries which are specified as plural, such as *staff*, are an exception to this, since these lexical entries do not undergo inflection by lexical rule, and are marked as inflected in the lexicon. However, since inflectional lexical rules only take lex:minus input, this setup necessitates the inclusion of a second uninflected base lexical entry for *staff* in order to license the plural form *staffs*.

Furthermore, the head-specifier phrase specifications require that the head daughter have an *o_list*-valued COMPS list, meaning that the noun must have already combined with its complement, or not at all. Nothing prevents the noun from having undergone modifications. (In these cases, the determiner combines a so-called N', not a noun.)

The resulting NP inherits its AGR, HEAD, *val* feature, KEY and ALTKEY values from the head daughter, the N'.

In terms of the specifics of co-selection between the determiner and the noun, nouns require that their specifier be a determiner with saturated valence lists, and that its AGR specifications be related to its own, though not token-identical. That is, the noun's agreement features will have values which allow coordination (and unify with other values), and these specifications are also those of the NP. Not so the SPR specifications, since this would allow unification with any value. For example, the (derived) lexical entry for *dog* is specified (under AGR) as divisible:minus_star, and pn:thirdsg_star, but the SPR value specifies divisible:minus,

and pn:thirdsg. The distribution of *the* is not restricted with respect to agreement, but other determiners do have restricted distributions.

For example, *a* is specified as pn:thirdsg, allowing it to occur with singular count and mass nouns, but not plural nouns. *These* is specified as pn:thirdpl, allowing it to occur with plural nouns, but not singular count or mass nouns. *All* is specified as divisible:plus, allowing it to occur with mass and plural nouns, but not singular count nouns.

5.1.2 Prenominal modification

Prenominal modification is always an instance of adj_n_int_phrase. These phrases are headed by the nominal daughter, but modifiers select the thing they modify via the MOD feature. Again, both daughters must be inflected; in the case of the adjective, this means being the output of the positive-adjective lexical rule, unless the base lexical entry is a comparative or superlative form, which are marked as inflected in the lexicon. I assume this rule is included because originally comparative- and superlative- lexical rules were also meant to be included; this is not now the case.

In any case, prenominal modifiers only modify nouns that have already combined with their complements, but have a non-empty SPR list. This rules out the possibility of having the head daughter of an adj_n_int_phrase be an NP, as in **good [the dog]*.

In order to pre-modify, the adjective must be specified as prd:minus and posthead:minus. These requirements rule out pre-modification by (most[1]) adjectives which have combined with complements, or which are post-modified. For example, there is a lexical entry for *true* which selects a clausal complement, and is specified as prd:plus, which rules out **the true that dogs sleep belief*. On the other hand, *older* is specified as prd:minus and posthead:minus, allowing *an older dog*, but the hadj_i_h rule, which licenses post-modification like *older than that dog* specifies that the mother be marked as posthead:plus, which then rules out **an [older than that dog] dog*.

Numerals, both cardinal and ordinal, are adjectives in MERGE. Thus, *the first dogs* and *the five dogs* are instances of prenominal modification. MERGE blocks recursive modification by cardinal and ordinal numbers, however, by use of the MODIFIED feature. In an adjn_i construction (adjective-noun intersective), the MODIFIED value of the mother is inherited from the non-head daughter. Since *five* and *first* are specified as modified:periph:plus, then, so are *five dogs* and *first dogs*. Furthermore, both *five* and *first* specify in their MOD feature values that they will only modify something specified as modified:periph:na. This rules out ungrammatical examples like **the five five dogs*, but also grammatical examples like *the first first dogs* and *the first five dogs*.

(40) a. the first good dogs
b. the good first dogs
c. the five good dogs
d. the good five dogs
e. the good good dogs
f. the good dog

[1] *More* and *most* are exceptions to this generalization.

g. * the five dog

Since most adjectives, like *good*, are specified as modified:periph:na, and leave the MODIFIED feature unspecified in their MOD value, all of the grammatical examples in (40) are licensed. Cardinal number adjectives are required to modify things where the agreement feature PN is specified as *thirdpl*, which rules out examples like (40g).

5.1.3 Degree specifiers of determiners, adjectives, and other specifiers

Determiners, adjectives and specifiers all select optional adverbial degree specifiers via the SPR list, to license things like *[only the]_DetP older dog*, *the [very old]_AdjP dog*, and *the [[very very]_AdvP old] dog*. These are all head-specifier constructions, and the particulars of selection are primarily semantic, via the KEY feature.

For example, *the* requires only that its specifier have a KEY value of type *just_only_degree_rel*, while *old* requires its specifier to have a KEY value of type *very_degree_rel*, and *very* requires that its specifier have a HEAD value of type *adv* and a KEY value of type degree_rel.

5.1.4 Possessives

MERGE covers the possessive constructions in (41).

(41) a. her dog
b. Kim's dog
c. the manager's dog
d. the manager who hired me's dog
e. * me's dog

Each of the possessives above is either a determiner lexical item, or a determiner phrase. Possessive pronouns are non-partitive determiners with specific pronominal person-number-gender information. The possessive morpheme *'s* (and its orthographic variant *'*) selects a non-optional specifier that is either a noun or an NP, forming the possessive determiner phrase. This phrase has the same selective properties as other determiners.

5.1.5 Special kinds of noun phrases

5.1.5.1 Pronominal expressions

Referring pronominals The MERGE lexicon includes both personal and reflexive pronouns, covers their distribution with respect to case, but since the grammar does not include an implementation of any binding theory, the distribution of personal vs. reflexive pronouns in English is not covered. This is reflected in the examples in (42) which are licensed or ruled out by the grammar.

(42) a. She sleeps.
b. They sleep.

c. * Her sleeps.
d. Kim visited her.
e. * Kim visited she.
f. Kim showed the office to her.
g. * Kim showed the office to she.
h. She visited herself.
i. * He visited herself.
j. * Herself sleeps.

Expletives To license sentences with the expletive *there*, such as (43), a special lexical entry for the verb *be* is used. This lexical entry has two elements on its COMPS list, an accusative NP and some kind of phrase.

(43) There is a bookcase in the office.

One would think these two complements exist in order two license the NP *a bookcase* and the PP *in the office* in (43) as the two complements of the verb *is*. But instead, only the NP is realized as a complement, the PP is realized as an adjunct with the help of the hadj_i_uns_aux rule. The second complement remains as an unexpressed synsem on the COMPS list of *is*. There does not seem to be any construction where the optional complement of this kind of *be* is used.

5.1.5.2 Names

All of the name expressions in (44) are licensed in MERGE.

(44) a. Kim
b. Browne
c. Mr. Browne
d. Kim Browne

All name expressions are licensed (at the top level) by the unary proper_np rule. First and last names are syntactically identical (though certain first names have gender specifications included in their lexical entries). First + last name combinations, as in (44d), are just instances of np-noun compounds, but title + last name combinations, as in (44c) involve prenominal modification by the title. Thus, *Mr*, though it has a HEAD value of type *adv*, has a MOD value specifying that it must modify an N' with a KEY value of type *abstr_named_rel*.

5.1.5.3 Date expressions

MERGE licenses a variety of date expressions; the variation illustrated in (45) is accomplished by a combination of lexical variation (both in terms of base and derived lexical entries) and optional complementation. However, the grammar does rule out ungrammatical examples like (45h).

(45) a. the first of January (of) 1984

b. the first of January
c. the first
d. January first 1984
e. January first
f. January 1984
g. January of 1984
h. * first 1984
i. 1/3/1984

The lexical head of the NPs in examples (45a)-(45c) is a the base lexical entry of *first*, which has a HEAD value of type *dofm_rel*. This lexical entry selects an ordinary specifier and an optional PP[of] complement, whose COMPKEY value must be of type *mofy_rel*—that is, an NP headed by a month noun like *January*.

There are two lexical entries for *January* which fit the bill. Both select ordinary specifiers, but one selects an optional NP complement with a KEY value of type *yofc_rel* (45f), and the other selects an optional PP[of] complement with a COMPKEY value of type *yofc_rel* (45g). *1984* has a HEAD value of type *noun* and a KEY value of type *yofc_rel*. Since both *january* and *1984* are proper nouns, they are licensed as specifier-less NPs by the unary rule proper_np.

In (45b) the optional complement of *january* is not realized, and in (45c) the optional complement of *first* is not realized. In (45a) both complements are realized.

On the other hand, the lexical head of the NPs in examples (45d) and (45e) is a lexical entry for *first* derived from the one discussed above by the lexical rule dofm_yofc. This derived lexical entry selects a specifier with a HEAD value of type *det* and a ALTKEY value of type *mofy_rel*, and it selects an optional NP complement with a KEY value of type *yofc_rel*. In these examples, *january* is a determiner; the lexical entry is derived from the nominal version by the lexical rule monthdet. Since *first* cannot be licensed as a specifier-less NP, the ungrammatical (45h) is ruled out.

Finally, the lexical head of (45i) is a base lexical entry for *three*, which has a HEAD value of type *noun* and a KEY value of type *dofm_rel*. This lexical entry selects a non-optional specifier with a HEAD value of type *det* and an ALTKEY value of type *mofy_rel*—that is, the derived lexical entry for *january* discussed above. *Three* also selects an optional NP complement with a KEY value of type *yofc_rel*.

Note that date expressions are somewhat idiosyncratic in that separate lexical entries for all ordinal and cardinal numbers 1-31 are needed in the lexicon for this analysis to be fully implemented.

5.1.5.4 Temporal modifying NPs

MERGE licenses temporal NPs as modifying expressions with the non-headed unary temp_nwh rule. In this construction, the daughter is an N' with a KEY value of type *modable_rel*, while the mother has a HEAD value of type *mod_np* and a non-empty MOD value, specifying that the phrase can modify a VP or an N'.

(46) a. Kim sleeps [three times a day].
b. Kim sleeps [every day].

c. Kim slept [Monday].
d. Kim slept [January first].
e. * Kim slept [the dog].

This rule licenses the NPs in (46a)-(46d) as modifying expressions, since they are headed by nouns with a KEY value of type *modable* rel, but it does not license *a dog* as such.

Example (46a) deserves further attention, because of its idiosyncratic analysis in MERGE. *Three times a day* is a bare NP; its lexical head is *times*, *three* is a prenominal modifier, and *a day* is a postnominal modifier. MERGE includes a lexical entry for *a* which is a preposition. This preposition can only modify an N', but there are no semantic restrictions on either its complement NP or the N' it modifies. Furthermore, it selects a specifier with a KEY value of type *very_deg_rel*. As it is currently implemented, the inclusion of this lexical entry has the result that MERGE licenses the ungrammatical, or at least very strange, NP *a dog very a day*.

5.1.5.5 Partitive constructions

In MERGE, the variation in (47) is licensed by positing a number of different lexical entries, some being base lexical entries, some being derived via lexical rules. In (47a), which does not have a partitive meaning, *all* is a determiner specifying the lexical head *dog*. *All* is called a *partitive determiner* (a determiner whose HEAD value is of type *part_det*) only because its lexical input is admissible input to the lexical rules that derive partitive nouns.

Partitive nouns express partitive meaning, implicitly or explicitly, depending on whether they have a complement or not. They are the lexical head of the NP they occur in, and select either a PP complement (47b), no complement (47c), or (in some cases) an NP complement (47d).

(47) a. All dogs sleep.
b. All of the dogs sleep.
c. All sleep.
d. All the dogs sleep.

Partitive determiners: any, some, either, few, many, several, both, most, all, more, enough, each, one. (With specific agreement properties.) Lexical rules derive two nominal lexical entries from each of these determiners: one which does not select a complement (part_nocomp_constr), and one which selects a PP[of] complement (part_ppof_agr_constr or part_ppof_noagr_constr, depending on the KEY value of the partitive determiner).

Partitive nouns which can select an NP complement (*all*), which do not occur as determiners so they cannot be derived by lexical rules (*none*), or both (*half*), are included in the lexicon as base lexical entries.

Unlike most nouns, the specifier of a partitive noun is something with a KEY value of type *degree_rel*. This licenses things like *very many of the dogs*, where *very* is the specifier of the partitive noun *many*. However, this uniformity of SPR values for all of the partitives overgenerates in some cases. For example, MERGE currently licenses the ungrammatical **very all the dogs* and **very one of the dogs*.[2]

[2]NPs like *this one* and *that one* are licensed by a non-partitive nominal lexical entry for *one*, with standard SPR specifications for a noun.

No-agreement partitive determiners: how-many, few, many, several, both, each, and one. Agreement partitive determiners: which, any, some, either, most, all, more, and enough. The difference is that an agreement partitive noun structure-shares its PN value with its complement, if one is realized. This enforces person-number agreement between the verb that selects a partitive-headed NP and the NP that is either the complement of the partitive noun, or embedded in its PP[of] complement. No-agreement partitive nouns simply inherit their AGR specifications from the partitive determiner input. No relationship between the AGR specifications of the partitive noun and the AGR specifications of the embedded NP is enforced. Thus, since *most* is an agreement partitive, MERGE licenses both *most of the dogs sleep* and *most of the dog sleeps*, but not **Most of the dogs sleeps*. Since *many* is a no-agreement partitive specified as pn:thirdpl, MERGE licenses *many of the dogs sleep*, but not **many of the dog sleeps*. However, since no agreement relationship between partitive and embedded NP is enforced, MERGE also licenses the ungrammatical **many of the dog sleep*.

all but five dogs sleep

Finally, the variation shown in (48) is licensed by a combination of lexical variation and different phrase structure rules.

(48) a. These five dogs sleep.
b. Five dogs sleep.
c. Five of the dogs sleep.

The base lexical entry for *five* (and all cardinal number expressions greater than one) is an adjective. Thus both (48a) and (48b) involve prenominal modification by *five*, but (48a is a head-specifier construction, and (48b) is a bare-plural NP licensed by the unary PSR bare_np. The *five* in (48c), however, is a partitive noun, derived from the adjective via the lexical rule partitive_num.[3]

Partitive numbers select specifiers similar to other partitives, so the ungrammatical **very five of the dogs* is licensed. In terms of agreement, the derived lexical entries for partitive numbers have AGR values specified as divisible:plus_star and pn:thirdpl, but the value of AGR is also structure-shared with the AGR value of the complement, thereby ruling out examples like **five of the dog*.

The same lexical rule takes ordinal numbers like *first* as input, licensing examples like *first of the dogs*. The partitive number *first* is not specified pn:thirdpl like *five* is, but it is specified as divisible:plus_star, ruling out ungrammatical examples like *the first of the dog*, since singular count nouns are marked divisible:minus

5.1.5.6 Compound nouns

In MERGE, all of the NPs in (49) involve compound noun constructions.

(49) a. her own dog
b. csli managers
c. a department office manager
d. paris france

[3]This lexical rule was a non-headed unary phrase in ERG.

In all compounds, the right-most noun is the head of the compound, and must be lexical. This lexical item determines the selectional properties of the whole. Complex compounds are built up stepwise; the non-head daughter (on the left) may be a lexical noun or a proper name NP (licensed by separate rules), but the result is always lexical.

One idiosyncratic feature of this grammar pertaining to compounds is of note. As in ERG, *own* is a noun which can either occur alone or as part of a compound.

5.1.5.7 Free relatives

MERGE licenses both of the examples in (50) as NPs, referred to as free relative constructions.

(50) a. what you need
b. when you visited the apartment

Free relative phrases may occur as the subject or complement of a verb or preposition, as shown in (51).

(51) a. [What you found] is sleeping.
b. I showed Kim [what you found].
c. [What you found] is in the apartment.
d. The dog is sleeping by [what you found].

Free relative phrases are licensed by a special rule; their internal structure is discussed in section 4.2.14.5.

5.1.6 Comparative and superlative expressions

MERGE licenses a number of comparative and superlative expressions, in three categories: prenominal modifiers, predicative adjective phrases, and noun phrases.

5.1.6.1 Prenominal modifiers

In MERGE, lexical entries for *more* and *most* select non-optional adjectival complements, to form adjective phrases which can occur as prenominal modifiers, as illustrated by (52a) and (52b).

(52) a. the [more competent] managers
b. the [most competent] manager

That is, since the PRD and POSTHEAD values of *more* and *most* are not specified, the phrases they head are admissible prenominal modifiers.

5.1.6.2 Predicative adjective phrases

Comparative AdjPs like (53a) and (53b) have an adjectival lexical head, augmented with either specifiers post-modifiers that occur independently of each other.

(53) a. Kim is [(twice) as old] as Abrams

b. Kim is [older than Abrams]

For example, an adverbial lexical entry for *as*, with a KEY value of type *eq_degree_rel*, can occur as the specifier of the adjective *old*. Furthermore, a lexical entry for *twice*, which has a HEAD value of type *adv* and a KEY value of type *very_deg_rel*, can occur as the specifier of the adverb *as*. Taken together, this licenses the AdjP *twice as old*. However, the string *as Abrams* is not part of the AdjP in (53a). Instead, a prepositional lexical entry for *as*, which selects an NP complement, can modify any VP or N', and here modifies the VP *is twice as old*. Thus, MERGE also licenses the ungrammatical examples *Kim sleeps [as Abrams]* and *the dog [as Abrams]*.

On the other hand, a prepositional lexical entry for *than* must modify something with a HEAD value of type *adj*, and a KEY value of type *comp_rel*. Thus, the PP *than Abrams* modifies *older* in (53b).

5.1.6.3 Comparative NPs

The licensing of (54) depends on multiple lexical entries for *as*, which function independently of one another.

(54) Abrams manages as many managers as engineers.

An adverbial *as* with a KEY value of type *eq_degree_rel* can function as the specifier of the determiner *many* to license *as many managers*. A prepositional *as* which takes an NP complement, can modify *managers* to license *managers as engineers*. Although this cannot occur as an NP in English, it is licensed as such in MERGE.

MERGE cannot license comparative NPs like *more programmers than engineers*, however, since the preposition *than* (discussed above) can currently only modify adjectives.

5.1.7 Postnominal modification

Every instance of postnominal modification in the MERGE is classified as a relative construction, either as a relative clause or as a reduced relative. Relative clauses are subcategorized as either *wh*-relatives (which here include *that*-relatives), and non-*wh*-relatives. Non-*wh*-relatives are themselves divided into the finite non-*wh*-relatives (commonly known as bare relatives), and infinitive non-*wh*-relatives. Examples of each construction are given in figure 5.1.

Briefly, relative clauses contain a "gap site" linked semantically to the noun they modify, and that gap may correspond to either an argument (subject or complement) or an adjunct position in the matrix clause or an embedded phrase or clause.

Wh-relatives have a preposed *wh*-phrase corresponding to the gap, non-*wh*-relatives do not. The term *wh*-phrase refers to a phrase consisting of a so-called relative pronoun alone, or embedded in another phrase. The phenomenon of preposing a phrase containing a relative pronoun is commonly known as *pied-piping*, and the MERGE handles these cases as well.

The matrix verb in a *wh*-relative or a bare relative is finite, while the matrix verb in an infinitive non-*wh*-relative is an infinitive. In a bare relative, the gap cannot correspond to the subject position of the matrix verb. Non-*wh*-relatives also lack an overt subject.

1. Relative clauses

   (a) *Wh*-relatives

   The dog who/which/that _ visited me sleeps.
   The dog who/whom/which/that I visited _ sleeps.
   I visited the apartment where the dog sleeps _.
   The dog to whom I showed an apartment _ sleeps.

   (b) Non-*wh*-relatives

      i. Bare relatives

      The dog I visited _ sleeps.
      **The dog _ visited me sleeps.*

      ii. Simple infinitival relatives

      The manager to hire _ sleeps.
      The manager _ to hire me sleeps.

2. Reduced relatives.

   The dog visiting the apartment sleeps.
   The dog in the apartment sleeps.

Figure 5.1: Example sentences containing postnominal modification

In the MERGE, reduced relatives do not represent a phrasal type *per se*, but rather a collection of phrasal types including prepositional phrases and present participial verb phrases.

Coverage does not include infinitive non-*wh*-clauses introduced by *for* (e.g., *the dog for me to visit*) or iterative modification by relative clauses (e.g., *the dog I visited that barks*). The former are simply not included in the coverage, while the latter are explicitly ruled out (see section (5.1.7)). However, iterative modification by reduced relatives is covered (e.g., *the dog in the apartment visiting me*), as well as modification by a relative clause when the noun has already been modified by a reduced relative (e.g., *the dog visiting me that sleeps*).

Example structures In this section, examples of structures licensed by the MERGE are given, with brief descriptions. In each, the subject of the clause is a noun phrase in which the noun is modified by a relative construction. Figure 5.2 contains a *wh*-relative introduced by *that*, with a subject gap, figure 5.3 contains a finite non-*wh*-relative, with a complement gap, and figure 5.4 contains a prepositional phrase licensed as a reduced relative.

In order to license these structures, several types of phrase structure rules are needed. In every case, some rule must license the modification itself; that is, the node labeled N which has two daughters: one dominating a noun, the other dominating the relative construction. In the case of a reduced relative, we do not need anything special to license the latter; as previously mentioned *reduced relative* does not refer to a phrasal type, and the modifying phrase is licensed by an ordinary phrase structure rule. In figure 5.4, for instance, the prepositional phrase is licensed by a head-complement schema.

In the MERGE, relative clauses require special schemata to license both the clause itself, that is, the S node whose sister dominates the modified noun, and extraction. Notice that this S node in a *wh*-relative (figure 5.2) is binary branching, while the S node in a non-*wh*-relative (figure 5.3) is unary branching; structurally, these categories of relative clause differ in the presence or absence of a *wh*-phrase daughter.

Every relative clause has a daughter with a finite or infinitive verbal head, further, every such structure dominates a node licensed by a unary branching extraction schema. These schemata are in some sense special instances of the type of schema whose would-be non-head daughter corresponds to the gap, be it subject, complement, or adjunct. Thus, a node licensed by an extraction rule will always occupy the place in the structure which would otherwise by licensed by a subject-head, head-complement, or head-adjunct schema.

Thus, in figure 5.2, the node labeled S/NP would in an ordinary clause be licensed by a subject-head schema, while in figure 5.3 the node labeled VP/NP would in an ordinary clause be licensed by a head-complement schema.

Finally, the preposed *wh*-phrase in a *wh*-relative clause does not rely on any special phrase structure rule; instead, relative pronouns are included in the lexicon as pronominal lexical items, and phrases with embedded relative pronouns are licensed by ordinary phrase structure rules, but marked with a special nonlocal feature specification (a nonempty *v*alue) which propagates from the relative pronoun to the mother node of the entire preposed phrase (see section (5.2.2.1)).

Licensing postnominal modification In all cases, postnominal modifiers combine with what is commonly known as an N' (loosely speaking, something with a *noun* HEAD value, an empty COMPS list, and a nonempty SPR list). That something must be a phrase (specified as

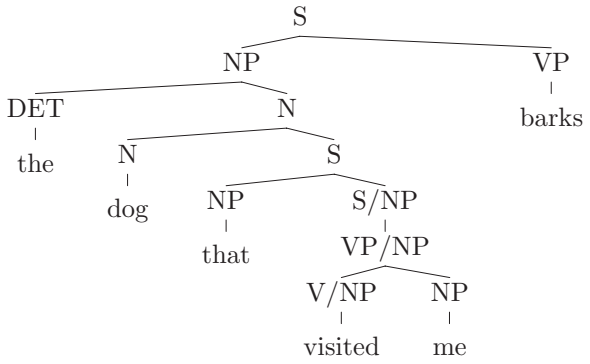


Figure 5.2: Postnominal modification by a *wh*-relative with a subject gap

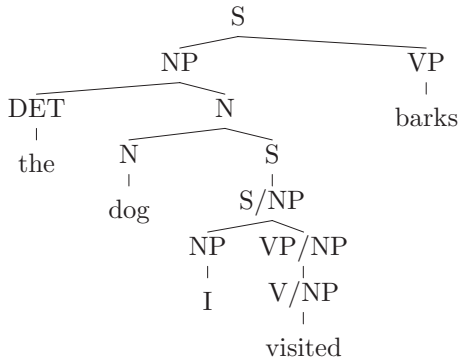


Figure 5.3: Postnominal modification by a bare relative with a complement gap

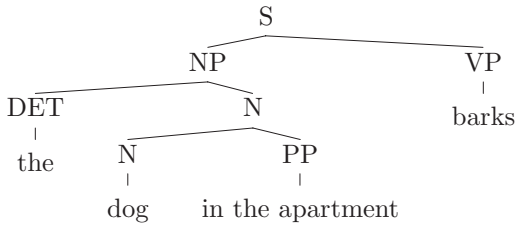


Figure 5.4: Postnominal modification by a prepositional phrase (reduced relative)

```
@n_adj_redrel_nontemp(HeadDtr,NonheadDtr) macro
    n_adj_redrel_nontemp,
    (head_dtr:(HeadDtr,
               synsem:local:cat:(head:(noun,
                                       val:comps:e_list))),
     nonhead_dtr:(NonheadDtr,
                  synsem:(phr_synsem,
                          local:cat:(head:(tam:tense:no_tense,
                                           prd:plus,
                                           mod:(hd:(local:cat:(head:Head,
                                                               val:Val,
                                                               agr:Agr,
                                                               cont:index:Index,
                                                               keys:key:Key),
                                                    nonlocal:Nonloc,
                                                    modified:Modif),
                                                tl:e_list),
                                      val:subj:[unexpressed])),
                          nonlocal:rel:e_list))).
```

Figure 5.5: Macro for schema licensing modification by a nontemporal reduced relative

lex:minus), specified as prd:plus, and have a MOD value consistent with the modified noun in a variety of respects, as specified in @basic_head_mod_phrase_simple.

Such phrases are either of type *n_adj_redrel_temp* or *n_adj_redrel_nontemp* (involving a so-called reduced relative), or of type *n_adj_relcl_phrase* (involving a relative clause).

Modification by a reduced relative As previously mentioned, reduced relatives are not a phrasal type *per se*, and unlike other relatives they have an empty REL list. In addition to the specifications discussed above, the non-head daughter in such a phrase must have a nonempty SUBJ list whose sole member is of type *unexpressed*, and TENSE value *no_tense*. Qualified candidates include prepositional phrases and present participial verb phrases. A simplified version of a macro describing a phrase of type *n_adj_redrel_nontemp* is shown in figure 5.5.[4]

Modification by a relative clause In addition to the specifications mentioned above, modification of an N' by a relative clause requires that the non-head daughter have a finite or infinitive VFORM value, and that its SUBJ and COMPS lists have a value of type *o_list*. This is a list all of whose members, if any, are specified as opt:plus and of type *unexpressed*. The mother is marked modified:periph:plus. This blocks iterative modification by relative clauses, since the MOD value of a relative clause specifies that it will only modify things

[4] All such descriptions throughout the rest of the documentation do not correspond to macros in the actual code, rather, they are simplified and collapsed descriptions for purposes of illustration.

```
@n_adj_relcl_phrase(HeadDtr,NonheadDtr) macro
    n_adj_relcl_phrase,
    (synsem:modified:periph:plus
     head_dtr:(HeadDtr,
               synsem:local:cat:(head:(noun,
                                       val:comps:e_list))),
     nonhead_dtr:(NonheadDtr,
                  synsem:(phr_synsem,
                          local:cat:(head:(vform:fin_or_inf,
                                           prd:plus,
                                           mod:(hd:(local:cat:(head:Head,
                                                               val:Val,
                                                               agr:Agr,
                                                               cont:index:Index,
                                                               keys:key:Key),
                                                               nonlocal:Nonloc,
                                                               modified:Modif),
                                                    tl:e_list)),
                                      val:(subj:o_list,
                                           comps:o_list))))).
```

Figure 5.6: Macro for schema licensing modification by a relative clause

marked modified:periph:na_or_minus. A simplified version of a macro describing a phrase of type *n_adj_relcl_phrase* is shown in figure 5.6.

5.2 The verbal domain

5.2.1 Complementation

5.2.1.1 Optional Complements

The treatment of optional complements is discussed in detail in section 2.1.1.3.

5.2.2 Modification

5.2.2.1 Relative Clauses

All relative clauses are non-inverted finite or infinitive clauses, specified as posthead:plus and prd:plus. Unlike other clauses, they have a nonempty MOD value.

In all cases relative clauses have a daughter with a verbal HEAD value and a nonempty SLASH list, whose value corresponds to an unrealized argument or adjunct.

Relative clauses are not headed phrases, so that SLASH value may be absent on the mother (and it is). Further, a nonempty value is stipulated on the mother, while not present on the

daughter.[5] Since these phrases are not subject to the HFP, various head features are passed explicitly from daughter to mother.

Relative clauses are distinguished between *wh*-relatives, which are binary phrases, and non-*wh*-relatives, which are unary phrases. The leftmost daughter of a *wh*-relative, a phrase of type *filler_head_rule_rel*, is a *wh*-phrase, characterized by a nonempty ᵥalue. Its LOCAL value is structure-shared with the SLASH value of the rightmost daughter.

The simplified macro describing a phrase of type *filler_head_rule_rel*, the type for *wh*-relative clauses, is shown in figure 5.7.

Non-*wh*-relatives include the maximally specific phrase types *fin_non_wh_rel_cl* (finite non-*wh*-relatives, or bare relatives) , and *inf_non_wh_rel_cl* (infinitive non-*wh*-relatives, or simple infinitive relatives). These are distinguished primarily by their VFORM and SUBJ values.

The mother has empty SUBJ and COMPS lists, regardless of what the daughter's values were (although the daughter's COMPS list must be of type *o_list*). Thus, the daughter may be phrasal, as is the case with simple infinitive relatives with complement gaps.

A simplified version of the macro describing phrases of type *fin_non_wh_rel_cl* (bare relatives) is shown in figure 5.8.

Relative pronouns and pied-piping The MERGE recognizes the relative pronouns *who*, *whom*, *which*, *whose*, *where*, *when* and *that*. These lexical items are distinguished from all others by the fact that they have a nonempty list as the value of the nonlocal feature REL. Like other pronouns, they are specified as lex:plus, inflected:plus, and have an empty ARG_S list. A partial description of the relative pronoun *which* is given in figure 5.9.

To account for those cases where the relative pronoun is embedded in a *wh*-phrase (the phenomenon referred to as pied-piping), this nonempty REL value must be propagated to the top of the phrase. The mechanisms responsible for such propagation are described in section (4.5.3). Briefly, a constraint ensures that the ᵥalues of everything on the ARG-S list of a lexical head are appended, and the resulting list is the ᵥalue of that lexical head. The macro @head_nexus_rel_phrase specifies that the REL value of a head daughter is identical to that of the mother.

5.2.3 Extraction

Extracted-argument phrases are *head_only* phrases, so the HFP applies, and are required to satisfy the description @head_valence_phrase, so nonlocal features are passed from daughter to mother.

In every case, some valence item is specified as an object of type *gap*, a *synsem* object whose LOCAL value is structure-shared with the element on its SLASH list. Previously discussed constraints and specifications ensure that this SLASH value ends up on the mother's SLASH list.

In a phrase of type *extracted_comp_phrase*, the first element on the COMPS list of the daughter is a gap synsem. The rest of the COMPS list, as well as the SUBJ and SPR values, are passed up to the mother. A simplified version of the macro describing a phrase of type *extracted_comp_phrase* is shown in figure 5.10.

[5]The REL value of a relative clause is not "collected" by the noun it modifies, since the relative clause is a modifier and does not appear on the ARG_S list of the noun.

```
filler_head_rule_rel(LDtr,RDtr)  macro
    filler_head_rule_rel,
    (synsem:(local:cat:(head:(vform:(fin_or_inf,
                                     Vform),
                              tam:Tam,
                              aux:Aux,
                              inv:(minus,
                                   Inv),
                              prd:plus,
                              mod:(Mod,
                                   hd:local:cat:cont:index:Index,
                                   tl:e_list)),
                        val:subj:(Subj,
                                  o_list),
                        posthead:plus,
                        cont:index:Index),
             nonlocal:(slash:e_list,
                       rel:ne_list)),
     dtrs:(hd:(LDtr,
               synsem:(local:Local,
                       nonlocal:rel:(hd:index:Index,
                                     tl:e_list))),
           tl:(hd:(Rdtr,
                   synsem:(local:cat:(head:(vform:(Vform,
                                                   fin),
                                            tam:Tam,
                                            aux:Aux,
                                            inv:Inv,
                                            mod:Mod),
                                      val:subj:Subj),
                           nonlocal:slash:Local),
               tl:e_list)))).
```

Figure 5.7: Macro for schema licensing *wh*-relative clauses

```
fin_non_wh_rel_cl(Dtr)  macro
    fin_non_wh_rel_cl,
    (synsem:(local:cat:(head:(vform:(fin,
                                     Vform),
                              inv:(minus,
                                   Inv),
                              prd:plus,
                              mod:(Mod,
                                   hd:local:cat:(head:noun,
                                                 val:(spr:ne_list,
                                                      subj:e_list,
                                                      comps:e_list),
                                                 cont:index:Index,
                                                 keys:key:Key),
                                   tl:e_list)),
                        val:(subj:e_list,
                             comps:e_list),
                        posthead:plus,
                        cont:index:Index,
                        keys:key:Key),
              nonlocal:(slash:e_list,
                        rel:ne_list)),
     dtrs:(hd:(Dtr,
               synsem:(local:cat:(head:(verbal,
                                        vform:Vform,
                                        tam:Tam,
                                        aux:Aux,
                                        inv:Inv,
                                        mod:Mod),
                                  val:comps:o_list,
                                  mc:plus,
                                  keys:key:Key),
                       nonlocal:(slash:(hd:(synsem:local:cat:(head:noun,
                                                              cont:index:Index,
                                                              keys:key:Key)),
                                            tl:e_list),
                                rel:e_list,
                                que:e_list))),
           tl:e_list)).
```

Figure 5.8: Macro for the schema licensing bare relatives

```
(phon:[which],
 synsem:(lex:plus,
         local:(arg_s:[],
                cat:head:noun_star),
         nonlocal:rel:[index:individual]),
 inflected:plus).
```

Figure 5.9: Partial description of *which*

```
extracted_comp_phrase(Dtr) macro
   extracted_comp_phrase,
   (synsem:local:cat:val:(spr:Spr,
                          subj:Subj,
                          comps:Comps),
    head_dtr:synsem:(local:cat:val:(spr:Spr,
                                    subj:Subj,
                                    comps(hd:(gap,
                                              nonlocal:slash:Slash),
                                          tl:Comps),
                     nonlocal:slash:Slash)).
```

Figure 5.10: Macro for the schema licensing complement extraction

Phrases of type *extracted_subj_phrase_fin* or *extracted_subj_phrase_inf* have empty valence lists. The head daughter has a gap specified as case:nom on its SUBJ list. A simplified version of the macro describing a phrase of type *extracted_subj_phrase_fin* is shown in figure 5.11.

Since adjuncts do not appear on any ARG_S list, adjunct "extraction" must be handled differently than those cases involving arguments. Phrases of type *extracted_adj_int_phrase* are a subtype of *head_only_phrase*, so the HFP applies, but are not head-valence phrases. Thus, a SLASH value may be specified on the mother which is not present on the daughter. That SLASH value has a MOD value consistent in a variety of respects (HEAD, VALENCE, etc.) with the daughter. The daughter must have an empty SUBJ list. A simplified version of the macro describing a phrase of type *extracted_adj_int_phrase* is shown in figure 5.12.

5.3 Coordination

The treatment of coordination in MERGE deserves special attention, since it helps to explain some unexpected aspects of the grammar elsewhere, and it requires significant complexity in the signature to support this treatment.

The general idea is that coordinate phrases are binary non-headed structures, and the right daughter is always a (so-called) head-marker phrase. However, the marker (a conjunction in this case) is the head of a head-marker phrase in MERGE.

The licensing of coordinate structures depends on the structure-sharing of certain feature

```
extracted_subj_phrase_fin(Dtr) macro
   extracted_subj_phrase_fin,
   (synsem:local:cat:(head:vform:fin,
                      val:(spr:e_list,
                           subj:e_list,
                           comps:e_list),
                      mc:minus),
    head_dtr:(Dtr,
              synsem:(local:cat:val:(subj:hd:(gap,
                                              local:(Local,
                                                     cat:head:case:nom)),
                                     comps:o_list),
                      nonlocal:slash:hd:Local))).
```

Figure 5.11: Macro for a schema licensing subject extraction

```
extracted_adj_int_phrase(Dtr) macro
    extracted_adj_int_phrase,
    (synsem:(local:cat:(head:mod:(hd:local:cat:head:noun,
                                  tl:e_list),
                        val:(subj:Subj,
                             spr:Spr,
                             comps:e_list)),
             nonlocal:slash:hd:cat:(head:mod:(hd:cat:(head:Head,
                                                      val:Val),
                                              tl:e_list),
                                    val:(subj:prolist,
                                         comps:o_list,
                                         spr:o_list))),
     head_dtr:synsem:local:cat:(head:(Head,
                                      verb),
                                val:(Val,
                                     subj:Subj,
                                     spr:Spr,
                                     comps:o_list))).
```

Figure 5.12: Macro for the schema licensing adjunct extraction

values between the sisters. Thus, the values of those features must be unifiable. The coordination of unlikes (in some sense or another) depends on the inclusion of special coordination-subtypes in the type hierarchy.

5.3.1 Coordination of nouns and NPs

The person-number agreement specifications of a coordinated phrase depend on the conjunction. Phrases conjoined by *and* have plural agreement, regardless of what the conjuncts are (since the result of conjunction with and is always plural). Agreement features of a phrase conjoined by *or* are determined by the right conjunct.

(55) a. She and I sleep.
b. * She and I sleeps.
c. She or I sleeps.
d. * She or I sleep.

More complex coordinate structures can be built up by means of the mid-level coordination rule, and the CONJ feature, as shown by the examples in (56).

(56) a. chiang and devito
b. browne chiang and devito
c. either browne or devito
d. browne or devito
e. either browne and devito or chiang

The right daughter in a top-level coordinate phrase must be specified as conj:complex_conj. Mid-level coordinate phrases are specified as conj:phr_conj, and head-marker phrases inherit the conj:lex_conj value from the marker daughter (the conjunction). Both *phr_conj* and *lex_conj* are subtypes of *complex_conj*. Since the right daughter of a mid-level coordination phrase must also have the specification conj:complex_conj, nested mid-level coordination phrases must eventually bottom out with a head-marker phrase: a conjunction followed by a noun or NP. Furthermore, either daughter in a coordinate phrase may itself be a coordinate phrase.

Finally, the value of the CONJ feature is also used to regulate the co-occurrence of the conjunction pairs *either/or*, *neither/nor*, and *both/and*. Special lexical entries for *or*, *nor*, and *either* specify within their *conj* value that the left conjunct should be marked by a particular conjunction. The top-level coordination rule specifies that this information is structure-shared with the *conj* value of the left conjunct.

The treatment of coordination relying on unification of various feature values results in some limitations which should be pointed out. For example, since both conjuncts structure-share the DIVISIBLE feature, it is not possible in this treatment of coordination to conjoin a plural or mass noun with a singular count noun. (This holds for ERG, also.)

Bibliography

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

Briscoe, Ted, 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.

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

Copestake, Ann (2002). *Implementing Typed Feature Structure Grammars*. Stanford, CA: CSLI Publications.

Copestake, Ann 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.

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

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

Meurers, W. Detmar 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*, Stanford, CA: CSLI Publications, pp. 29–63.

Meurers, W. Detmar, 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.

Meurers, Walt Detmar (1994). On Implementing an HPSG Theory – Aspects of the Logical Architecture, the Formalization, and the Implementation of Head-Driven Phrase Structure Grammars. In Erhard W. Hinrichs, Walt Detmar Meurers and Tsuneko Nakazawa (eds.), *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation*, Tübingen: Universität Tübingen, no. 58 in Arbeitspapiere des SFB 340, pp. 47–155. `http://ling.osu.edu/~dm/on-implementing.html`.

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

Parnas, David Lorge (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.

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