On Implementing an HPSG theory*

Aspects of the logical architecture, the formalization, and the implementation of head-driven phrase structure grammars

Walt Detmar Meurers
Sonderforschungsbereich 340, Universität Tübingen
Kleine Wilhelmstr. 113, 72074 Tübingen, Germany
email: dm@sfs.nphil.uni-tuebingen.de

In: Erhard W. Hinrichs, W. Detmar Meurers, and Tsuneko Nakazawa:
Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation.
Arbeitspapiere des SFB 340 Nr. 58, Universität Tübingen, 1994

Table of Contents

*I am grateful to Thilo Götz, Erhard Hinrichs, Tilman Höhle, and Paul King for valuable discussion and advice, and to Tsuneko Nakazawa for many interesting discussions of concepts in and alternatives to the theory proposed by Erhard Hinrichs and herself. For comments on a previous version of the paper I want to thank John Griffith, Tibor Kiss, Esther König-Baumer, Frank Morawietz, Jürgen Pafel, Frank Richter, and Susanne Riehemann.

1 Introduction

The paper documents the implementation of an HPSG theory for German in a constraint resolution system. It illustrates the reasoning and the choices involved in the formalization and implementation of HPSG grammars. As basis for the reasoning, a discussion of the logical setup of the HPSG architecture as proposed in Pollard and Sag (1994) (henceforth HPSGII) is supplied. The relationship between a linguistic theory and its implementation is explored from a linguistic point of view.

The grammar implemented covers the phenomena of aux-flip and partial verb phrase topicalization in the three sentence types of German: verb first, verb second, and verb last. It closely follows the analyses proposed in Hinrichs and Nakazawa (1991) and Hinrichs and Nakazawa (1994) (henceforth HN), except for a few cases noted in section 5.

The implementation is based on two systems: Bob Carpenter and Gerald Penn's ALE, and the Troll system developed by Dale Gerdemann and Thilo Götz based on a logic by Paul King. The discussion of computational architectures in this paper also includes references to other systems, such as Martin Emele and Rémi Zajac's TFS, or the CUF system developed by Jochen Dörre, Michael Dorna, and Andreas Eisele.[1]

The structure of the paper is as follows: The remaining part of this introduction briefly reports on the why and how of the project: the motivations behind the implementation, and the systems used. The second section describes the logical setup of HPSG: the description language and its interpretation. The third section deals with the feature logic building blocks of the HPSG architecture: the declaration of the domain of the grammar and the possibilities for formulating the constraints making up the linguistic theory. The focus is on the implications of the logical setup for the work of the linguist and examples from the implemented grammar are supplied. The fourth section goes through the linguistic modules of an HPSG theory: the lexicon, the specification of constituent structure, and the grammatical principles. The choices involved in formalizing them for implementation are discussed and illustrated with examples. The fifth section discusses the differences between the linguistic theory in HN and the implemented grammar and gives some technical comments on the implemented grammar. The complete grammar including a collection of test sentences and a test run are included in the appendix.

1.1 Motivation

There are at least two different motivations for implementing HPSG theories.

From a computational point of view a complex theory serves well as a test for the feature logic systems being developed. It can illustrate performance and theoretic devices, such as named disjunction or full negation, or more specific mechanisms, like definite clause attachments on lexical entries. The line of thinking behind such implementations starts from the devices, explores their usefulness in compactly encoding linguistic generalizations, and then illustrates the computational setup in an implementation. Even though the intentions under this approach are oriented towards computation, the work can provide feedback for linguistics in supplying new theoretical devices for expressing linguistic theories.

[1] For a brief overview of the main characteristics of TFS and CUF including some comments on the implications for the work of the linguist cf. Meurers and Götz (1993). Manandhar (1993) is a comparison of ALE, CUF, and TFS regarding the expressive power of the type system, the definite clauses, and the control scheme.

The intention of a linguist on the other hand is to use the development of an implementation as a tool to provide feedback for a rigid and complete formalization of a linguistic theory. Under this approach the work starts out from the theory and tries to put the mechanisms proposed in it to work together in a grammar in an elegant and correct fashion.

Since the work presented here started out from a linguistic theory, it is closer to the second view. Nonetheless, the implementation also shows that both systems employed, ALE and Troll, can be used to express and process rather complex grammars. One of the effects of the linguistic approach pursued is that the implemented grammar tries to stay close to the notation and formulation used in the linguistic theory. The computational aspect causes some limitations here, the major one being the departure from ID/LP format due to the phrase structure based systems used.

1.2 The systems used

This section briefly introduces the computational setup of the implementation. A detailed discussion of the theoretical properties and differences between the two systems (which made it interesting to try to use the same grammar in both approaches) is included in the next sections, where the HPSG architecture and its logical foundation are discussed.

The purpose of the Troll system is to efficiently implement a logic that gives a denotational semantics to typed feature structures as normal form descriptions as opposed to thinking of feature structures as models[2]. This allows unfilling of feature structures since there is a clear notion of semantic equivalence of different feature structures (cf. Götz (1994)). Unfilling is an operation which removes arcs not constraining the denotation of a feature structure. Technically it allows for efficient representation of descriptions, while for the linguist it serves to suppress the information that can be inferred from the signature. Troll also assumes a closed world interpretation of type hierarchies which seems natural for linguistics and HPSG in particular.

Due to the nature of Troll, the notation only represents the concepts relevant from the viewpoint of logic. For the implementation to reflect the linguistic theory, several extensions are necessary: Macros[3] serve to follow linguistic notation and avoid unnecessary, unclear and error prone repetition. Lexical rules, even though their theoretical status is not completely clear, are used in many HPSG publications including the one implemented, and therefore need to be part of the language. Finally, to enable debugging of a non-trivial grammar, a source level debugger allowing very selective output of information as well as good displaying and editing possibilities for the extensive feature structures is necessary.

Since ALE[4] offers the first two devices, macros and something resembling lexical rules, the grammar was written in ALE, and then automatically translated[5] to Troll. At the time this

[2]This ontological difference between the logic of King (1989) and that of Carpenter (1992) is elaborated in section 2.

[3]The use of macros in the lexicon and the problems involved in replacing them with an appropriate formal mechanism are discussed in section 4.1.2.

[4]For a description of the ALE system, the reader is referred to Carpenter (1993). ALE Version 1.0, release: 1. May 1993 was used. All comments refer to that version.

[5]A technical note on the translation: Since in Troll everything is encoded in a feature structure data structure, a few special types (e.g. *ps-rule*) and a type for every definite clause together with its appropriate argument types are added to the translated grammar. Encoding definite clauses as feature structures enables the grammar writer to restrict the arguments of each definite clause type by specifying the signature. This enables the system to do more static type checking.

work was carried out, neither of the two systems offered a real debugger; so debugging of the grammar was done the old way: by thoroughly studying the grammar and massive testing.

2 The HPSG ontology: A sketch of the logical foundations

Before going through the ingredients of an HPSG grammar and the possibilities for formalizing them in the various architectures, let us sketch the ontological and logical basis of the architectures we will mostly be dealing with: HPSGII, ALE, and Troll.

This sketch seems worth making here, since the setup has direct consequences on the possibilities in and the extendibility of the logic based systems. A lot of the differences between the systems discussed later on can be traced back to the way the syntax and semantics of the logic behind the system is set up. Additionally this short introduction to the logical basis allows us to introduce the terminology needed later on (and its diverging use in the various architectures).

The HPSG formalism as defined in HPSGII is at the basis of the linguistic analysis developed within the HPSG paradigm over the last several years.[6] We therefore start by looking at the ontological setup described in the introduction of HPSGII:

> In any mathematical theory about an empirical domain, the phenomena of interest are *modelled* by mathematical structures, certain aspects of which are conventionally understood as corresponding to observables of the domain. The theory itself does not talk directly about the empirical phenomena; instead, it talks about, or is *interpreted by*, the modelling structures. Thus the predictive power of the theory arises from the conventional correspondence between the model and the empirical domain. (HPSGII, page 6)

In other words, the first step of every scientific process is the modelling of the domain. Models[7] of empirically observable objects are established to capture the relevant properties. Theories then make reference to these models. This opens up three questions: What do the mathematical structures used as models for HPSG theories look like? How are they related to the linguistic objects? and How is the linguistic theory put to work on the models?

> In HPSG, the modelling domain ... is a system of *sorted feature structures*, which are intended to stand in a one-to-one relation with types of natural language expressions and their subparts. The role of the linguistic theory is to give a precise specification of which feature structures are to be considered admissible; the types of linguistic entities which correspond to the admissible feature structures constitute the predictions of the theory. (HPSGII, page 8)

The sorted feature structures chosen as models are graph theoretic entities. Pollard and Sag

[6] The setup of Pollard and Sag (1987) differs in several respects from that of HPSGII.

[7] Note that the term *model* has two usages: A mathematical entity which stands in a one-to-one relation to an object of the domain is called a model of that object. The second usage, which is avoided in this paper to prevent confusion, is to speak of the model of a theory as the set of "things" which are admitted by the theory. This double use of model is especially confusing, since the "things" in the formulation of the second usage, are often the models of objects in the first sense of the word.

require them to be *totally well-typed*[8] and *sort-resolved*[9]. The reason for this is that feature structures that serve as models of linguistic entities are required to be total models of the objects that they represent.[10] To get the full picture of the setup proposed, we need to answer two more questions:

How is a theory formulated? The theory is formulated using a specific description language.[11] The description language allows us to express that a feature structure has to have a certain type, that the value of an attribute of a feature structure has to have a certain type, or that the values of two attributes of a feature structure are *token identical*[12]. Additionally conjunction, disjunction and full negation are defined as operators on the descriptions.

What does it mean for a theory to specify admissible feature structures? A theory consists of a set of descriptions which are interpreted as being true or false of a feature structure in the domain. A feature structure is admissible with respect to a certain theory iff it satisfies each of the descriptions in the theory and so does each of its substructures. The descriptions which make up the theory are also called *constraints*, since these descriptions constrain the set of feature structures which are admissible with respect to the theory compared to the domain of feature structures specified by the signature. Note that the term constraint has been used for many different purposes. I this paper "constraint" will only be used for "description which is part of the theory".

The setup of HPSGII is summed up in figure 1. The usual symbols for description language operators are used: conjunction ($\wedge$), disjunction ($\vee$), negation ($\neg$), and implication ($\rightarrow$). Additionally, type assignment, path equality, and path inequality are noted as "$\sim$", "$=$", and "$\neq$", respectively.

$$\underset{(\sim, = \ | \ \wedge, \vee, \neg)}{\text{Descriptions}} \xrightarrow[\textit{Interpretation}]{} \text{Sorted feature structures} \xrightarrow[\textit{One-to-one Correspondence}]{} \text{Abstract linguistic object}$$

Figure 1: The setup desired in HPSGII

Several logics have been proposed to provide the formal foundations for this setup. Basically

[8] *Totally well-typed* means that a) "what attribute labels can appear in a feature structure is determined by its sort; this fact is the reflection within the model of the fact that what attributes ... an empirical object has depends on its ontological category." and b) "every feature that is appropriate for the sort assigned to that node is actually present."(HPSGII, p. 18)

[9] "A feature structure is *sort-resolved* provided every node is assigned a sort label that is ... most specific in the sort ordering."(HPSGII, p. 18)

[10] It is not uncontroversial whether total models of complete linguistic objects are what is needed for linguistics. Problems for such total models arise for example in the analysis of coordination (cf. the weak coordination principle and fn. 14 on p. 400 of HPSGII).

[11] In HPSGII the term *attribute-value matrix (AVM)* is used instead of *description*. For easier comparison of the different setups, we will only use the term *description*. Furthermore, *attribute* and *feature* are used synonymously throughout the paper. The same is true for *type* and *sort*, except in the use of *typed feature structure* (cf. Carpenter (1992)) vs. *sorted feature structure* (cf. HPSGII).

[12] The values of two attributes are *token identical* iff the two paths point to the same node. The other common identity notion is *type identity*: Two feature structures are *type identical* iff they are of the same type, the same attributes are defined on both feature structures, and the feature structures which are the values of the attributes are *type identical*.

there are two families of logics dealing with this task: the *Kasper-Rounds* logics[13] and the *Attribute-Value* logics[14]. Two representatives of these families are of particular interest for the following discussion, since they also have been used to construct computational systems for the implementation of HPSG grammars: the Kasper-Rounds logic defined in Carpenter (1992) on which the ALE system is based, and the Attribute-Value logic of King (1989) which underlies the Troll system. Figure 2 shows the setup proposed in Carpenter (1992). The descriptions of Carpenter (1992) describe *typed feature structures*, which model *partial information*.

$$\underset{(\sim, =, \neq \ \mid\ \wedge, \vee)}{\text{Descriptions}} \quad \underset{\textit{Interpretation}}{\longrightarrow} \quad \text{Typed feature structures} \quad \underset{\substack{\textit{One-to-one}\\ \textit{Correspondence}}}{\longrightarrow} \quad \text{Partial information}$$

Figure 2: The setup of Carpenter (1992)

This setup differs from the HPSGII desideratum described above. Carpenter (1992) uses typed feature structures to model *partial information*. Presumably there is a further level, in which the partial information is related to the linguistic objects, which is left undefined. Since the typed feature structures model partial information they cannot be total representations like the sorted feature structures of HPSGII. The typed feature structures used as models in Carpenter (1992) are therefore not required to be totally well-typed or sort-resolved. In the HPSGII setup "partial information" plays no role; the linguistic objects themselves are modelled by the sorted feature structures. This difference has consequences for the description language of Carpenter (1992). Moshier and Rounds (1987) show that in the setup of a Kasper-Rounds logic, full classical negation as part of the description language destroys subsumption monotonicity on typed feature structures which Carpenter and others claim has to be upheld if feature structures are to model partial information. In the discussion of recursive type constraints in chapter 15 of Carpenter (1992) (p. 233), Carpenter states that even for the formulation of implicative constraints with type antecedents (i.e. type negation) subsumption monotonicity cannot be achieved.[15] The description language of Carpenter (1992) therefore only contains path-inequations, a weaker form of negation.

$$\underset{(\sim, = \ \mid\ \wedge, \vee, \neg)}{\text{Descriptions}} \quad \underset{\textit{Interpretation}}{\longrightarrow} \quad \text{Concrete linguistic objects}$$

Figure 3: The setup of King (1989)

The setup of King (1989), displayed in figure 3, at first sight also differs from the HPSGII desideratum. The most apparent difference is the missing model level. The descriptions are directly interpreted as being true or false of linguistic objects, not of feature structures which stand in a one-to-one relation with linguistic objects, as in HPSGII. Apart from the philosophical consequences (which will not be discussed here) this gap between HPSGII and King (1989) can

[13] Cf. Rounds and Kasper (1986), Moshier and Rounds (1987), and Carpenter (1992).

[14] Cf. Johnson (1988), Smolka (1988), and King (1989).

[15] Note that the theories in an HPSGII architecture are formulated using implications on types; in fact, even stronger implicative statements with complex antecedents are used. We will see on pp. 15 of section 3.2.1 how certain stronger implicative statements can be reformulated as implications with type antecedents by encoding additional information in the signature.

be bridged. Since the models of linguistic objects in the first approach and the linguistic objects in the second are both total (i.e. the sorted feature structures of Pollard and Sag are defined that way and the linguistic objects of King are total since they are objects) there are no formal consequences to the dropping of the modelling level in King (1989). As proof of this King (1994) shows that a modelling level can be introduced without changing the rest of the logic. The second difference between the HPSGII desideratum and the King logic is that HPSGII talks about *abstract* linguistic objects while the King setup uses *concrete* linguistic objects[16]. King (1994) shows how this gap can be bridged as well. The logic proposed in King (1989) therefore can be used to provide the setup desired in HPSGII.

The difference between the two logics becomes clearer, when we take a closer look at how descriptions are interpreted in each setup.

Syntax	$\xrightarrow[Interpretation]{}$	**Semantics**
conjunction of descriptions	–	set intersection
disjunction of descriptions	–	set union
negation of descriptions	–	set complement

Figure 4: Set theoretic interpretation of description language expressions in King (1989)

In King (1989) descriptions are given a set theoretic interpretation: The interpretation of a description is a set of objects. Conjunction, disjunction, and negation as operations on descriptions are interpreted as set intersection, set union, and set complement, respectively. In such a set theoretic setup, an object satisfies a description iff it is in the denotation of that description.

Syntax	$\xrightarrow[Interpretation]{}$	**Semantics**
conjunction of descriptions	–	unification
disjunction of descriptions	–	eliminated by lifting disjunction to top level

Figure 5: Interpretation of description language expressions in Carpenter (1992)

In Carpenter (1992), the interpretation of a description is a set of typed feature structures. Conjunction of descriptions is interpreted as unification of feature structures and disjunction is eliminated by lifting it to the top level. The satisfaction relation is defined directly between feature structures and descriptions without using set theoretic denotations (cf. p. 53 of Carpenter (1992)).

Now that we have introduced the two logics, we can turn to the computational systems which are based on them. The ALE system uses the setup and the description language of Carpenter (1992). Additionally feature structures in ALE are required to be totally-well-typed. The main difference

[16] Abstract linguistic objects are also called linguistic "types", and the term linguistic "tokens" is used for concrete linguistic objects. This usage of the term "type" has nothing to do with the types as part of the signature of a typed feature logic. In order to avoid confusion, the terms "types" and "tokens" meaning abstract and concrete will not be used in this paper.

between Carpenter (1992) and the ALE system is that theories in ALE are expressed using a relational extension[17] of the description language, which is outside of the defined feature logic.[18]

The King (1989) logic supplies the Troll system with an interpretation of a signature and a description language which is adequate for HPSG II. However, there are some differences between the logic and the computational system: Just as in the ALE system, a relational extension of the description language is used to express linguistic theories. Also, currently only the conjunctive part of the description language is part of Troll. As supplement to the King setup, typed feature structures are introduced as normal form descriptions. Note that the typed feature structures here are part of the syntax, not of the semantics as in the Carpenter approach.

There are several consequences of the theoretic and computational setups just introduced for some commonly used terminology. When two descriptions are conjoined in Troll, *unification* is used to calculate a normal form description, the denotation of which is the intersection of the denotations of the descriptions conjoined; i.e. unification is an operation on the syntax and plays no theoretic role. In ALE, unification is the interpretation of the conjunction of descriptions, i.e. it is the semantics of a syntactic operation.

The same division into syntax and semantics applies to *subsumption*. In ALE, subsumption is a relation that makes statements about the complexity of the feature structures serving as semantic representation. If we try to add subsumption to the King setup (or that of HPSG II), we note that subsumption does not make sense as relation on objects (or the sorted feature structures) which are on the semantic side, since these are total representations. The only place for subsumption in the King setup therefore is on the syntactic side as a relation on descriptions. The interpretation of the subsumption relation on descriptions is the subset relation on the denotations of the descriptions. Note, however, that the subsumption relation is no proper part of the description language of the logic of King (1989). Summing up the last two paragraphs, neither unification nor checking for subsumption is part of the description languages of the two logical setups. In the discussion of the lexical rule mechanism in section 4.1.3 we will come back to this.

In the rest of the paper we will assume the ontological setup and terminology of King (1989). The following feature logic terminology is used throughout the paper: $\top$ is the least constrained type denoting all objects; $\perp$ is the inconsistent type denoting the empty set. *Minimal types* are the types which have no subtypes except for $\perp$, e.g. the most specific types, sometimes also called *varieties* or *species*. Since nothing of theoretical interest relies on the feature structures as normal form descriptions, feature structures will not appear on stage in the rest of the paper. Only when specifically speaking about the logic of Carpenter (1992) or the ALE system, will the Carpenter setup and its terminology be used.

3 The HPSG architecture: A linguist's view on the feature logic toolkit

After the ontological primer, we can now go on to discuss the HPSG architecture based on this setup. An HPSG theory consists of two ingredients: the declaration of the domain of linguistic

[17]For a discussion and the formal definition of a relational extension of a constraint language cf. Jaffar and Lassez (1987) and Höhfeld and Smolka (1988).

[18]The different possible ways of expressing a theory are discussed in section 3.2.

objects in a signature (consisting of the type hierarchy and the appropriateness conditions) and the formulation of constraints on that domain[19]. The perspective taken in the following discussion is always that of a linguist wanting to know what consequences each formal setup has on the formulation and implementation of an HPSG-style grammar. The reader interested in the mathematical definition and properties of the architectures is referred to the feature logic literature, in particular King (1989), Carpenter (1992), King (1994), and Götz (1994).

3.1 Declaring the domain of a grammar

From the viewpoint of linguistics, the signature introduces the domain of objects the linguist wants to talk about. The theory the linguist proposes for a natural language distinguishes between those objects in the denotation which are part of the natural language, and those which are not. First, we will take a look at the signature. The possibilities for writing theories are dealt with in the next section.

The signature serves as the data structure declaration for the linguistic theory. It consists of a type hierarchy and appropriateness conditions. The type hierarchy introduces the classes of objects the grammar makes use of, allowing us to refer to the classes by using the types as names. The appropriateness declarations define which class of objects has which attributes with which values.

At which types attributes can be introduced and the exact interpretation of the type hierarchy is subject of debate. Two interpretations of a type hierarchy are common: ALE is based on an open world interpretation, while HPSG theory and Troll use a closed world interpretation.

Briefly said, a closed world interpretation makes two assumptions: every object in the denotation of a non-minimal type is also described by at least one of its subtypes; and every object is of exactly one minimal type. An open world interpretation makes no such assumptions.

Once we commit ourselves to a closed world interpretation, more syntactic inferences can be made. While both interpretations allow the inference that appropriateness information present on a type gets inherited to its subtypes, we can now additionally infer the appropriateness specifications on a type from the information present on its subtypes.

In the implementation, the information resulting from these additional inferences allows us to specify less information by hand and enables us to express constraints on the interdependence of attribute specifications.[20] Besides the gained expressive power, this leads to syntactic detection of more errors as well as an increase in efficiency.

The following example from the implemented grammar illustrates the additional expressive power gained in a closed world interpretation. We want to capture that non-finite verbs are

[19]The different linguistic motivations behind the constraints (e.g. grammatical principles, lexical information) and the possible ways for expressing these are discussed in section 4.

[20]Note the correspondence with the so-called *feature cooccurrence restrictions* of Gazdar et al. (1985). Cf. Gerdemann and King (1993) and Gerdemann and King (1994) for a detailed investigation of this issue.

never inverted.

$$\begin{bmatrix}\text{VFORM} & \textit{n\_fin}\end{bmatrix} \rightarrow \begin{bmatrix}\text{INV} & \textit{minus}\end{bmatrix}$$

Figure 6: The implication to be captured

It is not possible to directly encode the intended relationship between VFORM and INV in the signature of the feature logics introduced. However, under a closed world interpretation the regularity can be encoded in a signature with a slightly modified type hierarchy.[21] Figure 7 shows the type hierarchy below *verb*. Two new subtypes are introduced.[22]

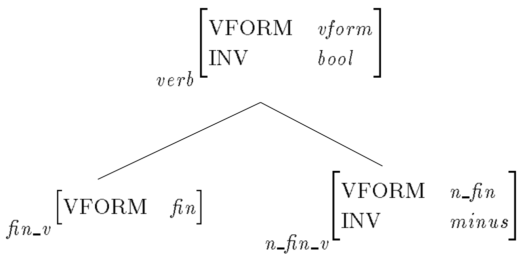

Figure 7: The subtypes of *verb* (a subtype of *head*)

Under a closed world interpretation the following inferences hold.

$$\begin{array}{lll}
\text{(a)} & \textit{fin_v} & \leftrightarrow & \begin{bmatrix}\text{VFORM} & \textit{fin}\end{bmatrix} \\
\text{(b)} & \textit{n_fin_v} & \leftrightarrow & \begin{bmatrix}\text{VFORM} & \textit{n_fin}\end{bmatrix} \\
\text{(c)} & \textit{n_fin_v} & \rightarrow & \begin{bmatrix}\text{INV} & \textit{minus}\end{bmatrix} \\
\text{(d)} & \begin{bmatrix}\text{INV} & \textit{plus}\end{bmatrix} & \rightarrow & \begin{bmatrix}\text{VFORM} & \textit{fin}\end{bmatrix} \\
\text{(e)} & \begin{bmatrix}\text{VFORM} & \textit{n_fin}\end{bmatrix} & \rightarrow & \begin{bmatrix}\text{INV} & \textit{minus}\end{bmatrix}
\end{array}$$

Figure 8: Some inferences under a closed world interpretation

The originally desired implication repeated as (e) is deduced from (b) and (c).

[21] The method used here for encoding implicative statements with complex descriptions as antecedent in the signature is discussed in detail in section 3.2.1.

[22] Once the subtypes of *verb* are introduced, the original VFORM attributes could be omitted without loss of information. However, this would change the original type hierarchy (with its linguistic motivation) even more, making it necessary to reformulate every reference to the verb-form in the whole grammar. In any case, this issue does not change anything regarding the fact that the implication (d) in figure 8 will only be obtained under a closed world interpretation.

Under an open world interpretation only the inferences in figure 9 are made:

$$\begin{array}{llll} (a) & \textit{fin\_v} & \rightarrow & \left[\text{VFORM} \quad \textit{fin}\right] \\ (b) & \textit{n\_fin\_v} & \rightarrow & \left[\text{VFORM} \quad \textit{n\_fin}\right] \\ (c) & \textit{n\_fin\_v} & \rightarrow & \left[\text{INV} \quad \textit{minus}\right] \end{array}$$

Figure 9: The inferences under open world interpretation

Compared to the inferences made under a closed world interpretation, under an open world interpretation the "←" direction in (a) and (b) is missing . Therefore the originally desired implication (figure 6) cannot be deduced. Additionally the inference (d) of figure 8 is missing. To still get the desired results one therefore has to ensure the "←" direction of (a) and (b) by hand; i.e. additional specifications on the lexical entries and the principles are necessary. The same is true for the implication (d). Under an open world interpretation the full set of inferences of figure 8 (which include the originally desired implication of figure 6) therefore only hold for the descriptions bearing the additional specifications; i.e. under an open world interpretation it is not possible to ensure that the full set of inferences will always be made for every object. Regarding the theoretical level, this can lead to incorrect analysis. In the computational systems based on an open world interpretation (e.g. ALE) this has the effect that the desired inferences will not be made for feature structures resulting in processing. The missing inferences in such computational systems therefore result in possibly incorrect, but in any case slower analysis.

The second variation in the concept of a signature concerns the appropriateness conditions. The feature introduction condition (FIC) of ALE demands that for every feature there is a unique least specific type at which this feature is introduced.

In the grammar implemented this turns out to be troublesome. As an illustration let us look at the signature definition of *head* objects. Figure 10 shows the type hierarchy as motivated by the linguistic theory.[23]

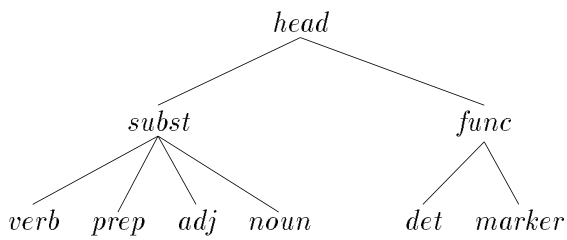

Figure 10: The type hierarchy below *head*

The problem arises with the introduction of the case and declension attributes, since CASE and DECL are appropriate for adjectives, nouns, and determiners, but not for markers and verbs. To satisfy the feature introduction condition, a new type *adj_noun_det* as subtype of *head* and with subtypes *adj*, *noun*, and *det* needs to be introduced for which the two attributes CASE and

[23] As before and throughout the paper, the most general type is on top. Note that Carpenter's hierarchies are drawn the other way around: his most general type is at the bottom.

DECL are appropriate.

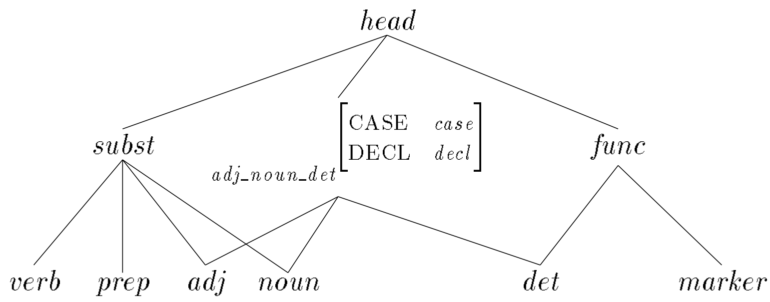

Figure 11: A first step towards satisfying the FIC

However, in the resulting structure in figure 11 the types *subst* and *adj_noun_det* do not have a unique greatest lower bound. This violates the formalization of a type hierarchy as *finite bounded complete partial order* (cf. Carpenter92). Therefore, an additional type *adj_or_noun* needs to be introduced. Figure 12 shows the final type hierarchy together with the appropriateness information for *adj_noun_det*.[24]

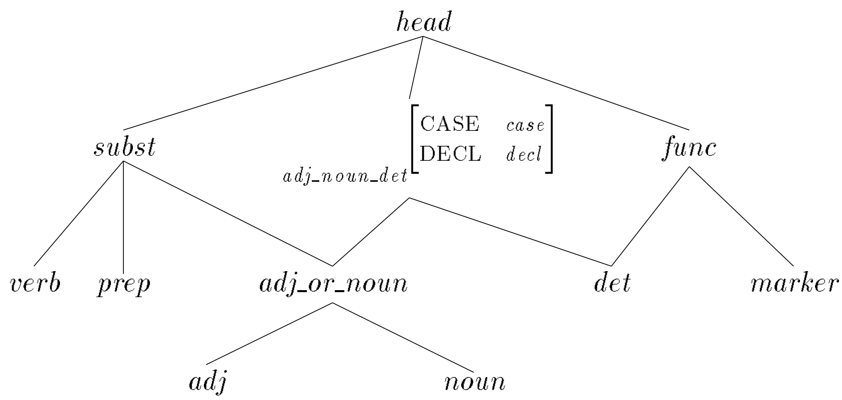

Figure 12: A type hierarchy satisfying the FIC for CASE and DECL

The feature introduction condition is part of the Carpenter (1992) logic and the ALE system. It plays no role in the logic of King (1989) or the Troll system.[25]

[24]The type hierarchy actually used in the implementation is more complicated still, since a type *non_noun* as subtype of *head* has to be introduced. This is necessary to introduce a type for lists of nominal signs, which is needed as constraint on the auxiliary entries and the output of the PVP extraction lexical rule in HN. Cf. section 4.1.3, p. 37 for a discussion.

[25]King and Götz (1993) show that the feature introduction condition can also be eliminated within the logic of Carpenter (1992).

3.2 Constraining the domain

Now that we have seen how the domain of linguistic objects is defined by the signature, we are ready to discuss how the theory, i.e. the constraints making up the grammar can be expressed. The theory divides the domain of objects specified by the signature into admissible ones, which are part of the natural language described, and those which are not admissible, i.e. not part of the natural language. In linguistic terms the constraints making up the grammar are the principles (e.g. the Immediate Dominance Principle (IDP), the Head Feature Principle (HFP), the Nonlocal Feature Principle (NFP), the Semantics Principle (SemP)) and the specification of the lexicon. The architectures of HPSGII and the various systems differ considerably regarding the question of how the constraints are expressed. Basically there are two main groups: those description languages directly constraining the domain of linguistic objects, and those in which relations on linguistic objects are expressed.

3.2.1 Group 1: Directly constraining the domain of linguistic objects

In HPSGII and the TFS system the grammatical constraints are expressed as statements about the domain of linguistic objects. The simplest implicative form (henceforth called *type definition*) is depicted in figure 13. It has the following interpretation: if something is of type t, then it has to satisfy the description D.

$$t \quad \rightarrow \quad D$$

Figure 13: The type definition: a simple implicative constraint

In addition to the type definitions, HPSGII also uses a stronger implicative statement of the form displayed in figure 14.

$$C \quad \rightarrow \quad D$$

Figure 14: The object definition: an implicative statement with possibly complex antecedent

The intended interpretation is that if something satisfies the possibly complex description C it also has to satisfy D. We will call the implicative constraints (including the type definitions) *object definitions*. In writing a grammar one usually wants to impose constraints on a certain group of objects (those in the denotation of the antecedent) and not on all objects (in which case a non-implicative statement would do). Therefore, object definitions are the main building blocks of linguistic theories in the HPSG architecture. As an example, take the HFP as formulated in the appendix of HPSGII:

$$\underset{phrase}{\left[\text{DTRS } \textit{headed-struc}\right]} \rightarrow \begin{bmatrix}\text{SYNSEM|LOC|CAT|HEAD} & \boxed{1} \\ \text{DTRS|HEAD-DTR|SYNSEM|LOC|CAT|HEAD} & \boxed{1}\end{bmatrix}$$

Figure 15: The Head-Feature Principle of HPSGII

The complex description $\underset{phrase}{\left[\text{DTRS headed-struc}\right]}$ is the antecedent of the implication. Note that using the type *phrase* as antecedent is not enough, since the HFP is only supposed to apply to phrases dominating headed structures. The HFP constraint demands that all objects of type *phrase* which have a DTRS attribute with a *headed-struc* object as value, also have to be described by the consequent, i.e. satisfy the description that the HEAD values of the object and that of its head daughter are token-identical.

A closer look at the antecedents of implicative constraints

To get an idea of the complexity of the description language, we need to distinguish between different kinds of antecedents of implicative constraints. Figure 16 displays three classes of descriptions, which can function as antecedents of implicative statements in the order of increasing complexity.

1. simple type description (e.g. t)
2. complex description with type assignments (e.g. $\underset{t}{\begin{bmatrix}\text{X} & \underset{u}{\begin{bmatrix}\text{Y} & \text{v} \\ \text{Z} & \text{w}\end{bmatrix}}\end{bmatrix}}$)
3. complex description with type assignments and path equalities (e.g. $\underset{t}{\begin{bmatrix}\text{X} & \boxed{1}u \\ \text{Y} & \boxed{1}\end{bmatrix}}$)

Figure 16: Three classes of descriptions

The simplest implicative statements are the type definitions, which only allow types as antecedent (class 1). One step up in complexity, complex feature structures which specify no structure sharing can be used as antecedents (class 2). Finally, in the most complex type of implications the antecedents are allowed to specify structure sharing information as well (class 3).

The implicative statements in HPSGII are mostly of class 2. However none of the computational systems allow the user to specify complex antecedents; only type definitions are supported.[26]

[26] This also applies to the systems in which a theory is expressed on the relational level (the group 2 setups) since the method described in section 3.2.2 for translating theories which are expressed as constraints on the domain (group 1) into theories which are expressed on the relational level (group 2) is restricted to type definitions as well. Therefore, until other general techniques for expressing HPSGII-style theories on the relational level are defined which might allow us to express implicative constraints with complex antecedents, both computational setups are restricted to type definitions.

Because of this discrepancy in expressive power, it is interesting to note that the implicative statements with complex antecedents used in HPSGII can be expressed as type definitions by modifying the signature, i.e. a new type can be introduced so that it has the same denotation as a given class 2 description. To see how this encoding works, let us first take a closer look at the appropriateness conditions (ACs).

In section 3.1 we had characterized ACs as part of the data structure declaration, the declaration of the domain in more logic based terminology. This means, the ACs take part in defining the domain over which the theory is formulated. The objects not satisfying the ACs are not in the domain. On the other hand, ACs can also be understood as rather weak constraints on the domain of objects as defined by the type hierarchy alone.[27] They restrict the values of the immediate attributes of a type without specifying structure sharing. Under this interpretation, the domain over which the theory makes statements also contains objects which do *not* satisfy the AC. However, while the one-level-only condition does not restrict the set of descriptions which can be encoded in the signature definition of a type, the no-structure-sharing condition is a real restriction. Descriptions specifying structure sharing cannot be encoded in a type by modifying the signature only. To encode a class 3 description in a type, one therefore additionally has to specify a constraint on the new type as part of the theory in order to express the structure sharing conditions on its attributes.[28]

Encoding the information of a class 2 description D in the signature works in the following way: A new type is introduced as subtype of the type of D.[29] The type specifications on the first level of attributes of D are encoded in the ACs of the new type. If D contains descriptions of further levels of attributes, new subtypes have to be introduced to mediate the information. Note that this introduction of additional types for each additional level quickly leads to an explosion of the type hierarchy.

To illustrate this mechanism let us look at an example from HN.[30] In HN every auxiliary is supposed to bear an argument raising specification, which in its basic form is shown in figure 17.

$$\begin{bmatrix}\text{COMPS} & \left\langle \begin{bmatrix}\text{SYNSEM|LOC|CAT} & \begin{bmatrix}\text{HEAD} & \textit{verb}\\ \text{VAL|COMPS} & \boxed{1}\end{bmatrix}\end{bmatrix} \middle| \boxed{1} \right\rangle\end{bmatrix}$$

Figure 17: Argument raising specification on a non-finite auxiliary (*valence* value shown)

[27] Note that ACs are inherited from a type to its subtypes, allowing generalizations to be expressed in a nice hierarchical way independent of how the theory is formulated.

[28] So far no HPSG theory to our knowledge makes use of class 3 antecedents for implicative statements. This means that currently only the class 2 antecedents, which can be encoded without manipulating the theory, i.e. by changing the signature only, are linguistically motivated.

[29] A complication arises, if D describes objects of different types. In this case, a new subtype has to be introduced for each of the types of object described by D.

[30] For this and the following examples I assume the signature of HN and the grammar implemented. It closely resembles the one in the appendix of HPSGII with the modifications of Chapter 9. However, note that all valence attributes (e.g. COMPS) take a list of signs as their value, not a list of synsems. The order of the valence lists of HN is inverted in the grammar implemented, i.e. valence lists look like ⟨*most-oblique-element* … *least-oblique-element*⟩. This order is used in the examples of this paper as well. A discussion of the motivation behind this change is given in section 5.1.

There are two options for encoding this in the grammar. Since we are dealing with lexical specifications, the methods for expressing generalizations over classes of lexical entries could be used. These are dealt with in section 4.1. The other possibility for attaching the argument raising specification in figure 17 to auxiliaries is to use the general mechanism for expressing grammatical constraints. To do this, the following object definition can be added to the grammar:

$$
{}_{\mathit{word}}\left[\text{SYNSEM|LOC|CAT|HEAD} \quad {}_{\mathit{verb}}\left[\text{AUX} \quad \mathit{plus}\right]\right] \rightarrow
$$
$$
\left[\text{SYNSEM|LOC|CAT|VAL|COMPS} \quad \left\langle \left[\text{SYNSEM|LOC|CAT} \quad \left[\begin{array}{ll}\text{HEAD} & \mathit{verb} \\ \text{VAL|COMPS} & \boxed{1}\end{array}\right]\right] \,\middle|\, \boxed{1} \right\rangle\right]
$$

Figure 18: An object definition to attach the argument raising specification

To rewrite this object definition as type definition, a new subtype of *word* (call it *aux-word*) needs to be introduced. We want it to have the same denotation as the complex antecedent of the implication in figure 18. Assuming the signature definitions given in the appendix of HPSGII as basis, under a closed world interpretation the following flood of signature changes and new definitions is necessary to achieve that.

a) *word* > *non-aux-word*, *aux-word*.
   i. ${}_{\mathit{non\text{-}aux\text{-}word}}\left[\text{SYNSEM} \quad \mathit{non\text{-}aux\text{-}synsem}\right]$.
   ii. ${}_{\mathit{aux\text{-}word}}\left[\text{SYNSEM} \quad \mathit{aux\text{-}synsem}\right]$.

b) *synsem* > *non-aux-synsem*, *aux-synsem*.
   i. ${}_{\mathit{non\text{-}aux\text{-}synsem}}\left[\text{LOC} \quad \mathit{non\text{-}aux\text{-}loc}\right]$.
   ii. ${}_{\mathit{aux\text{-}synsem}}\left[\text{LOC} \quad \mathit{aux\text{-}loc}\right]$.

c) *loc* > *non-aux-loc*, *aux-loc*.
   i. ${}_{\mathit{non\text{-}aux\text{-}loc}}\left[\text{CAT} \quad \mathit{non\text{-}aux\text{-}cat}\right]$.
   ii. ${}_{\mathit{aux\text{-}loc}}\left[\text{CAT} \quad \mathit{aux\text{-}cat}\right]$.

d) *cat* > *non-aux-cat*, *aux-cat*.
   i. ${}_{\mathit{non\text{-}aux\text{-}cat}}\left[\text{HEAD} \quad \mathit{non\text{-}aux\text{-}head}\right]$.
   ii. ${}_{\mathit{aux\text{-}cat}}\left[\text{HEAD} \quad \mathit{aux\text{-}verb}\right]$.

e) *head* > *subst*, *non-aux-head*.

f) *subst* > *verb*, *non-aux-subst*.

g) *non-aux-head* > *non-aux-subst*, *func*.

h) *non-aux-subst* > *non-aux-verb*, *prep*, *adj*, *noun*.

i) *verb* > *non-aux-verb*, *aux-verb*.

i. $_{non\text{-}aux\text{-}verb}\big[\text{AUX} \quad minus\big]$.

ii. $_{aux\text{-}verb}\big[\text{AUX} \quad plus\big]$.

Figure 19: Additional signature definitions to introduce the type *aux-word*

Two subtypes are introduced for each of the types *word*, *synsem*, *loc*, and *cat* in order to mediate the AUX specification from the *sign*-level to the *head* level. The hierarchy below *head* is more complicated. For easier comprehension it is displayed in graphical notation below. The corresponding type hierarchy of HPSGII from which this new hierarchy is derived, was already presented in figure 10.

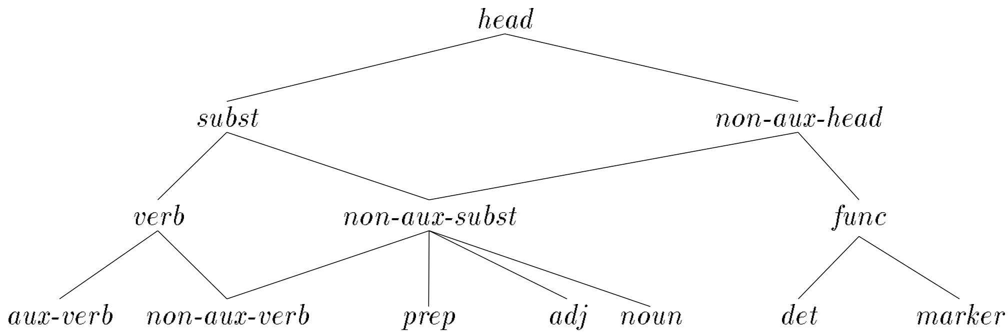

Figure 20: The resulting type hierarchy below *head*

Clearly these linguistically unmotivated changes to the type hierarchy in order to lift information to the "surface", i.e. the top level of an implication, is nothing that can be asked of a linguist encoding a grammar. One could therefore consider eliminating the hierarchical structure of the models of linguistic objects and return to simpler, flat models. Until there is linguistic evidence against the hierarchical structures[31] it is better motivated though, to extend the computational systems so that complex antecedents can be expressed directly. Introducing general negation into the formalism, would allow any complex antecedent to be expressed directly. However, while this might turn out to be possible in some underlying logics, losing the type backbone to the constraints (the type antecedents as "triggers" for the implicative constraints) causes significant computational problems and no system currently provides such an architecture. In this situation, the technique introduced above for encoding complex descriptions in types looks like a valuable method after all. An automated version of the encoding process could be included in a compilation step applying after the writing of the grammar. Thus the linguistically unmotivated mutation of the signature would also be hidden from the eyes of the linguist.

[31] We here ignore the problematic issue of lexical hierarchies (cf. Chapter 8 of Pollard and Sag (1987)). A short discussion of these hierarchies is included in section 4.1.2.

Recursivity

After looking at the antecedents of implicative constraints, we now turn to the consequent of the implications to see where the expressive power to make recursive statements (necessary to work with lists and sets) comes from.

TFS allows a more complex version of type definitions, which is displayed in figure 21.

$$t \quad \rightarrow \quad (D \ \& \ E_1 \ \& \ \ldots \ \& \ E_n)$$

Figure 21: The extended type definitions of TFS

Just as with simple type definitions, the extended type definition above is true of those objects of type t which are also described by D. Additionally, each of the new descriptions E_i appended with "&"[32] have to be satisfiable. In other words: there must be objects in the denotation of each description E_i. These extended type definitions enable us to state recursive definitions, since each of the objects in the denotation of E_i has to satisfy the constraints imposed on its type. If an E_i describes objects of the type t that is being defined (= the antecedent of the implication), one obtains a recursive statement.

It is possible to make sense of the meta operator operator "&" within the description language. Following Aït-Kaci (1985), one can add extra attributes ("junk slots") to linguistic objects to "store" additional objects. The descriptions which were added using the extra-logical "&" operator now only have to be added to the theory as descriptions of the objects residing in the newly introduced attributes.[33] Applying this junk slot encoding technique to the definition in figure 21, we introduce a list valued attribute STORE as appropriate for type t. The resulting description language encoding of figure 21 is displayed in figure 22.

$$t \quad \rightarrow \quad (D \wedge \left[\text{STORE} \quad < E_1, \ldots, E_n >\right])$$

Figure 22: Extended type definitions expressed using the description language only

Beside using the additional descriptions housed in the junk slot attribute to describe ordinary linguistic objects, one can also use statements like that in figure 22 to formulate recursive relations on linguistic objects at the same level at which the objects are described. The additional junk slot attribute then also allows calls to relations to be attached to descriptions of linguistic objects without the use of a meta operator "&". Note that when a recursively defined relation is used as description of an object's attribute value, the object will bear the full proof-tree in the attribute.

As example for the definition of a recursive relation at the level of the description language, the

[32] The "&" is not to be confused with the conjunction "$\wedge$" of the description language, which is interpreted as set intersection of the denotations of the conjuncts.

[33] This applies to the TFS system as well. The "&" operator in TFS can therefore be seen as surface syntax for the user making it unnecessary to worry about introducing additional attributes as junk slots to store the additional descriptions which one wants to test for satisfiability.

definition of the standard *append* relation on lists[34] is shown in figure 23.

$$append \quad \rightarrow \quad \left(\begin{bmatrix} \text{ARG1} & <> \\ \text{ARG2} & \boxed{1} \\ \text{ARG3} & \boxed{1} \\ \text{STORE} & <> \end{bmatrix} \vee \begin{bmatrix} \text{ARG1} & < \boxed{1} \mid \boxed{2} > \\ \text{ARG2} & \boxed{3} \\ \text{ARG3} & < \boxed{1} \mid \boxed{4} > \\ \text{STORE} & < {}_{append}\begin{bmatrix} \text{ARG1} & \boxed{2} \\ \text{ARG2} & \boxed{3} \\ \text{ARG3} & \boxed{4} \end{bmatrix} > \end{bmatrix} \right)$$

Figure 23: Defining append objects

The constraint in figure 23 defines what kind of append objects are admissible. The description "*append*" (i.e. a call to append, possibly as part of some other description) denotes a set of append objects having their attributes constrained as defined.

While HPSGII is not committed to a certain way of expressing recursive statements, an alternative possibility often found in the HPSG literature is to use definite clauses as a relational extension of the description language. Note that in contrast to the encoding of relations discussed above, the use of definite clauses described here is outside of the feature logic and it is not formally defined. The definitions of the definite clauses are allowed to be recursive, yielding the expressive power we were looking for. To use the relations thus defined, the calls to the definite clauses are attached to the implicative statements just like the additional descriptions are in TFS.

$$C \quad \rightarrow \quad (D \ \& \ rel_1(E_{11}, \ldots, E_{1i}) \ \& \ \ldots \ \& \ rel_n(E_{n1}, \ldots, E_{nj})).$$

Figure 24: A type definition with attached relational constraints[35]

Figure 24 only shows the use of relations, not their definition. Contrary to the setup used in TFS, the definition of the relation itself is specified on a level different from the level of the grammatical constraints defined in figure 24. Because the definite clause relations are not part of the description language, the calls to the definite clauses need to be added to the descriptions using the extra-logical operator "&". Note that for the same reason it is not possible to include the calls to the definite clause relations in a description as it was done with the help of the junk slot technique for the relations formulated within the description language. Definite clauses are defined as in PROLOG, the only difference being that while in PROLOG definite clauses over first order terms are formulated, here definite clauses over feature descriptions are used. Figure 25 shows the definition of the *append* relation as part of the relational extension of the

[34]The *append* relation specifies the third argument to be the concatenation of the two lists passed as first and second argument.

[35]C, D, and the E_{xy} are descriptions.

description language.

a) append(<>, $\boxed{1}$, $\boxed{1}$).

b) append(< $\boxed{1}$ | $\boxed{2}$ >, $\boxed{3}$, < $\boxed{1}$ | $\boxed{4}$ >) := append($\boxed{2}$, $\boxed{3}$, $\boxed{4}$).

Figure 25: Defining the append relation

Here *append*, being a three place relation, denotes the set of triples of list objects, which satisfy the definition given.

This ends the discussion of the first group of setups, in which the theories directly constrain the domain.

3.2.2 Group 2: Expressing relations on linguistic objects

The second group – represented by systems like ALE, CUF, and Troll – takes the concept of the relational extension of the description language just described, and makes it the only way to express the grammatical constraints of the theory. Relations over objects are formulated. Unlike in the group 1 setups, the objects cannot be constrained directly.[36]

The result is a setup with a clean distinction between the description language, and the relational extension of that language: the description language only serves to describe the objects which can be arguments of the relations. Figure 26 illustrates this setup.

$$rel_0(E_{01}, \ldots, E_{0i}) \quad :- \quad rel_1(E_{11}, \ldots, E_{1j}),\ \ldots,\ rel_n(E_{n1}, \ldots, E_{nk})$$

Figure 26: Two levels of constraints in a definition of the relation rel_0

Note that the question of how the constraints making up the theory are expressed is a separate issue from the formulation of a signature discussed in section 3.1. Both, group 1 style and group 2 style theories are based on an underlying signature. However, as discussed in the previous section, the appropriateness conditions of the signature can also be seen as expressing weak constraints on a domain defined by the type hierarchy only. Therefore, even in a group 2 system, i.e. in a system in which the theory is expressed exclusively on the level of a relational extension of the description language outside of the feature logic defined, some grammatical constraints can be expressed inside of the logic by encoding them in the signature.

Definite clauses are not the only relevant way to express relations. ALE and Troll offer a parser, which allows the formulation of phrase structure (PS) rules as relations on objects of a local tree. The phrase structure rules function as backbone to which the grammatical constraints expressed in definite clauses are attached. Such a setup was used for the implementation of HN.

[36] ALE additionally offers the possibility to specify type definitions. Therefore ALE belongs to both group 1 and group 2 architectures, i.e. it can be used to formulate grammars in two different ways: by specifying constraints on the domain and by writing down relations on objects. However, currently the formal status of such hybrid setups is rather unclear. In the grammar implemented only the relational level is used to express the linguistic theory.

The method used to encode the HPSG theory in a phrase structure backbone setup while at the same time trying to preserve the character of the original theory is discussed in sections 4.2 and 4.3.

The difference between the two groups of architectures is reflected in the queries which can be formulated in those architectures. While in the first group one queries with a description to see whether that description describes any objects, in the second group one asks whether a relation holds between objects.

So far we have not yet explained, how an HPSG grammar can be expressed in a group 2 setup. The HPSGII theory is based on an architecture in which the domain can be directly constrained by the theory (group 1). Still, for computational reasons, almost all computational systems use a group 2 setup. It is therefore interesting to take a look at how a setup like the first can be modelled in a system of the second group.

Modelling group 1 behavior in a group 2 setup

What does it need to get the feel of a group 1 setup in a group 2 system? For the linguist wanting to encode a grammar it boils down to two things: How are the constraints making up the theory specified? and How is the system queried? Something we need to integrate into a group 2 setup to get a group-1-like behavior regarding the constraint specification is the hierarchical organization of constraints. An object of a certain type has to satisfy all the constraints on objects of that type *and* all the constraints on objects of any of its supertypes. Regarding the queries, we want to be able to ask if a description is satisfied by any objects which satisfy the theory.

If we restrict ourselves to implicative constraints with type antecedents[37], a general solution is to introduce three relations for every type. One relation is needed to encode the constraints defined for the type, i.e. to specify the type and to encode the consequent of the type definitions for the type which are part of the theory. A second relation collects all constraints on the type (by calling the first relation) and its subtypes. Finally a third relation is responsible for adding

[37]This opens up a new area of application for the technique for encoding complex antecedents of implicative constraints as type antecedents by modifying the signature discussed in section 3.2.1 on pp. 15.

the constraints on the supertypes of the type. Let us illustrate this method with an example.

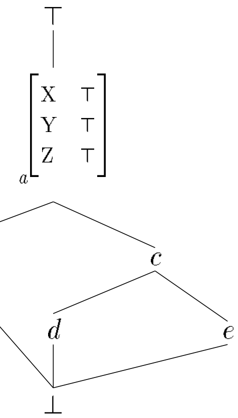

Figure 27: The example signature

As usual, the signature definition in figure 27 declares a domain of objects. The theory in figure 28 specifies which objects are admissible. It consists of two implicative constraints with types as antecedent.

$$c \quad \rightarrow \quad \begin{bmatrix} \mathrm{X} & \boxed{1}\begin{bmatrix}\mathrm{Z} & \mathrm{b}\end{bmatrix} \\ \mathrm{Y} & \boxed{1} \end{bmatrix}$$

$$d \quad \rightarrow \quad \begin{bmatrix} \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix}$$

Figure 28: The example theory in a group 1 setup

In figure 29 the group 2 theory with the three relations for each type is listed.[38] The relation name subscripts and the names of the lines in which the relations are defined have the following interpretation: "h" stands for the *hierarchy* of constraints below the type (including the type itself), "c" for the *constraint* on the type, and "t" for all constraints on the *type* including those inherited.

1. h) $\top_h(\boxed{1})$:- $\mathrm{a}_h(\boxed{1})$.

2. t) $\mathrm{a}_t(\boxed{1})$:- $\top_h(\ \boxed{1}a\ )$.
   h) $\mathrm{a}_h(\boxed{1})$:- $\mathrm{a}_c(\boxed{1})$, ($\mathrm{b}_h(\boxed{1})$; $\mathrm{c}_h(\boxed{1})$).
   c) $\mathrm{a}_c(a)$.

3. t) $\mathrm{b}_t(\boxed{1})$:- $\top_h(\ \boxed{1}b\ )$.

[38] A notation reminiscent of PROLOG is used for the relational level: disjunction is noted as ";" and conjunction by ",". The full set of three relations per type is only displayed to illustrate the general mechanism. In an implementation several of the relations defined could be eliminated, for example by partially executing the definite clauses at compile time.

h) $\mathrm{b}_h(\boxed{1})$:- $\mathrm{b}_c(\boxed{1})$.

c) $\mathrm{b}_c(b)$.

4. t) $\mathrm{c}_t(\boxed{1})$:- $\top_h(\ \boxed{1}c\ )$.

h) $\mathrm{c}_h(\boxed{1})$:- $\mathrm{c}_c(\boxed{1})$, ($\mathrm{d}_h(\boxed{1})$; $\mathrm{e}_h(\boxed{1})$).

c) $\mathrm{c}_c(\ c \wedge \begin{bmatrix} \text{X} & \boxed{1}\begin{bmatrix}\text{Z} & \boxed{2}\end{bmatrix} \\ \text{Y} & \boxed{1} \end{bmatrix}\ )$:- $\mathrm{b}_t(\boxed{2})$.

5. t) $\mathrm{d}_t(\boxed{1})$:- $\top_h(\ \boxed{1}d\ )$.

h) $\mathrm{d}_h(\boxed{1})$:- $\mathrm{d}_c(\boxed{1})$.

c) $\mathrm{d}_c(\ d \wedge \begin{bmatrix} \text{Y} & \boxed{1} \\ \text{Z} & \boxed{1} \end{bmatrix}\ )$.

6. t) $\mathrm{e}_t(\boxed{1})$:- $\top_h(\ \boxed{1}e\ )$.

h) $\mathrm{e}_h(\boxed{1})$:- $\mathrm{e}_c(\boxed{1})$.

c) $\mathrm{e}_c(e)$.

Figure 29: The relations encoding the example theory in a group 2 setup

Take for example the encoding for the type c. The relation c_c imposes the description language constraints for type c on the argument of the relation. Here the link between relational level and the description language level is made. The argument of the relation is assigned the correct type, and the consequent of the type definition for c (cf. group 1 theory in figure 28) is conjoined to that. No inherited constraints are specified here, only the constraints directly imposed on the type. Whenever in a group 1 theory a type restriction is specified, in the corresponding group 2 theory, in the type's c-relation – where all description language restrictions are encoded – a call to the type's t-relation is used. For example, the specification $\left[\text{X}|\text{Z}\ \ b\right]$ in the type definition for type c is encoded in the relation c_c as a call to the relation b_t. This ensures that all constraints on the type, including the ones inherited, are applied whenever a type restriction is specified. Note that the appropriateness conditions for the argument of a t-relation are preserved under this approach, since every c-relation ensures that the argument is assigned the correct type.

The relation c_h references the constraints on type c which were collected in c_c and adds the constraints defined for the subtypes of c all the way down to the most specific subtypes. In the example there is one more level to the most specific types d and e. This step ensures that every object in the denotation of the type c also is in the denotation of one of the most specific subtypes of c, basically enforcing a limited version of the closed world interpretation introduced in section 3.1.

Finally, the relation c_t imposes on its argument all constraints on type c, those directly specified for the type, and those inherited. This is done by collecting the constraints for type c from the constraint hierarchy all the way from the most general type *top* down to a most specific sybtype of c.

Summing up, this encoding technique provides us with a general mechanism for transferring theories expressed using class 2 implicative statements into an architecture in which a theory has to be formulated on the relational level. The "look and feel" of the original formulation is preserved. It is clear that one cannot expect the linguist to re-encode a theory in such an awkward way, but such a re-encoding could be done in a automated compilation step transferring the grammar written by the linguist into machine friendly code.

In HPSGII only very few types are actually constrained, i.e. appear as the type of the antecedent of an implicative constraint. Also, most currently implemented grammars are only used to query for (headed) phrases. The result is that only few relations need to be defined to work with the system. Therefore most current implementations of HPSG in a group 2 system do not follow anything like the general encoding scheme introduced above. On the other hand, an implementation close to the original HPSG theory is only possible if constraints on any kind of object can be expressed and queried separately from the top-level signs. Only such an implementation can be produced and used by the linguist to provide valuable feedback for a rigid and complete formalization of the linguistic theory. Therefore a general automatic translation procedure between the linguistic theory and the relational encoding along the lines described above seems desirable. It allows working close to the original linguistic theory, enables a modular development and testing of linguistic theories, while at the same time supplying runnable code.[39]

4 Modules of an HPSG theory: Special linguistic types of constraints and choices for encoding them

4.1 Specifying the lexicon

Like most current HPSG-style analyses, the theory proposed in HN is strongly based on lexical specification. In the light of the discussion of the formal background of HPSG in the previous sections, the set of constraints which linguists refer to as 'the lexicon' is a collection of implicative constraints no different from any other constraint in the grammar (e.g. the principles). Still there is a reason why lexical information deserves a second look. In any normal grammar the constraints making up the lexicon vastly outnumber the rest of the constraints of the grammar. From a linguistic as well as a computational point of view it is therefore interesting to analyze how generalizations over this big pool of constraints can be encoded.

The necessity to express lexical generalizations has long been recognized. Shieber (1986) comments the situation as follows: "First, we can come up with general techniques for expressing lexical generalizations so as to allow lexical entries to be written in a compact notation. ... devices as templates and lexical rules in PATR-II ... serve just this purpose: to express lexical generalizations. As grammars rely more and more on complex lexical encodings (as is the current trend in unification-based formalisms) these techniques become increasingly important. Second, the formalism can be extended in various ways to be more expressive"(p. 36). We will see in this section that Shieber's first point directly carries over to the current HPSG setup. The lexical rules are part of the HPSG architecture, and the templates (now often called *macros*) are used in implementations of HPSG grammars. Regarding Shieber's second point, we will show that the HPSG formalism is powerful enough to allow for lexical generalizations to be expressed within the general constraint formalism.

Basically, two kinds of generalizations over lexical information are used in HPSG. Instead of

[39] The TFS encoding Meurers (1993) (group 1) was transformed by hand (and sed) into a CUF grammar (group 2) along the lines described above. The performance results in CUF were comparable to those of the original grammar running under TFS.

completely specifying each lexical entry separately, properties of whole classes of lexical entries are specified. On the other hand, the existence of certain lexical entries is related to the existence of other entries. After an introduction to the basic setup of the lexicon, both kinds of generalizations and the formal mechanisms for expressing them are discussed.

4.1.1 Basics

In the simplest case, a lexicon is a constraint on objects of type *word*.[40] Since the set of possible lexical objects which are admitted by the linguistic theory need to be constrained, in an HPSG grammar there is no way around a statement defining the basic lexicon like that given in figure 30.

$$\textit{word} \quad \rightarrow \quad (L_1 \vee \ldots \vee L_i \vee \ldots \vee L_n)$$

Figure 30: The basic lexicon: a type definition for the lexical type *word*

In the terminology introduced in section 3.2, the figure 30 shows a type definition for type *word* with a disjunctive description as consequent. Each L_i is a lexical entry. The effect of this implicative constraint is that every lexical element has to be licensed by one of the disjuncts. There is no way for a lexical object in a syntactic tree to "sneak through" unconstrained, which could be the case if other specifications were included in the antecedent.

4.1.2 Generalizing over a set of entries

The question we want to answer in this section is: How can the same information be specified on several lexical entries? Referring to figure 30 this comes down to asking: How can information be specified on several lexical entries L_i without repeating it in the description of each disjunct. There are two possible answers to this question and one theoretically less interesting solution which at least avoids syntactic repetition. In the following discussion of the three possibilities, we speak of a description C which we want to specify on a set of lexical entries $\mathcal{G}$.

Expressing lexical generalizations outside of the lexical type definition

For the first possibility, one starts with leaving the description C out of the specification of each of the L_i disjuncts in $\mathcal{G}$. The provisional result is a lexicon in which the lexical entries in $\mathcal{G}$ are too unrestricted. Next, the constraint C is introduced in the theory as an additional object definition. The antecedent is specified to describe exactly the objects which are described by the lexical entries in $\mathcal{G}$, i.e. it specifies a property unique to the elements of $\mathcal{G}$.[41] The consequent

[40] In common HPSG theories *word* is the only "lexical" type. In case one wants to postulate phrases in the lexicon (i.e. for proper names or bare plural noun phrases) a separate lexical type *lex-phrase* could be introduced. The type *lex-phrase* would be defined as subtype of *phrase*, with sister-type *struc-phrase*. The attribute DTRS should then be defined as appropriate for *struc-phrase* instead of for *phrase*. In the following, this issue is ignored and *word* is assumed to be the only lexical type. Other kinds of lexical types will play a role though in the discussion of the so called lexical type hierarchies in section 4.1.2.

[41] If the elements of the set $\mathcal{G}$ do not have a unique property to distinguish its elements from all other linguistic objects, such a unique property needs to be artificially introduced in each of the concerned L_i specifications in the *word* type definition. However, such a specification naturally contradicts the original intention to specify

of the object definition is the constraint C.

We have already seen an example for this encoding technique in section 3.2.1. The object definition of figure 18 (p. 16) was defined to attach the argument raising specifications to auxiliaries.

Lexical type hierarchies

The second possibility is to introduce subtypes of the lexical type *word* and attach the description C to the consequent of the type definition of a subtype. Since under a closed world interpretation of a type hierarchy every object that is of a type t also has to be described by one its subtypes, the real work under this approach is to make sure that via manipulation of the signature we manage to have one of the subtypes of *word* describe exactly those *word* objects which belong to the set $\mathcal{G}$ (every element of which we want to constrain with C). Like in the first encoding proposed above, this requires us to have a group defining property for the elements of $\mathcal{G}$. Following the method described on pp. 15 of section 3.2.1, the description of the group defining property then has to be encoded in a type, i.e. a subtype of *word*. An example for such an encoding of a complex description in the signature definition of a type is given in figure 19 (pp. 16) where the type *aux-word* is introduced to reference all auxiliary-verb words. Once the subtype of *word* is introduced in the signature, it can function as antecedent of an implicative constraint and can specify the lexical entries of $\mathcal{G}$ as displayed in figure 31. Note the conjunct C which is the description valid for all disjuncts, i.e. for all lexical entries which are of type *word-subtype*.

$$\textit{word-subtype} \ \rightarrow \ (L_j \vee \ \ldots \ \vee L_k) \wedge C$$

Figure 31: Type definition for a lexical subtype

When this approach is used to express several different generalizations over different sets of lexical entries, the result is a complex subtype hierarchy below *word*, which has sometimes been called a *lexical type hierarchy*.

Comparing this lexical type hierarchy approach to the first proposal, one notes that the encoding of a complex description in a type (which complicates the signature and restricts the antecedents to class 2 descriptions) is necessary because of theoretical considerations in the lexical type hierarchy approach only. This is caused by the fact that in the lexical type hierarchy approach a subtype of the lexical type is introduced to get a hold on the lexical entries of the set $\mathcal{G}$. The correct denotation of this subtype has to be ensured by manipulating the signature. In the first approach there is no theoretic reason for restricting the implication antecedent to types. However, in practice, the restriction of current computational systems to type definitions forces the user in both approaches to encode the class 2 antecedents describing the group defining property as types.

Macros as syntactic placebo

By now, the person interested in implementing a grammar is probably rather desperate. Since, as already mentioned several times, current computational systems do not allow the grammar writer to express implications with complex antecedents, both proposals made so far for expressing

information without repeating it in each disjunct. It is questionable whether the set of entries does form a natural set over which one should express generalizations. The approach in such a case becomes equivalent to (and as theoretically uninteresting as) the macro approach described below.

lexical generalizations have a fundamental flaw. They force the implementer to encode additional information in the signature, which severely complicates the hierarchy.[42] In order to at least avoid writing the same description over and over in the specification of lexical entries, a syntactic grouping of information under one name is an emergency solution that is better than nothing. The idea borrowed from the template mechanism of PATR-II is to give a name to a set of descriptions and to use that name in place of the descriptions. These syntactic abbreviations nowadays are usually called *macros*. The use of the macro mechanism for lexical specification is displayed in figure 32.

$$\textit{word} \;\rightarrow\; ((L_1 \wedge M) \vee L_2 \vee (L_3 \wedge M))$$

Figure 32: An example type definition for the lexical type *word* using a macro M

The macro name M abbreviates the description C. It is conjoined to each of the lexical entries belonging to the set $\mathcal{G}$. In the example of figure 32, L_1 and L_3 belong to the set while L_2 does not.

Before we turn to illustrating the macro mechanism, let us make a remark on lexical specification in those computational architectures which express grammatical constraints on the level of a relational extension of the description language (group 2 architectures). So far, the discussion of the three possibilities for expressing generalizations over lexical entries were all based on an architecture like that of HPSGII, in which the domain of objects can be directly constrained (group 1 architectures). Without using the technique for expressing group 1 theories in group 2 architectures described in section 3.2.2, the first two possibilities for expressing generalizations discussed above are not available in a group 2 setup since the domain cannot be directly constrained. One therefore only has the possibility to use the macro method, the syntactic grouping of descriptions under a name. This also is the case for the specification of lexical entries in those group 2 systems which use a phrase structure backbone. As a result, the phrase structure backbone based ALE and Troll implementation documented here makes heavy use of macros to specify the lexicon.

To facilitate the presentation of examples from the implemented grammar, we now briefly introduce the macro mechanism of ALE.[43] Figure 33 shows a simple definition of a macro in the ALE implementation.

```
unmarked macro (synsem:loc:cat:marking:unmarked).
```

Figure 33: Definition of macro "unmarked"

A call to the macro, i.e. `@unmarked`, is equivalent to writing $\left[\text{SYNSEM|LOC|CAT|MARKING} \;\; \textit{unmarked}\right]$, i.e. the macro "unmarked" denotes *sign* objects whose marking values are of type *unmarked*.

[42] The reader thinking that "severely complicates" is an overstatement here, is reminded that encoding the relatively simple description of a word with a verbal head and $\left[\text{AUX} \;\; \text{plus}\right]$ specification already demanded 19 signature definitions (cf. figure 19 on p. 16).

[43] Note that, since macros are abbreviations for (ALE) descriptions, they cannot contain calls to definite clauses.

A macro can have parameters passed to it, allowing more versatile definitions.

```
head(X) macro (synsem:loc:cat:head:X).
```

Figure 34: Definition of macro "head"

Here a call, e.g. `@head(verb)`, abbreviates the description of a sign, whose head value has the value passed as the argument – the type *verb*.

Other macros can be called in the definition of a macro. A restriction is that macro definitions are not allowed to be recursive. By using the call to a macro as a specification in the definition of another macro, hierarchies of definitions are formed. These hierarchies are subsumption hierarchies of descriptions. They have no theoretic status in the architecture of HPSG and should not be confused with type hierarchies as part of the signature. In the absence of other theoretically more satisfying possibilities (i.e. in the current implementation environments), such macro hierarchies are very useful for hierarchically grouping information. In the implemented grammar, the lexical entries were provided with the same information through such a hierarchy of lexical macros. It is displayed in figure 35.

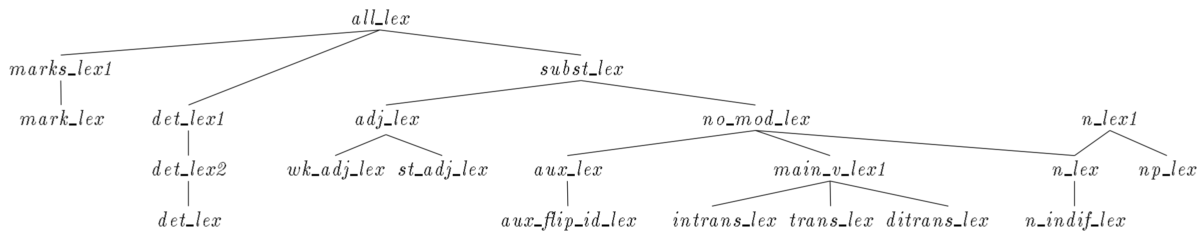

Figure 35: The lexical macro hierarchy

The macros become more specific the further down in the hierarchy one goes. Note that contrary to type hierarchies, where we know that every linguistic object has to be of a most specific type, any of the macros in the hierarchy can be used to specify a lexical entry, not just the most specific ones.

Part of the information grouped in the lexical macro hierarchy above serves to specify theory independent linguistic information (e.g. HEAD or CASE specification) and linguistic information driving a specific analysis (e.g. argument raising, NPCOMP value), other specifications seem to have a different status in supplying more general HPSG architecture mechanisms with information (e.g. NONLOCAL, QSTORE values). It seems obvious that this wealth of information and its dependence structure should find its way into a proper HPSG device – even though this section has illustrated that in most architectures (group 2 and those of group 1 which do not allow complex antecedents) it is not clear what such a device should look like.

The macro hierarchies have often been confused with the lexical type hierarchies discussed on p. 26. Macro hierarchies define abbreviations for complex descriptions by hierarchically organized definitions of macros. The macros can then be used in the specification of constraints. The confusion arises from the fact that the lexical types are introduced as encodings of complex descriptions (cf. p. 26). Nonetheless, lexical *macro* hierarchies and lexical *type* hierarchies are two completely different formal entities. The concepts and terminology from one can therefore not be carried over to the other without a redefinition.

Summing up, there are three distinct approaches for expressing generalizations over lexical entries. It is possible to use several of these approaches in one grammar, but one should be aware that they employ different formal mechanisms to achieve their goal.

4.1.3 Relating lexical entries

Now that we have discussed basic lexical entries and the methods for expressing generalizations over them, we can turn to the second type of generalization used in HPSG-style theories: relations specifying a lexical entry on the basis of another entry. Traditionally these relations are called *lexical rules*. Formally they are thought of as "meta-relations", since they relate lexical entries, i.e. descriptions and not the objects which are related by ordinary relations. Since a clear formalization of lexical rules as meta-relations has not yet been given,[44] we will only sketch under what idea of functionality linguists have used the lexical rule formalism in the formulation of linguistic theories. Following this introduction of the lexical rule mechanism, we will investigate how far we can get by expressing the generalizations captured in lexical rules as meta-relations strictly on the level of the description language. We then take a look at a computational mechanism which in the ALE system provides some of the functionality of lexical rules. As illustration of that computational mechanism, we show how the PVP extraction lexical rule of HN can be encoded in ALE. The discussion of lexical rules ends with some remarks on the (non-)correspondence between linguistic and computational generalizations. Finally, as a possible theoretical and computational alternative to the lexical rules, an extension of the description language, the *named disjunction* mechanism, is introduced.

Lexical rules

In traditional terms, a lexical rule is a pair of two meta-descriptions, one for each class of descriptions related. Views vary significantly regarding the interpretation of the information on the two meta-descriptions and that of the lexical rule as such.

Application criterion:
Two different assumptions are made regarding the question of when a lexical rule applies to a lexical entry (a description): Either the lexical entry has to be *more specific than* the left-hand side of the lexical rule, or the lexical entry has to be *consistent with* the left-hand side of the lexical rule. Using the "more specific than" option is the more restrictive approach, which corresponds to the view of information in the lexical entry triggering the lexical rule. The "consistent with" option on the other hand additionally allows descriptions which are part of the left-hand side of a lexical rule to further narrow down the denotation of the lexical entry, which makes it impossible to distinguish between the specifications of the entry and that of the rule.

Often the term *subsumption* is used for the "more specific than" relation and *unifies with* instead of "is consistent with". One should be aware that using these terms when talking about the lexical rule mechanism is very informal. Unification and subsumption are not even part of the

[44] Pollard (1993) is a sketch of such a formalization.

description language.[45] It is therefore not possible to "lift" the two relations from the description language level to the meta description level. In fact, the issue of lexical rules as meta-descriptions is even more complicated, since even a formally defined "meta-subsumption" or "meta-unifies-with" would be inappropriate because entities on two different language levels need to be related. A lexical entry is a *description* whereas a lexical rule is a pair of *meta-descriptions*. The easiest solution would be to drop the idea of lexical rules as meta-relations and use ordinary relations on objects to capture the desired functionality. However, the following section will show that such an encoding misses some of the functionality associated with lexical rules. Before we analyze this matter further, let us finish the description of the functionality associated with lexical rules in intuitive terms.

The nature of the transfer:
Once the rule applies, the tags specified in the left-hand side "pick out" information from the lexical entry and "supply" it to the occurrences of the tags in the right-hand side. This information transfer can be done via "*unification*" or *copying*.[46] The specifications on the left-hand side of a lexical rule therefore has two functions: it triggers rule application and provides information handles.

Unchanged specification carryover:
Information "not changed" by the lexical rule is assumed to be transferred to the output. Several possibilities arise here concerning the interpretation of "not changed" and again the nature of the transfer.

A formalization of lexical rules as meta-descriptions will have to refer to these choices. One should bear in mind that the exact formulation chosen has immediate consequences on the linguistic theories which can be formulated using the lexical rule mechanism. Höhle (1994) shows that a lexical rule mechanism with unification as the application criterion does not work properly in a grammar where complement extraction is done via lexical rules and argument raising à la Hinrichs and Nakazawa is used to construct the verbal complex.

Looking at the theory proposed in HN a further desideratum of the lexical rule mechanism presents itself. The PVP extraction lexical rule and the split-NP lexical rule share a common mechanism. Roughly speaking, partially extracted constituents are related to remnants in base position. This is implicit in the paper, but cannot be expressed in the formulation of the lexical rules. Using a hierarchically structured tool to relate lexical entries, it should be possible to express different levels of generalizations and capture the resemblance. To our knowledge, no formulation of such hierarchical lexical rules is proposed in the literature.

Lexical rules as part of the description language

In this section we investigate how far we can get by using only the description language to capture the functionality of lexical rules. Figure 36 shows a type definition for *word*, which

[45] In the Carpenter setup unification and subsumption play a role on the semantic level, not on the syntactic level of the description language. In the King setup, on which HPSGII can be based, unification and subsumption play no role at all. Cf. 2

[46] The tags which are often used to notate the information transfer in lexical rules as meta-descriptions should not be confused with the tags used in the description language. A description language tag is interpreted as identity of the set of objects denoted, whereas the lexical rule "meta-tags" specify identity of descriptions.

defines an extended lexicon including lexical rules.[47]

$$\mathit{word} \quad \rightarrow \quad (L_1 \vee \ldots \vee L_i \vee \ldots \vee L_n \vee \boxed{1}\begin{bmatrix}\text{STORE} & < {}_{\mathit{lex\_rule}}\begin{bmatrix}\text{OUT} & \boxed{1}\end{bmatrix} >\end{bmatrix})$$

Figure 36: A lexicon with added lexical rule relations

The formulation makes use of the junk slot technique discussed on p. 18 of section 3.2.1 for encoding relations on the level of the description language. The type *word* is assumed to have an additional appropriate attribute STORE which is *list*-valued. Furthermore a new type *lex_rule* is introduced below $\top$, having IN and OUT as appropriate attributes with *word* values. The type definition for *lex_rule*, in which the different lexical rules are specified, is given in figure 37.

$$\mathit{lex\_rule} \quad \rightarrow \quad ({}_{\mathit{lex\_rule}}\begin{bmatrix}\text{IN} & D_1 \\ \text{OUT} & E_1\end{bmatrix} \vee \ldots \vee {}_{\mathit{lex\_rule}}\begin{bmatrix}\text{IN} & D_m \\ \text{OUT} & E_m\end{bmatrix})$$

Figure 37: Specifying the lexical rules

How does this description language encoding work? The definition of the lexicon in figure 36 is a type definition on the type *word* just like the basic lexicon of figure 30. Every object of type *word* has to satisfy one of the standard lexical entries L_i or it satisfies a special lexical entry $\boxed{1}$. The special entry $\boxed{1}$ is the description which is the OUT value of the *lexical_rule* object residing in the special entry's junk slot attribute STORE.[48]

The admissible *lex_rule* objects are defined in figure 37. Because of the special lexical entry $\boxed{1}$ in the type definition of word, every OUT description of a lexical rule object is a possible word. To relate the OUT value to another lexical entry, structure sharing between the descriptions D_i and E_i of each disjunct in figure 37 needs to be specified. Note that the appropriateness conditions for *lex_rule* ensure that the value of the IN attribute is of type *word*. It therefore has to satisfy the type definition for word, i.e. one of the lexical entries of figure 36. Naturally that lexical entry can again be the output of a lexical rule.

Let us illustrate the definition of a lexical rule, i.e. one of the disjuncts of figure 37, with an example based on the HPSGII appendix signature. For expository purpose only, assume we want to write a lexical rule, which extracts the subject of intransitive verbs with base verb-form

[47] Several equivalent formulations are possible. For example, two subtypes of *word* can be introduced, one for the standard lexicon (*normal_word*) and one for the results of lexical rules (*lex_rule_word*). The type *lex_rule_word* has an appropriate, word typed IN attribute. The "out" specifications are specified directly on the *lex_rule_word* object. This encoding is basically the one used in the TFS grammars documented in Kuhn (1993) and Keller (1993).

[48] Note that the description of the special entry is a cyclic structure. This can be avoided if one instead introduces a *lex_rule* subtype of word as described in the previous footnote. However, the encoding with the cyclic structure has the advantage of clearly separating lexical rule objects from words. This makes it easier to compare the lexical rules in the description language setup with the other formalizations of lexical rule relations.

and inserts it on the inherited slash. The resulting encoding is shown in figure 38.

$$
\left[\begin{array}{ll}
\text{IN} & \left[\begin{array}{ll}
\text{PHON} & \boxed{1} \\
\text{SYNSEM} & \left[\begin{array}{ll}
\text{LOCAL} & \left[\begin{array}{ll}
\text{CAT} & \left[\begin{array}{ll}
\text{HEAD} & \boxed{2}\left[\text{VFORM}\ \ \text{bse}\right] \\
\text{SUBCAT} & < \boxed{\text{x}} > \\
\text{MARKING} & \boxed{3}
\end{array}\right] \\
\text{CONTENT} & \boxed{4} \\
\text{CONTEXT} & \boxed{5}
\end{array}\right] \\
\text{NONLOCAL} & \left[\begin{array}{ll}
\text{TO-BIND} & \boxed{6} \\
\text{INHERITED} & \left[\begin{array}{ll}
\text{SLASH} & \boxed{7} \\
\text{REL} & \boxed{8} \\
\text{QUE} & \boxed{9}
\end{array}\right]
\end{array}\right]
\end{array}\right] \\
\text{QSTORE} & \boxed{10} \\
\text{RETRIEVED} & \boxed{11}
\end{array}\right]_{word} \\
\text{OUT} & \left[\begin{array}{ll}
\text{PHON} & \boxed{1} \\
\text{SYNSEM} & \left[\begin{array}{ll}
\text{LOCAL} & \left[\begin{array}{ll}
\text{CAT} & \left[\begin{array}{ll}
\text{HEAD} & \boxed{2} \\
\text{SUBCAT} & <> \\
\text{MARKING} & \boxed{3}
\end{array}\right] \\
\text{CONTENT} & \boxed{4} \\
\text{CONTEXT} & \boxed{5}
\end{array}\right] \\
\text{NONLOCAL} & \left[\begin{array}{ll}
\text{TO-BIND} & \boxed{6} \\
\text{INHERITED} & \left[\begin{array}{ll}
\text{SLASH} & \left[\begin{array}{ll}
\text{ELT} & \boxed{\text{x}} \\
\text{ELTS} & \boxed{7}
\end{array}\right] \\
\text{REL} & \boxed{8} \\
\text{QUE} & \boxed{9}
\end{array}\right]
\end{array}\right]
\end{array}\right] \\
\text{QSTORE} & \boxed{10} \\
\text{RETRIEVED} & \boxed{11}
\end{array}\right]_{word}
\end{array}\right]_{lex\_rule}
$$

Figure 38: An example for a lexical rule description

The description of the verb-form (tagged $\boxed{2}$) on the input ensures that only *bse* verbs can undergo this lexical rule. The tag $\boxed{\text{x}}$ references the (only) subcat list element of the input and is identified with an additional element of the slash set of the output description. The tags $\boxed{1}$ through $\boxed{11}$ ensure that all other specifications are structure shared between the input and the output.

What effect does this encoding have on the choices mentioned above for lexical rules as meta-relations: the application criterion, the nature of the transfer, and the carrying over of unchanged information?

To make the following discussion easier to read, we call one of the disjuncts of figure 37, a description of admissible lex-rule objects, R. It's IN attribute bears the specification D and its OUT attribute the description E. Furthermore L is a lexical entry L_i of figure 36.

When does the lexical rule R apply to a lexical entry L? The input specification D of the lexical rule R describes a set of word objects. Only those word objects which also satisfy the type definition for *word* satisfy our theory. Looking at that type definition in figure 36, this means that the word objects which are denoted by D also have to be in the denotation of a lexical entry L to satisfy the theory. The result is that a lexical rule applies to those objects which are in the intersection of the denotation of the lexical entry L and the input description D. This semantic characterization on the syntactic side corresponds to the conjunction of the descriptions of D and L, which in the Troll setup is computed as the unification of the feature structure normal form representation of both descriptions. In the informal terminology of the lexical rule as meta-relation sketch above, the setup of lexical rules in figure 36 therefore entails that a lexical rule applies if its input description "unifies with" the description of a lexical entry.

Now to the nature of the transfer. To state that two descriptions are partly identical, structure sharing (token identity) between parts of the descriptions of the IN attribute's value and that of the OUT attribute needs to be specified. Using the expressive power of the logic only, it is not possible to demand type identity without explicitly using the same description twice. The logic does not contain anything corresponding to the intuitive notion of copying. The result for our lexical-rule-like mechanism formulated within the description language is that the only way to relate the input and the output descriptions D and E is by specifying structure sharing between their sub-descriptions.

Similarly, it is not possible to specify any "carrying over" of "unchanged" specifications. All "carrying over" is done by explicitly specifying structure sharing, i.e. all descriptions have to be explicitly specified on the OUT attribute. The grammar writer therefore has to decide for each lexical rule which information is to be preserved and ensure that it gets transferred from the input to the output of the rule. For two reasons it is sometimes very difficult to fulfill this task. Assume the following signature:

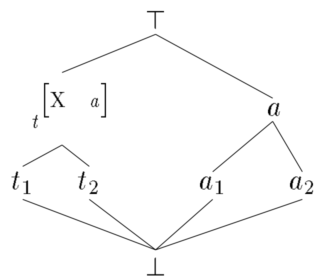

Figure 39: An example signature

Now take the description $_{t_1}[\text{X} \quad a_1]$ and suppose we want to specify a second description so that it specifies all the information of the first, except that the value of its X attribute is specified as a_2. This is only possible if we can access the type of an object independent of its attribute specifications. No computational system currently provides this possibility. The type information t_1 of the first description therefore can not be specified on the second feature structure without copying the whole t object including the $[\text{X} \quad a_1]$ specification. The only solution is to split the lexical rule into many different lexical rules, one for each of the possible types of the objects which cannot be carried over completely. In the example above this results in two lexical rules: one for t_1 objects and one for t_2 objects. In every more complex case this leads to a serious multiplication of lexical rules which loses some, possibly all originally intended generality.

As a slightly more complex example based on the signature definitions given in the appendix of

HPSGII suppose (for expository purpose only) we want to specify a lexical rule, which relates all non-predicative entries to predicative ones. The only thing the lexical rule does is change the PRD attribute's value from *minus* to *plus*. All signs having a PRD attribute (i.e. the signs with substantive heads) can undergo this lexical rule. Regarding the question of which specifications of the input are supposed to be carried over to the output, the answer has to be that everything except for the HEAD value should be passed, since the HEAD value bears the PRD specification. The unintended but unavoidable result is that the type of the HEAD value object of the lexical entry (i.e. *prep*, *noun*, *verb*, or *adj*) is lost after the application of the lexical rule. The only solution is to split the original lexical rule into one for each HEAD subtype. These lexical rules then carry the specification of the HEAD subtype on the input and the output. Note that this completely loses the originally intended generalization over all non-predicative signs.

The second problem regarding the specification transfer from input to output is caused by the fact that the feature geometry of lexical entries which can undergo a single lexical rule can vary significantly. In some cases the grammar writer therefore has to split up a lexical rule and specify which information gets carried over depending on the shape of the input. For example for the prd-lexical-rule of the last paragraph, a separate lexical rule would have to be made for verbs (to ensure the VFORM specification is carried over), nouns (to ensure the same for CASE), prepositions (PFORM), and for the only remaining substantial signs without specific head attributes, the adjectives. Again, the generalization over all substantial signs is lost.

Summing up, it is possible to encode some of the functionality commonly associated with lexical rules on the level of the description language. However, the resulting description language mechanism commits the user to explicitly specify which information of the input is supposed to be present on the output. We have seen that this can cause problems. Nonetheless, the kind of description language mechanism defined currently is the only way to include at least the core of the functionality of lexical rules in a well-defined and formal way inside of the feature logic on which HPSGII is based. The restriction of the description language mechanism to an explicit transfer of specification is mainly caused by the problem to formalize the notion of a transfer of "unchanged specifications". We will see in the next section that the computational mechanism to encode lexical rules used in the ALE system has exactly the same deficits as our description language encoding.

The PVP extraction lexical rule and its implementation in ALE

Before turning to the specific ALE mechanism for lexical rules let us comment on the procedural metaphors commonly used with lexical rules. Even though lexical rules formally are binary relations without explicit order, one usually speaks of an "input" description and an "output" description of a lexical rule. The meaning of this is that the lexicon without the lexical rules is usually set up so that it contains certain lexical entries matching the input descriptions of the lexical rules, but none which match the output descriptions. When the procedural metaphor behind this terminology is transferred into a procedural interpretation in the sense that lexical rules produce new lexical entries (like in the ALE system), this can lead to unintended results. The problem is that different application orders can produce several instances of the identical lexical entry.

Take for example two lexical rules R_1 and R_2 which both apply to a lexical entry A. Additionally assume R_1 and R_2 modify different parts of A and that the intersection of the denotation of the output description of either of the rules and that of the input description of the other rule is not empty. In this situation, a procedural interpretation of lexical rule application would produce four new entries, one resulting from the application of R_1, one from the application of R_2, one from applying R_1 and then R_2, and finally one where first R_2 and then R_1 was applied. The last two possibilities cause two identical lexical entries to be added to the lexicon. Clearly such a production of identical entries is an unwanted effect of the procedural interpretation of lexical rules.

In the ALE system, lexical rules are basically a special notation for unary phrase structure rules with an additional mechanism to alter the morphology. When the grammar is compiled, the closure under lexical rule application is computed. New lexical entries are produced and added to the lexicon. As mentioned above this can lead to identical entries as the result of several possible orders of application of one lexical rule on the output of another. Application is done with unification as test, and the new entry is created with copies of the information specified in the tags. Just as with the description language formalization just discussed, only the information explicitly specified on the right-hand side of the lexical rule finds its way into the new entry. The resulting problems noted above therefore apply to the ALE formalism as well.[49] An additional complication of the specification transfer issue can arise from the procedural interpretation of the lexical rules. Because of different possible application orders, an insufficient specification of propagation of information from input to output can lead to unwanted unilateral subsumption between entries (in addition to identical entries resulting from applying the same lexical rules in different order). Whether all unilateral subsumption can be eliminated by changes to the lexical rules depends on the possibility to completely pass unchanged information. As discussed above this can sometimes only be achieved at the loss of generality by splitting a lexical rule into many different lexical rules.[50]

To illustrate a lexical rule encoding in the ALE system, let us take a closer look at the PVP extraction lexical rule which in HN drives the analysis of possibly partial verb phrase extraction.

[49] In the implemented grammar it was possible to avoid the problem regarding the different feature geometries of input and output lexical entries because of the homogeneous nature of the application domain of the lexical rules in the fragment (mostly verbal entries).

[50] In the implementation, extra PROLOG routines were used to detect and eliminate mutually and unilaterally subsuming lexical entries. With additional specifications on the lexical rules and entries it was possible to eliminate the unwanted lexical entries in the special case of the implemented grammar.

Figure 40 shows the lexical rule as given in HN[51].

$$\left[\text{SYNSEM}\ \left[\begin{array}{ll}\text{LOC|CAT} & \left[\begin{array}{ll}\text{HEAD} & {}_{verb}\!\left[\begin{array}{ll}\text{VFORM} & bse\\ \text{AUX} & plus\end{array}\right]\\ \text{VAL|COMPS} & \left\langle \left[\text{SYNSEM|LOC}\ \left[\begin{array}{ll}\text{CAT} & \left[\begin{array}{ll}\text{HEAD} & {}_{verb}\!\left[\text{VFORM}\ \boxed{1}\right]\\ \text{VAL} & \left[\begin{array}{ll}\text{COMPS} & \boxed{2}\\ \text{SUBJ} & \boxed{3}\end{array}\right]\end{array}\right]\\ \text{CONT} & \boxed{4}\end{array}\right]\right] \;\middle|\; \boxed{2} \right\rangle\end{array}\right]\\ \text{NONLOC|INHER|SLASH} & \{\}\end{array}\right]\right]$$

$$\longrightarrow$$

$$\left[\text{SYNSEM}\ \left[\begin{array}{ll}\text{LOC|CAT|VAL|COMPS} & list(\neg\left[\text{SYNSEM|LOC|CAT|HEAD}\ \ verb\right])\\ \text{NONLOC|INHER|SLASH} & \left\{\left[\text{SYNSEM}\ \left[\begin{array}{ll}\text{LOC} & \left[\begin{array}{ll}\text{CAT} & \left[\begin{array}{ll}\text{HEAD} & {}_{verb}\!\left[\text{VFORM}\ \boxed{1}\right]\\ \text{VAL} & \left[\begin{array}{ll}\text{COMPS} & \langle\rangle\\ \text{SUBJ} & \boxed{3}\end{array}\right]\end{array}\right]\\ \text{CONT} & \boxed{4}\end{array}\right]\\ \text{NONLOC|INHER|SLASH} & \boxed{5}\end{array}\right]\right]\right\}\end{array}\right]\right]$$

Figure 40: The PVP extraction lexical rule

Figure 41 shows the corresponding ALE encoding:

```
pvpelr lex_rule
   (In, @verb, @bse,
            @plus_aux,
       @val_comps([ (@verb, @vform(Vform),
                     @val_comps(Comps),
                     @val_subj(Subj),
                     @cont(Cont))
                   | Comps ]),
       @no_in_slash)

**>
   (Out, @val_comps( (L, list_n_sign) ),
        @in_slash([ (@verb, @vform(Vform),
                     @comps_sat,
                     @val_subj(Subj),
                     @cont(Cont),
                     @in_slash(L)) ]))
if
   pass2(In,Out)
morphs
   X becomes X.
```

Figure 41: The encoding of the PVP extraction lexical rule

[51]The ordering on the comps list is inverted compared to the figure given in HN. Cf. section 5.1 for a discussion of this issue. Also, the $\left[\text{VFORM}\ \ bse\right]$ specification on the input is only implicit in the formulation of the PVP extraction lexical rule given in figure 19 of HN.

Due to the systematic use of macros, the implementation encoding closely resembles the lexical rule given in HN. The only difference concerns the value of the COMPS attribute in the output description. Since ALE does not support polymorphic types or negation of descriptions, the "list of non-verbal signs"-description on the COMPS attribute cannot be expressed directly. In the current theory, restricting the COMPS value of the output of the PVP extraction lexical rule to be a list of nominal signs, will also makes the intended correct predictions.[52] The negated descriptions therefore can be replaced with the description of a list of nominal signs. Due to the absence of polymorphic types, the method for encoding descriptions in the signature definition of a type (cf. section 3.2.1, pp. 15) was used to introduce a new type *list_n_sign* for "list of signs with a nominal head". This type constraint was then used to restrict the value of the COMPS attribute in the output of the lexical rule in figure 41.

The call to the definite clause *pass2* at the end of the lexical rule encoding of figure 41 takes care of the carrying over of information not changed by the lexical rule. The definition of the definite clause and the macro involved in passing all but the changed COMPS and INHERITED information is given in figure 42.

```
pass2(@core2(Qstore,Retr,Head,Marking,Npcomp,Val_subj,Val_spr,Cont,Conx,To_bind),
      @core2(Qstore,Retr,Head,Marking,Npcomp,Val_subj,Val_spr,Cont,Conx,To_bind))
   if true.

core2(Qstore,Retr,Head,Marking,Npcomp,Val_subj,Val_spr,Cont,Conx,To_bind) macro
   (word, (qstore: Qstore),
          (retr: Retr),
          @head(Head),
          @marking(Marking),
          @npcomp(Npcomp),
          @val_subj(Val_subj),
          @val_spr(Val_spr),
          @cont(Cont),
          @conx(Conx),
          @to_bind(To_bind)).
```

Figure 42: The mechanism for passing the unchanged specifications (everything but comps and inherited slash)

The parameter list of the macro core2 is used as an interface to all relevant attribute specifications. In the definite clause pass2 the macro core2 is called twice, once for the input and once for the output. For both calls the same variables are passed as arguments, ensuring identity between the relevant attribute specifications of input and output.

Besides the PVP lexical rule driving the analysis of partial VP topicalization, several others lexical rules were included in the implementation to obtain a simple and consistent grammar: A

[52]The same reasoning holds for the constraint on the elements which can undergo argument raising on the comps list of auxiliary verbs. Just as in the PVP extraction lexical rule case, the grammar implemented restricts them to be nominal signs.

morphological rule relating the negative indefinite "kein" to the various forms of "ein"; several morphological rules with syntactic effect obtaining finite, past participle, and substitute infinitive forms of verbs; syntactic rules introducing unbounded dependencies in lexicalized style for complement and subject extraction; and finally a rule freeing the complement order of ditransitive verbs.

Linguistic generalizations and computational compactness

In the last sections we have been concerned with mechanisms for expressing generalizations over the lexicon by relating lexical entries. We sketched the functionality desired by the linguists, showed that a formal mechanism inside of the logical setup of HPSGII does not capture all of these desiderata, and displayed the computational mechanism used in ALE which is equivalent to the description language mechanism regarding the functionality captured. The discussion so far was focussed on how the linguistic generalization are expressed, i.e. on the functionality of the mechanism.

Regarding computational mechanisms for expressing lexical rules, an issue besides the functionality offered deserves some discussion. From the computational point of view it is important whether the methods used to express the linguistic generalizations have correspondents on the computational side. In other words, in an ideal computational setup, the methods used to express linguistic generalizations should have a counterpart in a compact and elegant computational encoding which can be dealt with using efficient algorithms. We have seen that in the ALE system there is no such correspondence. The lexical rules are only used to produce new lexical entries for a bigger lexicon, which is then used in processing. This is particularly inelegant since the new entries produced usually only differ in very few specifications, namely those altered by the lexical rules.

There seem to be two possibilities to improve on this situation. One could use the ALE mechanism with an alternative processing strategy. It would supply new lexical entries whenever a lexical entry is needed in the local tree being processed.[53] However, such a straightforward application of lexical rules "on the fly" is very inefficient since every time something needs to be licensed by a lexical entry all lexical rules have to be applied in all possible orders to collect all possibly matching entries. A more complicated scheme could use some kind of tabelling method to keep track of the lexical entries already produced. But since it is completely unconstrained which specifications can be changed by a lexical rule, in the general case such a tabelling method will result in compiling out the lexicon upon the first lexical lookup to make sure that all possible solutions are found. Naturally this is even less desirable than compiling out the lexicon beforehand.

The other possibility uses the insight that the lexical entries resulting from lexical rule application usually only differ in very few specifications compared to the number of specifications in the lexical entry as a whole. Instead of expressing relations between lexical entries – which in a formal mechanism causes significant problems regarding the transfer of unchanged specifications from input to output – one can use a single lexical entry and encode the possible variations in that entry. Note that the variations cannot be captured with simple disjunctions since the value of one specification might depend on the value of another. Let us therefore call them systematic covariations.

[53] The same effect follows from using the description language mechanism introduced above in a group 1 setup.

Using relations to encode systematic covariation in lexical entries

The question we are interested in is how the lexical rules proposed by the linguists can be (automatically) translated into a system of relational dependencies introducing systematic covariation within lexical entries. A full discussion of this topic is beyond the scope of this paper. We will therefore only sketch an approach using the relational extension of the description language. As an alternative, we will briefly introduce an extension of the description language with so called "named disjunctions" in the next section.

The lexical rules and the mechanism for introducing systematic covariation are intended to capture the same generalizations. Still, they are very different in the way they work: The systematic covariation encoded as relational dependencies in the lexical entries capture the difference between a "basic" lexical entry and all those lexical entries which are the result of (successively) applying the lexical rules to that basic entry. Or in more computational terms: in the covariation approach, calls to definite clauses are attached to lexical entries. The definite clauses introduce disjunctive specifications in a lexical entry.

Lexical rules on the other hand relate lexical entries. Compared to the covariation setup they lift the relational dependencies from within a lexical entry up to the level of the lexical entries. This allows us to generalize over whole classes of lexical entries. These kind of generalizations over classes of lexical entries seem to be lost in the covariation approach, since the covariation is encoded inside of each lexical entry. This is not true though if we can use a call to the same relation in all or a natural class of lexical entries to introduce the intended covariation. To be able to define such a relation, we need to know which lexical rules can apply in what order on which lexical entries. This information can be retrieved from the input and output specification of the lexical rules. By checking which input of a lexical rule is compatible with which output, we can obtain a graph representation of all possible application orders of lexical rules to an entry. Figure 43 is such a graph for the lexical rules which in the implemented grammar apply to the class of verbal entries.

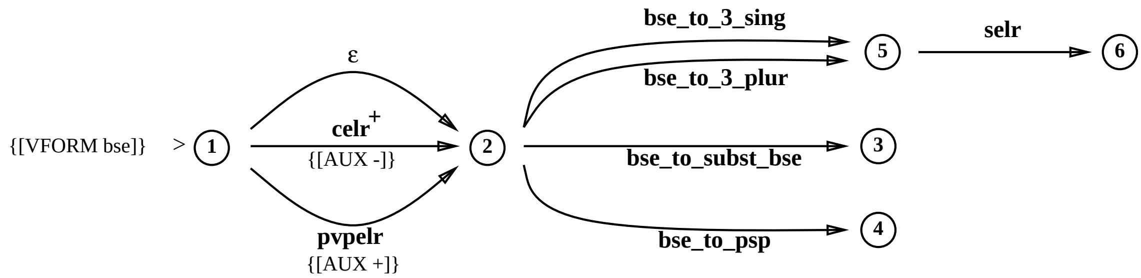

Figure 43: The lexical rules applying to verbal elements and their interdependence

Each edge corresponds to the application of a lexical rule[54]. Each node relates to one solution of a definite clause attached to a lexical rule. The specifications in curly brackets describe the set of lexical entries for which the graph as a whole or a certain edge is applicable.

The graph displayed applies to verbal entries with *bse* verb-form, which in fact are the only verbal

[54]The abbreviated lexical rule names have the following expansions: subject extraction (SELR), partial vp extraction (PVPELR), finitivization (bse_to_3_sing, bse_to_3_plur), finitivization (finit.), and past participle formation (bse_to_psp, bse_to_subst_bse). The celr$^+$ is the complement extraction lexical rule modified in such a way that it allows the extraction of any number of complements greater than 0. It for example only has to be applied once to extract both objects of a ditransitive verb. The ϵ stands for no change.

entries specified in the basic lexicon of the fragment. The PVPELR only apply to auxiliary verbs and the CELR only to full verbs. The restrictions which are already ensured by the specification of the output of a lexical rule do not need to be listed separately. For example the SELR only applies to finite verbs. Since the finitivization rules leading to node 5 ensure that their result is finite, no additional specifications are necessary.

For the issue we are pursuing, the graph can be used as backbone for an encoding of the lexical rule regularities in the relations called in the lexical entries. Such a backbone is constructed by encoding the graph as finite state automaton in one definite clause which is attached to every lexical entry. Every node is a solution, i.e. a possible lexical entry, and every label of an arc is a call to a relation encoding the variation introduced by the corresponding lexical rule.

In one of the Troll versions of the grammar Guido Minnen and I encoded lexical rules in the way described. This drastically reduced the number of lexical entries compared to the original compiled out ALE lexicon. The result is a lexicon which captures at least some of the linguistic generalizations behind the lexical rules in a computationally usable way.

An alternative mechanism: named disjunction

Before we leave the area of generalizations over lexical information, let us briefly introduce an alternative mechanism using an extension of the description language for expressing systematic covariation in lexical entries: named disjunctions of complex descriptions used in the specification of lexical entries.

To illustrate this mechanism we will look at an example from the interaction of morphology and syntax. In HN (adapting a proposal from Pollard (1990)) base verbs differ from finite verbs in three respects. The subject of a finite verb is encoded together with its complements, whereas that of a base entry is specified in the separate attribute SUBJ. Nominative case as well as number and person information is specified for the subject of finite verbs, but not for that of base ones. Finally the verb form is encoded in the attribute VFORM. A fully expanded lexicon would contain seven unrelated entries and the relation between these would be lost. Note also that the described complex interaction of morphological and syntactic features of finitivization cannot be encoded using simple disjunctions, since the case and agreement assignment, as well as the valence encoding depends on the verb form. Figure 44 shows the resulting named disjunction encoding.

$$
{}_{cat}\begin{bmatrix}
\text{HEAD} & {}_{verb}\begin{bmatrix}\text{VFORM} & \{_{\text{v}}\ \mathit{fin}\ ;\ \mathit{bse}\}\end{bmatrix} \\
\text{VAL} & \begin{bmatrix}
\text{SUBJ} & \{_{\text{v}}\ <>\ ;\ <\boxed{1}>\} \\
\text{COMPS} & \{_{\text{v}}\ <\left(\boxed{1} \wedge \begin{bmatrix}\rightsquigarrow\text{CASE} & \mathit{nom}\end{bmatrix}\right) . \boxed{2}>\ ;\ \boxed{2}\} \\
\text{SUBCAT} & <\boxed{1} . \boxed{2}>
\end{bmatrix}
\end{bmatrix}
$$

Figure 44: A named disjunction expressing finitivization à la Pollard (1990) (*cat* value shown)

The notation needs some explanation. The $\rightsquigarrow$F abbreviates the path starting with SYNSEM down to the attribute F. The angle brackets are used as list notation, to distinguish them from the square brackets of the AVMs. Named disjunctions are noted as $\{_{\text{name}}\ disj_1\ ;\ disj_2\ ;\ \ldots\ ;\ disj_n\}$. A named disjunctions is interpreted in the following way: In every disjunction with the same name, the disjunct in the same position has to be chosen.

In the figure 44 the named disjunction v encodes the described dependence. If the first disjunct of the named disjunction v is chosen, the finite verb-form is selected, the SUBJ attribute is assigned the empty list, and the subject as first element on the COMPS list receives nominative case. In case the second disjunct is chosen, the VFORM is *bse*, the SUBJ attribute contains the singleton list with the subject as element, and only the complements are encoded in COMPS. The subject and the complements of the finite and the non-finite choice are related, since the SUBCAT list is outside of the disjunction.

If the specification in figure 44 is included in the grammar as constraint on signs with verbal head value, an example lexical entry for a verb can be specified as in figure 45.

$$ {}_{word}\begin{bmatrix} \text{PHON} & \{_w\ \{_n\ \{_p\ \textit{liebe} ; \textit{liebst} ; \textit{liebt}\} ; \{_p\ \textit{lieben} ; \textit{liebt} ; \textit{lieben}\}\ \} ; \textit{lieben}\} \\ \leadsto\text{VFORM} & \{_w\ \textit{fin} ; \textit{bse}\} \\ \leadsto\text{SUBCAT} & < \begin{bmatrix}\leadsto\text{IND} & \boxed{1} \wedge \begin{bmatrix}\text{NUM} & \{_n\ sg ; pl\} \\ \text{PER} & \{_p\ 1st ; 2nd ; 3rd\}\end{bmatrix}\end{bmatrix} , \begin{bmatrix}\leadsto\text{IND} & \boxed{2}\end{bmatrix} > \\ \leadsto\text{CONT} & {}_{love}\begin{bmatrix}\text{LOVER} & \boxed{1} \\ \text{LOVED} & \boxed{2}\end{bmatrix} \end{bmatrix} $$

Figure 45: An example entry for “love”

The lexical entry specifies the semantics and identifies the thematic roles with the syntactic indices in the standard fashion. Additionally the named disjunction mechanism takes care of two things. The disjunction w takes care of the choice between finite and base phonologies by relating the phonology value to the value of VFORM. In the case of a finite verb there is an additional choice regarding the agreement between the index of the subject and the phonology of the verb. The named disjunction n mediates the choice between singular and plural, and p links the person specification on the index to the correct phonology. If instead we are dealing with a base verb, the named disjunctions n and p only occur once, namely in the specification of the subject index, since the second disjunct of w is chosen. Such a single occurrence of a named disjunction is equivalent to a single disjunction. The index of the subject can therefore have any of the values noted.

The encoding using named disjunctions bears a close resemblance to encodings on the relational level discussed above, e.g. in the form of definite clause attachments to lexical entries. Nonetheless, there is a difference worth noting: The named disjunction mechanism is weaker than the definite clause mechanism in that it does not allow recursive definitions. Only simple covariation can be expressed, which is exactly what is needed for the desired encoding.

Summing up, the advantages of the named disjunction mechanism are its declarative semantics, its restricted expressive power corresponding to the needs, and the resulting compact encoding of linguistic generalizations in a computationally usable way. Which class of lexical generalizations can be captured in this way, should therefore be interesting to investigate.[55] For a

[55]The reader interested in constraints on named disjunctions as models for lexical rules should note that the example given crucially depends on two things. To relate the information of one disjunct of a named disjunction to another disjunct of a disjunction with the same name, the disjuncts have to structure share with an object outside of the disjunction. In the example this is done via the SUBCAT list, which contains the subject of both finite and non-finite verbs. The second point to mention concerns the disjunction names. These names only make sense local to a description. Since the example is based on a constraint applying to signs with verbal heads and a

detailed elaboration of a framework and computational system for named disjunction the reader is referred to Griffith (forthcoming).

Concerning the computational systems, it remains to be mentioned that Troll currently only supports named disjunction of types, while ALE does not support them at all.

4.2 Structure licensing

In HPSG, constituent structure is licensed via *immediate dominance schemata* and *linear precedence rules*. Using a pure constraint logic programming approach to computation, this mechanism can be directly encoded. Such an approach, however, leaves us with two problems: one regarding the control structure to be applied in constraint resolution - since the special guiding status once attributed to fixed information on phrase structure is now lost in a vast pool of non-distinguishable constraints; the other in licensing linear order with LP statements which makes strong demands regarding the expressive power of the constraint language.[56]

The alternative, traditional approach to structure licensing uses the information about the number of daughters of a certain construction – and optionally other information specified on constituents together with the required word order – and encodes it in *phrase structure rules*. Generalizations over that information are lost.[57] The obtained phrase structure backbone can then be used to drive highly specialized efficient algorithms to license structures. Finally, to express some general constraints on rules, definite clauses can be attached to them. In a phrase structure backbone architecture with definite clause attachments, the number of daughters is the only information which - compared to an ID/LP approach - has to be additionally specified.[58] All the specifications on constituents distinguishing one constituent from the other and thereby specifying a certain ordering as well, can theoretically be included in the definite clause attachments. Nonetheless parsing algorithms are more efficient, the more information is present on the constituents.

While the traditional approach has the clear disadvantage of differing from the ID/LP format used in HPSG theory, it nevertheless leaves the implementer using the current parser based systems[59] with an interesting choice regarding generalization vs. performance: generalizations over rules can be captured by encoding the information in a definite clause which is attached to all the rules generalized over. The performance drawback of this encoding is that - since (in general) definite clauses are executed after a construction has been accepted by the parser - they do not contribute to the information available to the parsing algorithm.[60] This is a disadvantage for processing, since as mentioned the phrase structure backbone approach is faster the more information is specified per constituent in a rule, while the exact algorithm used determines the constituent (e.g. leftmost constituent) on which information specification has the most effect.

description representing the lexical entry, the choice of the verb-form has to be repeated in the lexical entry itself in order to mediate the choice between the constraint and the lexical entry.

[56] Cf. Meurers and Morawietz (1993) for a discussion of the LP formalism.

[57] For an approach introducing some word order generalizations into a phrase structure grammar, the reader is referred to the SUP formalism introduced in Meurers and Morawietz (1993).

[58] The operator "cats>" supplied by ALE allows to specify a list of daughters, which at compile time do not have to be of fixed length. Nonetheless it does enforce the number of daughters to be fixed at run time.

[59] In principle, parsers can be extended to ID/LP rules and possibly to ID/LP schemata as well. Cf. Shieber (1984), Seiffert (1991), and Morawietz (1994).

[60] Note that partial execution of definite clauses at compile time can make further information available for the parser.

4.2.1 An example for phrase structure rules with the feel of ID schemata

In the following, one solution to the generalization vs. performance choice is presented using an example from the implementation of HN. The general idea is to use macros and a definite clause to express the generalizations over the group of PS rules which replace one ID schema. Regarding terminology, we will speak of the macro and definite clause as the *core* of a *group* of PS rules which encode one ID schema.

Generalizations over PS rules are captured in two ways: On the one hand, the specifications valid for all constituents of a certain function (e.g. head, complement, or mother) are collected in a macro which is used in all occurrences of a daughter of that function in every rule of a group. The specifications specific to a rule are captured via parameters passed to the macro or by conjoining the specific descriptions to the call of the macro. On the other hand, generalizations over constraints on the whole tree (like the grammatical principles discussed in the next section) are expressed in a definite clause which is attached to all PS rules of one group. These two methods for the specification of phrase structure rules conserve at least some of the generalizations and intuitions behind the original ID schemata of the linguistic theory.

To illustrate how the ID/LP constraints of HN were encoded as PS rules using the method described, let us take a look at the *head-complement* ID schema.[61] This ID schema licenses constructions with up to five daughters and a highly varied word order depending on the specification of the head (e.g. AUX and INV specification on verbal heads) which - as follows from the discussion above - leads to a high number of phrase structure rules.[62] The head-complement ID schema of HN is shown in figure 46.[63]

$$\begin{bmatrix} \text{SYNSEM|LOC|CAT} & \begin{bmatrix} \text{HEAD} & \textit{verb} \\ \text{COMPS} & \langle\rangle \\ \text{NPCOMP} & \text{plus} \end{bmatrix} \end{bmatrix} \rightarrow \begin{array}{l} \text{H} \ [\text{NONLOC|TO-BIND|SLASH} \ \ \langle\rangle], \\ \text{C} \ [\text{SYNSEM|LOC|CAT|HEAD} \ \ \neg\textit{verb}]^{+}, \\ (\text{C} \ [\text{SYNSEM|LOC|CAT|HEAD} \ \ \textit{verb}]). \end{array}$$

Figure 46: The head-complement ID schema of HN

In the implementation the specifications of the mother and the head category of this ID schema are captured in the following core definitions, which apply to all PS rules of the head-complement

[61] In HN verbal projections are licensed by the *verbal-complex* and the *head-complement* schemata. Because of the valence encoding issue discussed in section 5.1, the implementation additionally distinguishes a group of *head-subject-complement* rules for finite sentences from the *head-complement* group of PS rules, which in the implementation only licenses non-finite VPs and finite sentences with extracted subjects.

[62] While a high number of rules naturally slows down performance, some performance can be regained, since some of the rules become so specific that it is known which class of lexical heads are possible in which rule. Therefore more information can be specified in the rule, speeding up the system. It should be clear though that this additional specification of information already encoded in the lexical entries is a theoretically unmotivated repetition. Trying to lose this redundancy by only specifying the information in the rules, conflicts with the increasing lexicalization going on in linguistics today. Phrase structure based formalizations therefore seem to be inappropriate for modern lexicalized theories. Still, future pre-compilation techniques might be able to automatically add additional specifications to phrase structure rules, which can be deduced from interaction with the specification of the lexicon used.

[63] Because the head-complement ID schema of HN depicted in figure 46 (= figure 21 of HN) demands the occurrence of at least one non-verbal complement, HN also call it Head-NP-Complement ID.

group.

```
hc_mother(S)        macro (@verb,
                           @comps_sat,
                           @plus_npcomp,
                           @val_subj(S), @spr_sat).

hc_head(S,C)        macro (@no_to_slash,
                           @val_subj(S), @spr_sat, @val_comps(C)).
```

Figure 47: The macro core of the *head-comp* group

Before going through the specifications, a short note on the notation used might be appreciated.[64] Variable names, serving the same function as the structure sharing tags in HPSG, are noted with a capitalized first letter. Names following an "@" character are calls to macros, and will be replaced at compile time by the complex description which the macro abbreviates. The type of the description abbreviated is noted in the name of the macro. Macros with a suffix "_y" are of type *synsem*, those without suffix (all the macros in figure 47) of type *sign*. Finally, the commas within the parenthesis are interpreted as conjunction (except when separating the arguments of a macro), and the square brackets represent the normal list notation of PROLOG.

Comparing the core definitions of the implementation with the original formulation of the Head-Complement ID schema we note a close similarity: The macro `hc_mother`, which specifies all the information on the mother of each PS rule of the head-complement group, contains the specifications present on the mother of the ID schema in figure 46. In addition – for reasons discussed below – all valence information is included, which accounts for the extra `@val_subj` and `@spr_sat`. The same holds for the head: like the head in the ID-schema, the macro `hc_head` contains a TO-BIND specification. In addition the valence information is accessed. The specification of the one optional verbal and the more than one nominal complements in figure 46 cannot be elegantly expressed in a macro for the complement function. The number of complements is not fixed in the schema. Still certain properties of possibly occurring complements need to be specified, which makes it necessary to fix a certain order of complements. The complements therefore need to be specified separately in each PS rule of the head-complement group.

Figure 48 shows one PS rule of the *head-comp* group which makes use of the macro core. First, some comments on the ALE notation used: The operator `===>` separates the mother of a phrase structure rule from the daughters. Each daughter is prefixed with `cat>` and ends with a comma. Finally, the `goal>` operator allows us to add calls to definite clause relations to the phrase structure rule defined. The phrase structure rule can only be used to license a local tree, for which the relational constraints attached to the rule are satisfiable.

[64]The macro mechanism of ALE was introduced in section 4.1.2 on pp. 26. For a full definition of the ALE syntax, the reader is referred to Carpenter (1993). A set of comments on the additional conventions used in our grammar can be found in section 5.4.2.

```
hc_minus_inv2 rule
   (Mother, @hc_mother(S)) ===>
      cat> (C2, @obj_np),
      cat> (C1, @obj_np),
      cat> (Head, @hc_head(S,[C1,C2]), @n_fin, @minus_inv),
      goal> hc_goal(hcni2, Mother, Head, [C1,C2]).
```

Figure 48: Phrase structure rule of the *head-comp* group

This rule licenses non-inverted non-finite VP constructions in which a verbal element combines with two complements. For the mother category of the construction, only the information valid for the whole HC group (the core specification defined in figure 47) is specified via the call to the macro `@hc_mother`. The variable `Mother` is conjoined to that specification to feed the definite clause `hc_goal` discussed below. Similarly, the macro `@hc_head` is called as description of the head, and the variable `Head` is conjoined to the whole specification to be used in the call to the definite clause. Two more things should be mentioned. For reasons that will be explained in the next section, the valence principle is compiled into each rule. The argument list of the mother macro (`S`) and that of the head macro (`[S],[C1,C2]`) serve to mediate the valence information. The conjuncts `@n_fin` and `@minus_inv` in the description of the head daughter specify that only non-finite non-inverted heads can appear in this rule. This is not included in the general `@hc_head` specification, since the HC-ID also licenses inverted or finite constructions. The specification of the complement daughters as nominal complements results from two sources. The HN specification of the head-complement ID schema of figure 46 demands there to be at least one non-verbal complement and an inspection of the lexical elements which can appear as heads in this rule tells us that both complements have to be object noun-phrases (`@obj_np`).[65] At the bottom of the rule, the goal expressing the constraints applying to all HC constructions (`@hc_goal`) is called and the rule specific specifications (the name of the rule for identification in the parse tree, the mother, the head, the subject and the list of complements) are passed as arguments. The definition of the definite-clause core for the *head-comp* group is shown in figure 49.

```
hc_goal(Subtype, Mother, Head, Comps) if
   dtrs(Mother, @head_comp_cs(Subtype, Head, Comps)),
   principles(Mother, Head, Head, [Head|Comps], Head).
```

Figure 49: The definite-clause core of the *head-comp* group

The call to the *dtrs* relation serves to build up a parse tree. It is included in the definite clause core and not in the PS rules in order to separate it as much as possible from the linguistically motivated specifications. The call to the principles on the other hand serves as universal attachment of the grammatical principles. All information necessary for the principles is supplied via

[65] This is a good example for the additional specifications in phrase structure rules mentioned in footnote 62.

the five arguments. This allows us to use exactly the same *principles* relation for all PS rules. We will see in the next section how the principles are encoded in this relation.

Finishing up the issue of encoding dominance relations, it remains to be mentioned that in the grammar the head-complement schema, the verbal-complex schema, and the extra head-subject-complement schema are encoded using the mechanism just described. For each of the other ID schemata in the fragment (marker-head, specifier-head, adjunct-head and filler-head) there is a one to one correspondence between PS rules and ID schemata, since they introduce a fixed number of daughters.

4.3 Expressing the principles

In the last section we saw two options for specifying constraints on a construction: directly encoding the constraints in a rule ensuring maximal performance, and the usage of definite clauses yielding maximal generality. When deciding how to encode the principles of a grammar (e.g. the head feature principle or the semantics principle of HPSGII) we are again faced with those two possibilities.[66]

Due to the decision to stay as close to the linguistic theory as possible, the head feature principle (HFP), the nonlocal feature principle (NFP), the semantics principle (SemP), and the marking principle (MarkP) were each encoded in a separate definite clause and then conjoined in a clause "principles" which then was attached to all groups of PS rules. Regarding processing, not specifying these principles in the phrase structure backbone has only a small negative effect, since these principles mostly relate information in the mother to information in the daughters, as opposed to expressing relations between the daughters. Since most of the information originates in the lexicon, the bottom-up strategy employed by the systems used would only marginally profit from the information mediated by those principles. The interface between a rule and the principles consists of five arguments: Mother, Syntactic_Head, Semantic_Head, List_of_Dtrs, and Marking_Dtr[67]. This encoding allows an elegant general encoding of the principles, without having to specify choice procedures (e.g. picking out the semantic head) depending on the type of the construction. Figure 50 shows the definition of the principles in the implemented grammar.

```
principles(Mother, Head, Sem_head, Dtr_list, Marking_dtr) if
   (hfp(Mother, Head),
    nfp(Mother, Head, Dtr_list),
    sem_p(Mother, Sem_head, Dtr_list),
    marking_p(Mother,Marking_dtr)).
```

Figure 50: Definition of the definite clause *principles*

The head feature principle (HFP) is encoded as displayed in figure 51.

[66] I here only note the possibilities in a phrase structure backbone based system. As discussed in the section 3.2 on expressing constraints, the principles in HPSGII are expressed as implicative constraints just like the rest of the grammar. In a general group 1 constraint resolution system one has the choice to directly encode the principles as object definitions à la HPSGIIor to compile the principles into the object definitions expressing the immediate dominance schemata. Choice here is influenced by the strength of constraints that can be expressed in the language used (e.g. if complex antecedents are available) and the question of when which kind of constraint is resolved (compile or runtime).

[67] The Marking_Dtr is the marker of a head_marker construction and the head in any other construction.

```
hfp(@head(X),@head(X)) if true.
```

Figure 51: Definition of the definite clause *hfp*

When we compare this definition with the HFP as given in HPSGII (which, for the sake of convenience, is repeated below), we note that only the consequent of the implication is expressed in the definite clause. The antecedent is implicit in the encoding, since the definite clause *principles* is only attached to PS rules licensing phrases dominating headed structures.[68]

$$_{\textit{phrase}}\left[\text{DTRS } \textit{headed-struc}\right] \longrightarrow \begin{bmatrix} \text{SYNSEM|LOC|CAT|HEAD} & \boxed{1} \\ \text{DTRS|HEAD-DTR|SYNSEM|LOC|CAT|HEAD} & \boxed{1} \end{bmatrix}$$

Figure 52: The HFP

Note that the HFP encoding does not actually make use of the expressive power of the definite clause mechanism. Were we not interested in capturing generalizations as discussed above, the HFP could be encoded directly in the PS rules. This is different for the nonlocal feature principle (NFP) shown in 53.

```
nfp(@in_slash(Diff), @to_slash(Head_To_slash), Dtr_list) if
   (collect_in_slash(Dtr_list,In_slash),
    list_diff(In_slash,Head_To_slash,Diff)).
```

Figure 53: Definition of the definite clause *nfp*

Again the definition from the appendix of HPSGII is included below. Since in the fragment we are only concerned with the nonlocal feature SLASH, the references to the other nonlocal features QUE and REL are omitted.

> In a phrase whose DAUGHTERS value is of sort *headed-structure*, the value of SYNSEM|NONLOCAL|INHERITED|SLASH is the set difference of the union of the values on all the daughters and the value of SYNSEM|NONLOCAL|TO-BIND|SLASH on the HEAD-DAUGHTER.

Figure 54: The NFP for the nonlocal feature SLASH

Note that this time it is not easily possible to rewrite this text definition as an AVM. The "union of the values on all the daughters" cannot be directly expressed, since each subtype of *constituent-structure* has different attributes in which the daughters of a construction are stored. In the implementation, the list of all daughters therefore is one of the arguments passed to the principles, which enables us to use a uniform, recursively defined relation `collect_in_slash` to

[68] The fragment does not contain any non-headed structures. Therefore the definite clause principles can be attached to all PS rules.

collect the INHERITED|SLASH values of all daughters. Note that – for reasons explained in section 5.2 – in the implementation SLASH has a list instead of a set value. The "set difference" of the original definition therefore becomes the `list_diff` relation in figure 53. Without going into the definitions of the two predicates doing the collecting and the calculation of the difference, it should be clear that specifying these operations on lists of undetermined length requires the extra expressive power of the definite clause mechanism. As a last comment on the encoding of the NFP note that the call to the macros `@in_slash` and `@to_slash` in the first two arguments of `nfp` serve to select the INHERITED|SLASH value of the mother sign, which is passed as the first argument, and the TO-BIND|SLASH of the head sign, which is passed as the second.

The valence principle (VP) does not fit well into the schema used for encoding the principles as definite clause attachments on PS rules. As mentioned above, the HFP, NFP, SemP, and MarkP mainly relate information between the mother and the daughters. The VP additionally relates information between the head and the other daughters. Since this is valuable information for any processing regime, it is encoded separately in each group of rules without the usage of definite clauses. Additional motivation for this dividing up of the valence principle comes from the fact that it does not easily lend itself to a general formalization. The formulation in HPSGII uses variables over attributes and attribute name manipulation (e.g. to obtain SUBJ-DTR from SUBJ) both of which are neither part of the logic behind HPSGII nor that of ALE or Troll. Encoding the function of a sign in the value of an attribute of the sign - as proposed in Meurers and Morawietz (1993) or realized in Meurers (1993) - enables the grammar writer to refer to the function. All subcategorization requirements can then be encoded in a single attribute and all daughters of a local tree in a single list valued daughter attribute of constituent structure without losing the distinction between the different functions.[69]

5 Relating the specific linguistic theory to its implementation

This section illustrates some issues concerning the specific linguistic theory of HN which arose in writing the grammar. It explains where the implementation differs from the original theory.[70]

5.1 Encoding valence

In the discussion on German sentence structure about the existence of a verb phrase in German, Pollard (1990) proposes that German lacks a finite vp while it does have a non-finite one. This is technically reflected in the following way[71]: The subject of finite verbs are encoded together with the internal complements (in attribute SUBCAT), while that of non-finite verbs is specified in a separate attribute SUBJ. This lexical information is put to work via an id schema which

[69]To be able to quantify over all daughters - as is necessary to collect the nonlocal features in the nonlocal feature principle and the quantifiers in the semantics principle - one has to keep track of all daughters in a list anyway. However, as discussed in conjunction with the interface specification for the principles, this can be done without permanently encoding them using an additional attribute.

[70]Except for the differences commented on in the discussion of examples from the grammar in the previous sections, every difference between the linguistic theory and the implemented grammar is discussed below. This ensures that the work on the implementation of a theory as test for the rigid, explicit, and correct formalization of a linguistic theory can provide feedback for the further development of the linguistic theory.

[71]This is the same analysis of finitivization which has already been subject of some discussion to illustrate the named disjunction mechanism on pp. 40. Only the difference in constituent structure involved is relevant in the following discussion.

realizes all elements in one local tree which are subcategorized for in SUBCAT. A single ID schema now suffices to build finite sentences and verb phrases and - since no ID schema ever refers to the SUBJ attribute - one explains that only subjects of finite verbs can be realized. HN adopt this elegant technique.[72]

From an implementor's point of view the proposal loses at least some of its elegance. As referred to in section 4.1, HN propose an argument raising specification as part of the lexical entries of auxiliary verbs. Figure 55 shows the basic form of the argument raising specification of HN.[73]

$$\begin{bmatrix}\text{COMPS} & \boxed{3}\end{bmatrix} \quad \& \quad \text{append}(\boxed{1}, \begin{bmatrix}\text{SYNSEM|LOC|CAT} & \begin{bmatrix}\text{HEAD} & \textit{verb} \\ \text{COMPS} & \boxed{1}\end{bmatrix}\end{bmatrix}, \boxed{3})$$

Figure 55: The HN argument raising specification on a non finite auxiliary (*valence* value and relational attachment shown)

Looking at figure 55, one sees that argument raising introduces a tag in the lexical entry ($\boxed{3}$) which in the lexicon is completely unrestricted. The set of objects described by the first argument and the result argument of the definite clause *append* only get narrowed down by the usage of the lexical entry in a tree. The HN use of append in the lexical entry of auxiliary verbs therefore presupposes a rather sophisticated control mechanism for constraint resolution which executes a definite clause at the time its essential arguments are instantiated. Neither ALE nor Troll offered such control mechanisms at the time of the implementation.[74]

If the order of the COMPS list is reversed, the usual encoding of lists (in which the first element and the rest of a list can be directly accessed) results in an encoding which no longer needs the *append* relation. We can simply structure share the embedded verb's COMPS with the tail of the outer COMPS list. This change of the valence list order further reduces complexity by eliminating the calls to the definite clauses which were needed in local trees to access the last, most oblique element first since those usually get realized first.

However, ordering the elements on the subcategorization list by decreasing obliqueness (as in Pollard and Sag (1987)) is also problematic; again because of the argument raising mechanism. To see this, the argument raising specification with inverted comps list (cf. figure 17) is repeated as 56 below.

$$\begin{bmatrix}\text{COMPS} & \left\langle \begin{bmatrix}\text{SYNSEM|LOC|CAT} & \begin{bmatrix}\text{HEAD} & \textit{verb} \\ \text{COMPS} & \boxed{1}\end{bmatrix}\end{bmatrix} \middle| \boxed{1} \right\rangle\end{bmatrix}$$

Figure 56: Argument raising specification on a non-finite auxiliary (*valence* value shown)

If we try to formulate a lexical rule forming a finite auxiliary in the style of Pollard (1990) on the basis of the valence specification in 56, we are stuck with the impossible: trying to add the subject

[72] Note a minor notational change. HN use a COMPS attribute in the style of chapter 9 of HPSGII instead of the SUBCAT attribute used in Pollard (1990).

[73] HN use a functional notation for the append relation in their diagrams.

[74] The delay patterns of CUF are an example for a mechanism which would handle at least these kind of cases (cf. Dörre and Dorna (1993)). The newest version of Troll offers a rather sophisticated freeze mechanism as well.

as the least oblique element to the end of a list of undetermined length. Even the desperate move of putting the subject into the head of the list (which corrupts the theoretically motivated obliqueness ordering) is no solution, since then - due to the now in general undetermined position of a possibly occurring verbal-complement - we need to separate phrase structure rules for finite and non-finite verbs, or for auxiliaries and main verbs. This, however, is exactly which we intended to avoid via encoding the subject on COMPS.

Concerning the implementation and in the light of the problems with an argument raising specification using a call to append, it therefore seems to be cleaner to keep the subject encoded in SUBJ for non-finite and finite verbs, which allows easy list manipulation with a COMPS list ordered by decreasing obliqueness.

Since under this approach one needs to separately license constructions that realize subjects and those that do not, a *head-subject-complement* ID schema is introduced in the grammar to license finite sentences containing subjects. The *head-complement* ID of the implemented grammar therefore only licenses non-finite VPs and finite sentences with extracted subjects.

5.2 Specifying SLASH sets depending on COMPS lists

A problem similar to the use of append in the lexical entries of auxiliaries discussed above arises from the formulation of the PVP-Topicalization Lexical Rule in HN. In order to relate the list-valued COMPS list to a set-valued SLASH list HN attach a relation *same-members* to the lexical rule (cf. figure 19 of HN). The definite clause *same-members* can only be executed once it is clear which constituents have been syntactically realized where. In the terminology of lexical rules of meta-relations, the same-members specification therefore only works with unification as application criterion. Regarding the computational treatment of lexical rules, it demands that the lexical rules be not completely compiled out, but at least partly computed "on demand" while building the syntactic tree. Even if this should turn out to be possible[75] such an approach would demand a highly sophisticated control structure.

Since one might want to argue anyway that the embedding constraint on extraction is to be captured in syntax, making the value of SLASH a last-in-first-out list store is a possible solution to our problem, since it allows us to replace the *same-members* relation in the PVP extraction lexical rule by a simple token-identity constraint. The pragmatic solution chosen for the grammar implemented therefore is to change the value of the SLASH attribute to be a list instead of a set. The resulting lexical rule was already shown in figure 40 of p. 36.

5.3 The Cooper-storage mechanism

In the HPSGII theory a Cooper-storage mechanism is used to calculate all scopings of the quantifiers in a sentence. Since this mechanism is integrated into the HPSG setup just like the rest of the constraints making up the grammar, in a straightforward implementation all scopings of quantifiers will be computed at the same time as the grammatical constraints are applied to a local tree. This leads to the same local tree being admitted several times, the only difference between the trees being the number of quantifiers retrieved. This is an undesirable

[75]The discussion of the lexical rule mechanism in section 4.1.3 suggests that there are some severe theoretical and computational problems for such an approach.

effect, since these local trees are identical for all syntactic properties that play a role regarding the grammaticality of the trees.

The concept of an underspecified discourse representation defined in Reyle (1993) replaces the forced computation of all possible quantifier scopes with a representation of quantifiers underspecified for scope. Restrictions on the scope relations can then be formulated on the basis of the underspecified representation. This setup avoids an unmotivated multiplication of syntactic structures while at the same time introducing an elegant mechanism for expressing theories about the semantic scope of quantifiers. Frank and Reyle (1992) show that this setup can be transferred to HPSG and implemented in an HPSG grammar.

Since the implementation of HN documented here is concerned with syntactic phenomena only, the semantics included in the grammar simply follows the proposals made in HPSGII. However, in the semantics principle an additional specification ensures that no quantifiers are ever retrieved. Every sentence therefore contains the collection of all its quantifiers in its QSTORE attribute as a simple kind of a representation underspecified for quantifier scope. This way we obtain a workable setup avoiding the multiple structures with the help of a minor modification.[76]

5.4 Technical comments on the implementation

The appendix contains the complete ALE grammar including a collection of test sentences for all relevant constructions. Additionally the printout of a test run is included to illustrate the performance of the grammar in the ALE system.

5.4.1 The lexicon

The lexicon of the implementation contains the entries necessary to follow the analyses of aux-flip and partial verb phrase topicalization in V1, V2, and VL sentences. As discussed in section 4.1.2, the implementation contains a hierarchy of lexical macros allowing an easy extension of the lexicon. All lexical entries are specified using calls to the lexical macros.

After lexical rule application, the lexicon contains the following entries:

Complementizer

daß (that)

Adjectives

klein (small), gut (good)

Determiners

definite: *der (the)*; indefinite: *ein (a), einige (some)*; negative indefinite: *kein (no)*

Nouns

Mann (man), Tisch (table), Buch (book), Geschichte (story), Verwandter (relative), Karl, Peter,

[76] Naturally, the modification of the semantics principle made is not intended to be a theoretical proposal. It merely is a loose link between an HPSGII-style semantics and an underspecified representation of quantifiers inspired from Reyle (1993) and Frank and Reyle (1992). In any more semantic-oriented grammar fragment that link would have to be firmly established.

Anna, Sarah

Adjectives, determiners and nouns are included in all possible number, gender, case, and declension pattern specifications.

Main verbs

intrans: *lachen (laugh)*; trans: *lieben (love), finden (find), lesen (read)*; ditrans: *geben (give), erzählen (tell)*

Auxiliaries

haben (have), werden (will), können (can), müssen (must), wollen (want to)

Verbs are given in their third person singular and plural, finite, past participle[77], base forms, and (for auxiliaries) the substitute infinitive forms.

5.4.2 Macros and notational conventions

Many abbreviations used by HPSG linguists serve to provide easy access to information deeply embedded in descriptions. The grammar contains an abbreviation of type *sign* and one of type *synsem* (which are the types of objects most often talked about in the grammar) for all relevant properties. The result is that the descriptions in the implementation often closely resemble the specifications in HN.

Since macros contain descriptions of objects of a specific type, the type of a macro is encoded in its name. Suffix "_y" is used for macros of type *synsem* and macros without suffix are of type *sign*. Those macro names ending in "_lex" are macros of type *sign* belonging to the lexical hierarchy used to specify lexical entries.

It would be of great value to the user if systems would automatically infer the correct feature path at which a specification belongs.[78] For example the description a) below could automatically be expanded to that given in b).

a) $word \wedge verb \wedge plus\_aux$

b) $_{word}\left[\text{SYNSEM|LOC|CAT|HEAD} \quad _{verb}\left[\text{AUX} \quad \text{plus}\right]\right]$

Clearly some notational conventions (e.g. for the *plus_aux* case above) and some decisions for the search strategy (e.g. shortest path wins) have to be made, but there seem to be no problems which render such an extension impossible. Such a notational mechanism would eliminate many of the usages of macros and the syntactic errors commonly made by the grammar writer in defining those macros.

[77] Only past participle formation with "haben" (have) is included. Therefore the past participle of *werden* is not part of the lexicon.

[78] A similar suggestion was made by Jo Calder at the 1st International HPSG Workshop in Columbus, Ohio, July/August 93.

References

Aït-Kaci, H. 1985. *A New Model of Computation Based on a Calculus of Type Subsumption.* PhD thesis, University of Pennsylvania, Philadelphia, PA, USA.

Carpenter, B. 1992. *The logic of typed feature structures.* Vol. 32 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press.

Carpenter, B. 1993. ALE – The Attribute Logic Engine, User's Guide. Technical report, Laboratory for Computational Linguistics, Philosophy Department, Carnegie Mellon University, Pittsburgh, PA 15213, May.

Dörre, J., and M. Dorna. 1993. CUF – a formalism for linguistic knowledge representation. In *Computational aspects of constraint based linguistic descriptions I*, ed. J. Dörre, 1–22. Universität Stuttgart: DYANA-2 Deliverable R1.2.A, August.

Frank, A., and U. Reyle. 1992. How to Cope with Scrambling and Scope. In *Konvens 92, 1. Konferenz "Verarbeitung natürlicher Sprache"*, ed. G. Görz. Springer.

Gazdar, G., E. Klein, G. Pullum, and I. Sag. 1985. *Generalized phrase structure grammar.* Cambridge, Massachusetts: Harvard University Press.

Gerdemann, D., and P. J. King. 1993. Typed Feature Structures for Expressing and Computationally Implementing Feature Cooccurence Restrictions. In *Proceedings of 4. Fachtagung der Sektion Computerlinguistik der Deutschen Gesellschaft für Sprachwissenschaft*, 33–39.

Gerdemann, D., and P. J. King. 1994. The Correct and Efficient Implementation of Appropriateness Specifications for Typed Feature Structures. In *Proceedings of COLING-94. Kyoto, Japan.*

Götz, T. W. 1994. A normal form for typed feature structures. Arbeitspapiere des SFB 340 Nr. 40, Universität Tübingen.

Griffith, J. forthcoming. *Disjunction and Efficient Processing of Feature Descriptions.* PhD thesis, Universität Tübingen. Tentative Title.

Hinrichs, E. W., and T. Nakazawa. 1991. Linearizing Finite Aux in German Complex VPs. To appear in: Nerbonne, John, Klaus Netter and Carl Pollard (eds.): *German Grammar in HPSG*, CSLI Lecture Notes. Paper presented at the Conference 'HPSG in German' held at the University of the Saarland, Saarbrücken.

Hinrichs, E. W., and T. Nakazawa. 1994. Partial-VP and Split-NP Topicalization in German - An HPSG Analysis. In: Erhard W. Hinrichs, W. Detmar Meurers, and Tsuneko Nakazawa: *Partial-VP and Split-NP Topicalization in German – An HPSG Analysis and its Implementation.* Arbeitspapiere des SFB 340 Nr. 58, Universität Tübingen.

Höhfeld, M., and G. Smolka. 1988. Definite relatios over constraint languages. LILOG technical report, number 53, IBM Deutschland GmbH.

Höhle, T. N. 1994. Spurenlose Extraktion. Handout of June 1, 1994, Universität Tübingen.

Jaffar, J., and J.-L. Lassez. 1987. Constraint Logic Programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich*, ed. ACM, 111–119.

Johnson, M. 1988. *Attribute-value Logic and the Theory of Grammar.* Vol. 16 of CSLI Lecture Notes. Stanford, CA: CSLI.

Keller, F. 1993. Encoding HPSG Grammars in TFS, Part III - Encoding Revised HPSG. ms., Universität Stuttgart.

King, P. J. 1989. *A logical formalism for head-driven phrase structure grammar*. PhD thesis, University of Manchester.

King, P. J. 1994. An Expanded Logical Formalism for Head-driven Phrase Structure Grammar. Arbeitspapiere des SFB 340, Universität Tübingen.

King, P. J., and T. W. Götz. 1993. Eliminating the feature introduction condition by modifying type inference. Arbeitspapiere des SFB 340 Nr. 31, Universität Tübingen.

Kuhn, J. 1993. Encoding HPSG Grammars in TFS, Part I & II - Tutorial. ms., Universität Stuttgart.

Manandhar, S. 1993. CUF in Context. In *Computational Aspects of Constraint Based Linguistic Descriptions I*, ed. J. Dörre, 43–56. Universität Stuttgart: DYANA-2 Deliverable R1.2.A, August.

Meurers, W. D. 1993. A TFS Implementation of C. Pollard: “On Head Non-Movement”. commented implementation. SFB 340/B4, Universität Tübingen.

Meurers, W. D., and T. Götz. 1993. Using TFS and CUF. Handout of May 10, 1993, SFB 340/B4, Universität Tübingen.

Meurers, W. D., and F. Morawietz. 1993. An Alternative Approach to ID/LP - SUP Grammars. Paper presented at the 1st International Workshop on Head-Driven Phrase Structure Grammar held at Columbus, Ohio.

Morawietz, F. 1994. Direct Parsing of (Typed) Unification ID/LP Grammars. Master's thesis, Seminar für Sprachwissenschaft, Universität Tübingen, Tübingen.

Moshier, M. A., and W. C. Rounds. 1987. A Logic for Partially Specified Data Structures. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages. München, Germany*, 156–167.

Pollard, C. 1990. On Head Non-Movement. In *Proceedings of the Tilburg Conference on Discontinuous Constituents*. also in: Horck, Arthur and Wietske Sijtsma (in press): "Discontinuous Constituency"; Berlin: Mouton de Gruyter.

Pollard, C. 1993. Lexical rules and metadescriptions. Handout of October 5, 1993, Universität Stuttgart.

Pollard, C., and I. Sag. 1987. *Information-Based Syntax and Semantics, Vol. 1: Fundamentals*. CSLI Lecture Notes 13. Stanford, CA: CSLI.

Pollard, C., and I. Sag. 1994. *Head-Driven Phrase Structure Grammar*. Chicago: University of Chicago Press and Stanford: CSLI Publications. (version of Feb 1, 1994).

Reyle, U. 1993. Dealing with Ambiguities by Underspecification: Construction, Representation and Deduction. *Journal of Semantics* 123–179.

Rounds, W. C., and R. T. Kasper. 1986. A Complete Logical Calculus for Record Structures Representing Linguistic Information. In *Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science, Cambridge, MA, USA*, 38–43.

Seiffert, R. 1991. Unification–ID/LP Grammars: Formalization and Parsing. In *Text Understanding in LILOG*, ed. O. Herzog and C.-R. Rollinger, 63–73. Berlin: Springer.

Shieber, S. M. 1984. Direct Parsing of ID/LP Grammars. *Linguistics and Philosophy* 7:135–154.

Shieber, S. M. 1986. *An Introduction to Unification-Based Approaches to Grammar*. CSLI Lecture Notes No. 4. Chicago: Chicago University Press.

Smolka, G. 1988. A Feature Logic with Subsorts. LILOG technical report, number 33, IBM Deutschland GmbH.

Appendix A: The grammar