Revised version of the paper presented at the Conference on Formal Grammar, 12-13 August 1995, Barcelona, Spain.

Towards a Semantics for Lexical Rules as used in HPSG

Walt Detmar Meurers
Sonderforschungsbereich 340 B4/B8, University of Tübingen
Kleine Wilhelmstr. 113, 72074 Tübingen, Germany
E-mail: dm@sfs.nphil.uni-tuebingen.de

Abstract

We show how a mechanism that captures the functionality of lexical rules can be integrated into the logical setup for HPSG provided by King (1989, 1994) without having to enlarge the logic with a metalevel or extra notions such as defaults. A notation for lexical rules is introduced and a rewrite system is defined to map the lexical rules as specified by the linguist into ordinary constraints in the King logic. We therefore show how a denotational semantics for lexical rules as used in HPSG linguistics can be provided.

Keywords: lexical rules, HPSG, logical issues of the HPSG architecture.

1 Introduction

Lexical rules have been used by linguists to express generalizations over the lexicon for over 25 years. In the paradigm of HPSG they have become one of the key mechanisms used in current analysis, be it for cross-serial dependencies and preposition stranding in Dutch, clitic climbing in French and Italian, verb-second, finitivization and partial constituent fronting in German, or passive formation and traceless extraction in general.

Looking at lexical rules from a logical perspective yields a less favorable picture. A formal foundation for HPSG theories has been provided and some studies of various formal and computational aspects exist, but so far no lexical rule mechanism capturing the semantics intended in HPSG linguistics has been formalized.

This paper is intended to help clarify how a lexical rule mechanism could be formalized. More specifically, we show how a mechanism that captures the functionality of HPSG lexical rules can be integrated into the logical setup for HPSG provided by King (1989, 1994). We formalize a notation for lexical rules for use by linguists and define a rewrite system that takes this linguistic lexical rule notation into ordinary descriptions in King's setup. These descriptions can then be interpreted in the usual way. As a result, we obtain a denotational semantics for the linguistic lexical rule notation defined.

The work presented here builds on the discussion of lexical rules in Meurers (1994) and especially the ideas developed in Meurers and Minnen (1995), where a computational treatment of lexical rules is proposed.

The presented research was sponsored by projects B4 and B8 of the Sonderforschungsbereich 340 of the Deutsche Forschungsgemeinschaft. I want to thank Thilo Götz, Erhard Hinrichs, Tilman Höhle, Paul King, Guido Minnen, Carl Pollard, Mike Calcagno, Bill Rounds and the participants of the HPSG Workshop (21–23. June 1995, Tübingen, Germany) and the ACQUILEX II Workshop on Lexical Rules (9–11. August 1995, Cambridge, UK) for valuable comments and discussion. Of course, I am responsible for all remaining errors.

We start out with a discussion of what should be the formal basis of lexical rules. In section 3 we then discuss what lexical rules are supposed to do and which of these effects a linguist wants to write down explicitly and which remain implicit. In section 4 we then analyze what a linguistic lexical rule notation could look like and how the full intention of the linguist can be deduced, i.e., how a fully explicit description can be deduced from the lexical rule notation provided by the linguist. We then turn to the formal definition of a lexical rule notation along the lines discussed: Section 5.1 introduces the formal basis of HPSG we will be using, a version of the logic defined in King (1989). Section 5.2 defines the syntax of the linguistic lexical rule notation. In section 6 we then give a semantics of the linguistic lexical rule notation in terms of a rewrite system which transforms every lexical rule as specified by the linguist into an ordinary description, which can then be interpreted like any other description in King's logic.

2 What should be the formal basis of lexical rules?

In traditional HPSG terms, a lexical rule is a pair of metadescriptions, one for each class of lexical entries related: "lexical rules must be seen as implicative relationships between lexical entries; but lexical entries themselves are constraints on feature structures (not feature structures themselves), so evidently a higher-order formalism must be developed within which such relationships can be expressed." Pollard and Sag (1994, p. 395, fn. 1).

Some questions arise when taking a closer look at the implications of this. From a logical perspective, it seems questionable whether one really wants to extend the logical setup for HPSG by another level, just to formalize lexical rules. There are two other potential candidates for a metalevel formalization: the raising principle (cf., Pollard and Sag (1994, p. 140)) and the lexical template hierarchies (e.g., Pollard and Sag (1987, chapter 8)). However, the raising principle seems to be an instance of a more general principle ensuring full interpretation, and it has been argued that full interpretation should be a consequence of the logical setup, e.g., follow from the use of some kind of resource sensitive logic, not a consequence of explicitly formulated constraints.[1] Regarding the second candidate, it seems to be possible to express the intention behind the lexical template hierarchies by regular type hierarchies and ordinary implicative constraints. Therefore, a kind of metalevel grouping of lexical descriptions is unnecessary. As a result, lexical rules appear to be the only remaining candidate for a metalevel formalization. However, current HPSG theories have a clear logical basis in the logic of King (1989, 1994) and we will show below that a functional equivalent of metalevel lexical rules can in fact be integrated in this setup without altering it.

A question regarding logical and linguistic issues concerns the intended status of a lexical entry serving as input to a lexical rule. The logical property of interest here is that it is undecidable whether a lexical entry which is intended to serve as input to a lexical rule is satisfiable with respect to a grammar, i.e., grammatical. A metalevel lexical rule therefore can derive grammatical entries from ungrammatical lexical entries as well as from grammatical ones.[2] Or expressed differently, in general one cannot make sure that a metalevel lexical rule only applies to lexical entries having a non-empty denotation in the model of a grammar.[3]

[1] Thanks to Erhard Hinrichs, Tilman Höhle and Carl Pollard for a clarifying discussion of this point.

[2] This was pointed out to me by Thilo Götz.

[3] Note that it is possible to specify metalevel lexical rules in such a way that they produce grammatical lexical entries only on the basis of grammatical ones. This is done by specifying the lexical rules to include the complete input in the output, e.g., by introducing an extra attribute in the output of the lexical rule to store the complete input in. However, since one of the motivations for a metalevel formalization of lexical rules is to avoid representing the source of the derivation of a lexical entry inside of the lexical entry, this possibility

To decide whether this is a positive or a negative consequence of a metalevel formalization, the exact consequences of this for linguistic theory will have to be discussed in some future work. In any case, it is a clear difference between a metalevel formalization and a description language formalization of lexical rules which we will pursue in this paper. Under a description language formalization such as the one introduced below, a lexical rule is an ordinary description which is only grammatical if each of its subdescriptions are grammatical. And one of these parts, the in-description, will only be grammatical if the conjunction of the in-description and a lexical entry is grammatical, as we will see in section 2.1. In other words, since a description language lexical rule relates sets of objects in the denotation of a theory, it has no access to ungrammatical objects. Note that this does not mean that under a description language formalization it is impossible to express that some lexical entry cannot be used unless it has undergone some lexical rule. Such 'phantom' lexical entries can be specified in a way which makes them unusable in a syntactic construction, but which does not keep them from being lexical entries which satisfy the grammatical constraints.

Finally, from a linguistic point of view, we believe that it is questionable whether a formalization of lexical rules as metarelations corresponds to the most straightforward interpretation of the lexical rules recently proposed in the HPSG literature. For example, using lexical rules to perform operations on argument raised complements as proposed in many current HPSG theories for German, French, and Italian means that lexical rules treat information which is *not* part of the lexical entry, i.e., the input of a metalevel lexical rule. We therefore believe that it is more natural to think of these lexical rules as relating sets of fully specific *word* objects and not underspecified descriptions of these objects.

Due to these logical and linguistic questions arising under the traditional conception of lexical rules, we think it is worthwhile to see how far we can get with an alternative setup. Even though we agree with Pollard (1993) when he states that "[w]e want to be able to say in the logic: 'every word must either satisfy one of the basic lexical entries, or else it must satisfy $r_i(\varphi)$ [r_i being a lexical rule] for some i and some (not necessarily basic) lexical entry φ."', we disagree when he continues with: "Our representational formalism is too weak to express this, so it must be augmented further." We will show that a mechanism which captures the full functionality of lexical rules as used in HPSG linguistics can be integrated into the logical setup for HPSG provided by King (1994) without having to enlarge the logic with a metalevel or extra notions such as copying or defaults.

In the following, we use the term "lexical rule" to refer to a mechanism such as the one formalized in this paper which provides the functionality which has been assumed for lexical rules in (recent) HPSG theories.

2.1 Lexical rules inside the logic

We see a lexical rule as a binary relation on the set of word objects. In general, there are several options for expressing relations. Either, one encodes the relations inside of the description language, or one expresses the relations by constructs which are part of the relational extension of the description language. Let us take a look at how lexical rule relations can enter the theory under these possibilities to formalize relations.

is not very attractive. Rather, Pollard (p.c.) pursues the idea that one does want to have lexical rules which derive grammatical entries out of ungrammatical input entries.

2.1.1 Lexical rules as description language mechanism

In an HPSG grammar, the lexicon is defined by an implicative constraint[4] on objects of type *word*. The statement in figure 1 is sometimes referred to as the *word principle*. Implicative statements such as the word principle, which are part of the theory and have a type as antecedent, will be called *type definitions* in the following.

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

Figure 1: Defining a basic lexicon

There are several possibilities for introducing lexical rules into this setup of the lexicon. In general any relation can be encoded in the description language as a description of an object with one or more attributes. In HPSG, for example the immediate dominance schemata or the HFP and other principles express relations between a mother and one or more daughters. Copestake (1992), using a feature logic setup based on Carpenter (1992), makes use of this possibility to encode relations and defines a type *lexical-rule* with two attributes 0 and 1 having a type *lex_sign* as appropriate values. Following a computational, rule-based approach, the type *lexical-rule* is defined as a subtype of *rule* and a parser uses lexical and non-lexical rules to build up structure. Since Copestake (1992, section 2.6.4), mostly for efficiency reasons, chooses a rule-based and not a constraint based setup like HPSG, she does not integrate the lexical rules into a word principle like the one shown in figure 1.

To integrate lexical rules into the word principle, an additional attribute (a so-called "junk slot") needs to be introduced on objects of type *word* to house the lexical rule, or (in the alternative encoding described below) the second word related to. This junk slot technique for encoding (possibly recursive) relations on the level of the description language was first used by Aït-Kaci (1984). To formulate a word principle encoding a lexicon including lexical rules, the junk slot technique has for example been used in several implementations of HPSG grammars (Keller (1993), Kuhn (1993), Meurers (1993)).

Figure 2 shows a type definition for *word*, which defines an extended lexicon including lexical rules.

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

Figure 2: An extended lexicon introducing basic entries and entries derived by lexical rules

The type *word* is assumed to have an additional appropriate feature STORE, the junk slot, which is *list*-valued. Furthermore, a new variety *lex_rule* is introduced, having IN and OUT as

[4]We will use the term *descriptions* for the formulas of a formal language like the one of King (1989). The term *specifications* is used for this as well. The denotation of a description is the set of objects described. *Constraint* will be used for those descriptions which are part of the theory (in a formal sense). Only those linguistic objects are grammatical that satisfy every description in the theory, i.e., every constraint. Speaking of terminology, we will use $\top$ as the least constraint type, $\perp$ as the inconsistent type, and the term *varieties* is used to refer to most specific types, a variety is a type not having any subtypes apart from $\perp$. The relevant formal definitions are provided in section 5.1.

appropriate features with *word* values. The relevant part of the signature is shown in figure 3.

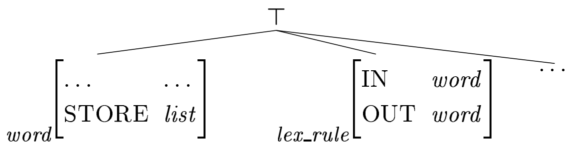

Figure 3: A signature for the modular lexical rule encoding

We will later refer to the setup introduced here as the *modular* lexical rule encoding, since it clearly separates the lexical rule descriptions from the lexical entries and the lexical rule objects from the word objects. The type definition for *lex_rule*, in which the different lexical rules are specified, is given in figure 4.

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

Figure 4: Defining lexical rule objects in the modular encoding

So how does this encoding work? The type definition in figure 2 says that every object of type *word* is described by an ordinary lexical entry[5] L_i $(1 \leq i \leq n)$ or it is the value of the OUT attribute of a *lex_rule* object. To relate the words described by OUT to other words, i.e., to express lexical rule relations, the in- and out-descriptions D_j and E_j $(1 \leq j \leq m)$, abbreviated as LR-IN and LR-OUT, need to be specified. One disjunct on the right-hand side of the implication in figure 4 encodes one lexical rule, which are abbreviated as LR. To state that certain properties of both sets of word objects related are identical, path equalities between LR-IN and LR-OUT are specified. Note that the appropriateness conditions for *lex_rule* ensure that the value of the IN feature is of type *word*. It therefore has to satisfy the type definition for *word*, i.e., one of the lexical entries of figure 2. Naturally the lexical entry satisfied can again be the last disjunct, i.e., the output of a lexical rule.

Note that the description of the last disjunct, i.e., the special entry, in figure 2 is a cyclic structure. Even though this is not necessarily problematic, an alternative encoding of a lexicon with lexical rules avoiding cyclic structures is possible. Two subtypes of *word*, say *simple-word* and *derived-word*, can be introduced with *derived-word* having an additional appropriate attribute IN with *word* value.

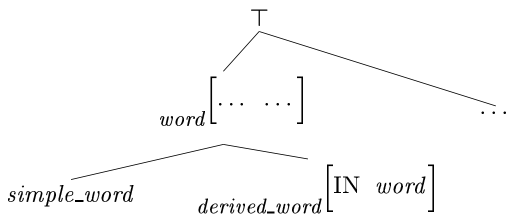

Figure 5: The signature for the word-in-word encoding of LRs

[5]For cleanness sake, in ordinary lexical entries the junk slot STORE should be assigned *e_list* as value.

The following type definitions then define the lexicon including lexical rules:

$$\begin{array}{lcl} \textit{simple-word} & \rightarrow & L_1 \vee \ldots \vee L_n \\ \textit{derived-word} & \rightarrow & \big([\text{IN } D_1] \wedge E_1\big) \vee \ldots \vee \big([\text{IN } D_m] \wedge E_m\big) \end{array}$$

Figure 6: The theory for the word-in-word encoding

In this encoding, the in-description D_j of a lexical rule j ($1 \leq j \leq m$) is specified on the IN-attribute, while the out-description E_j is specified directly on the *derived-word*. This eliminates the structure sharing we needed in the modular lexical rule encoding.

2.1.2 Lexical rules as 'proper' relations

If we chose a formal language for HPSG which allows us to use the relational extension within descriptions, such as the relations defined in Götz (1995) building on King (1989), Höhfeld and Smolka (1988), and Dörre (1994), the proposals for representing a lexicon including lexical rules discussed in section 2.1.1 above can be expressed without having to introduce lexical rules or derived words as types. The lexicon with added lexical rules of figure 2 in a version with lexical rules expressed by 'proper' relations looks as follows.

$$\textit{word} \rightarrow L_1 \vee \ldots \vee L_n \vee \text{lex\_rule}(\textit{word})$$

Figure 7: A lexicon with added lexical rule relations

$$\begin{array}{lcl} \text{lex\_rule}(D_1) & := & E_1. \\ & \vdots & \\ \text{lex\_rule}(D_m) & := & E_m. \end{array}$$

Figure 8: Defining the lexical rule relation

Note that a functional notation of relations is used, since this way ordinary descriptions and relations will have the same denotation, a set of objects. Just as before, D_j is the in-description of lexical rule j and E_j its out-description. What is different in this encoding is that now the lexical rules are defined on a different level than the *word* objects. Which *word* objects satisfy our theory is defined using the description language, while the lex_rule relation is defined using the relational extension of that description language.

2.1.3 Which representation of lexical rule relations should we use?

Having introduced three possibilities for representing lexical rules within the logic, we need to pick one as the basis of our formalization of lexical rules.

Since one of the aims of our work is to show that lexical rules can be formalized in a standard logic for HPSG, we will introduce lexical rules as types, and not as 'proper' relations in the way discussed in the last section. Even though introducing lexical rules in the universe of linguistic objects might be unappealing from a philosophical point of view, there also are some advantages to encoding lexical rules as types. First of all such an encoding allows us to hierarchically organize lexical rules, which makes it possible to express generalizations over groups of lexical rules in a straightforward way. For example, Hinrichs and Nakazawa (1994) say that the Partial-VP Topicalization Lexical Rule, the Split-NP Topicalization Lexical Rule, and the Split-NP Topicalization Lexical Rule 2 share a common mechanism. However, without a possibility to hierarchically group lexical rules and formulate constraints on these groups, they are unable to express the general properties shared by these three rules in any formal way. Apart from such groupings of lexical rules to express common properties of certain subgroups, there also seems to be a clear need to constrain lexical rules in general in a principled way, just like the universal principles of HPSG constrain the rest of the grammar. When lexical rules are introduced as types, constraints on lexical rules can be expressed like any other principle by adding the appropriate implicative description to the theory. Apart from these formal arguments, there also are some interesting linguistic consequences of the fact that under the modular and the word-in-word encoding a word contains its full derivation history, i.e., those words it was derived from by a lexical rule. Just to give an example, one possibility this opens up is to have the binding theory operate on the valence lists of a word before extraction lexical rules have applied. This would eliminate the need of a separate ARG-S list for the sake of the binding theory.[6]

Summing up, it seems to be the case that apart from our original motivation to use the standard logical basis of HPSG, there also are some other interesting aspects of the word-in-word and the modular encoding of lexical rules which make it worthwhile to try and use one of the two encodings in the following. Having decided for a pure description language representation, we still need to choose between the modular and the word-in-word approach. Here a choice is hard to motivate, since both encodings are rather similar and little seems to depend on a choice. Since the modular description language encoding (figures 2, 3, and 4) has the advantage of clearly separating lexical rules and their in- and out-descriptions from words, we will use it, rather than the second in the rest of the paper.

Note that nothing of real importance to the formalization of a lexical rule notation in the next sections depends on the choice made here. The idea behind the lexical rule notation discussed and formalized in the following sections can (presumably) be used for any kind of representation of lexical rules as relations. In fact, even when a metalevel representation of lexical rules is used, some of the discussion of how lexical rules should be noted and how that notation is interpreted should carry over. The reader interested in a metalevel formalization of lexical rules is referred to (Calcagno, 1995) and (Calcagno and Pollard, 1995).

3 Lexical rules and linguistic specifications

Now that we have explored possible ways to formalize lexical rules as ordinary constraints, one could think that we're done. Using the modular lexical rule encoding, the linguist can specify lexical rules as a constraint on *lex_rule* and the theory will do what it's supposed to. However, explicitly having to specify everything a lexical rule is intended to do, does not really correspond to what HPSG linguists are used to. Instead, one usually only writes down those things which really matter for what the lexical rule is intended to do. The rest stays implicit and is to be inferred from what is said. So, even though in the last section we found a way to write down fully explicit lexical rule relations so that they are interpreted as intended, we still have to go one step further: We need to develop a notation in which linguists can write down lexical rules in an elegant way which leaves some things implicit as long as they can be inferred. Once we know what such a linguistic lexical rule notation (LLR-N) should look like, we need to show how lexical rules specified in LLR-N are translated to constraints on *lex_rule*. The rest of the paper serves to explore and formalize these two steps.

[6] This idea is due to Tilman Höhle.

To avoid possible confusion here, let us repeat that we distinguish the language used by the linguist to write down a lexical rule (leaving some things implicit) from the explicit description of what the lexical rule is intended to do. The first will be called linguistic lexical rule notation (LLR-N) and it will be explored and formally defined in the following sections. For the second, the explicit descriptions, the ordinary descriptions of the formal language of King are be used. Lexical rules as specified by the linguist are abbreviated as LLR and they are written as shown in figure 9. The in- and the out-description LLR-In and LLR-Out can be written down in a slightly extended version of AVMs in which we allow two additional symbols that will be introduced below: $\flat$ and $\sharp$. In order to distinguish the augmented language used by linguists to specify LLRs from the ordinary description language, we will call the expressions used in LLRs linguistic descriptions or L-descriptions.

$$\text{LLR-In} \mapsto \text{LLR-Out}$$

Figure 9: A notation for linguistic lexical rules (LLRs)

Fully explicit descriptions of lexical rule objects are abbreviated as LR and they are written in the formal language of King or in an AVM notation as shown in figure 10.[7]

$$_{\textit{lex\_rule}}\begin{bmatrix} \text{IN} & \text{LR-IN} \\ \text{OUT} & \text{LR-OUT} \end{bmatrix}$$

Figure 10: A notation for lexical rules (LRs)

We start with a brief overview of what explicit lexical rule relations are intended to do.

3.1 What needs to be expressed?

Consider two word objects related by a lexical rule. We can distinguish three parts:

a) Certain properties of one object are related to **different** properties of the other object,

b) certain properties of one object are related to **identical** properties in the other object, and

c) certain properties of one object have **no relation** to properties of the other object, either because

   i. the linguist wants certain properties to be unrelated, or because

   ii. certain properties are appropriate for one object but not the other.

For example, a lexical rule relating German base form verbs to their finite forms, among other things needs to

a) relate the base verb form specification and the base morphology to a finite verb form and the corresponding finite morphology,

[7]Note that AVMs are used in in an augmented variant in LLRs and in the standard way in LRs since they are a convenient notation. Nonetheless one should bear in mind that there is a difference between LLRs and LRs. The first is an abbreviation of the second and it does not make sense to interpret LLRs directly as descriptions in King's setup.

b) ensure that the semantic predicate expressed is the same for both objects, and

c) i. ensure that the finite verb can appear in inverted or non-inverted position regardless of the inversion property of the base verb (which in fact can only occur in non-inverted position).

An example for the case c-ii.), where certain properties cannot be transferred, could occur in a nominalization lexical rule which relates verbs to nouns. Since verb form is inappropriate for nouns, that specification cannot be transferred from the verb.

Regarding terminology, following Meurers and Minnen (1995) we will refer to the the task of descriptions ensuring that certain properties are identical (case b) as *property transfer*.

Now we're ready to explore how the three cases a), b), and c) are explicitly or implicitly expressed in a notation which a linguist can use to write down lexical rules (LLR-N).

3.2 What needs to be written down?

When writing down a lexical rule, the linguist only needs two express two of of the three cases (relating differing properties, relating identical properties, unrelated properties). When the linguist specifies those properties which are intended to differ (a) and one more case (b or c), the third kind can be deduced; i.e., it does not have to be expressed explicitly and could be called the "default" specification of lexical rules (in a non-technical sense). So there are two possibilities here: either we have a LLR-N in which those properties which are intended to be identical in the objects described by LR-IN and LR-OUT (case b) are explicitly mentioned. Then non-related properties can remain unexpressed. Or we have a LLR-N in which we explicitly have to mention those properties which are intended to be unrelated in the objects described by LR-IN and LR-OUT (case c-i.). In that case identical properties can remain implicit.

3.2.1 Leaving non-relatedness (case c-i.) unexpressed

At first sight it might seem natural to ask the linguist to express in a lexical rule those specifications which relate properties, i.e., cases a) and b), and keep unexpressed which parts of the objects are unrelated (case c). However, in highly lexicalized theories like HPSG a lexical entry contains many specifications of which only few are relevant in a specific lexical rule. Asking the linguist to explicitly specify that all those specifications without relevance to the lexical rule are identical in the objects related (in case they are appropriate) therefore would amount to asking for a lexical rule with many specifications which are of no direct importance to what the lexical rule is intended to do.

Furthermore, specifying all identities by hand sometimes can only be achieved by splitting up a lexical rule into several instances.[8] Let us illustrate this with an example based on the signature given in the appendix of Pollard and Sag (1994). Assume, for expository purpose only, a lexical rule deriving predicative signs from non-predicative ones.

$$\left[\text{SYNSEM}|\text{LOCAL}|\text{CAT}|\text{HEAD}|\text{PRD}\ \ \text{-}\right] \mapsto \left[\text{SYNSEM}|\text{LOCAL}|\text{CAT}|\text{HEAD}|\text{PRD}\ \ +\right]$$

Figure 11: PRD-LLR (for exposition only)

A linguist having to specify property transfer by hand would be forced to write down five different rules, one for each subtype of the *substantive* heads (those which can have a PRD

[8] Cf., Meurers (1994, section 4.1.3) and Meurers and Minnen (1995).

attribute). There are two reasons why one needs to split up the rule for each subtype if one explicitly specifies property transfer. First, the subtype of *substantive* needs to be preserved, but no path equality can be specified since the attribute PRD which gets changed is below that type. Second, the subtypes have different appropriate attributes for which we need to make sure that they are transferred. Figure 12 shows one of the five lexical rules a linguist would have to write to explicitly write down property transfer, the one for verbs.

$$
\begin{bmatrix}
\text{PHON} & \boxed{1} \\
\text{SYNSEM} & \begin{bmatrix} \text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & {}_{verb}\begin{bmatrix} \text{PRD} & - \\ \text{VFORM} & \boxed{2} \\ \text{AUX} & \boxed{3} \\ \text{INV} & \boxed{4} \end{bmatrix} \\ \text{SUBCAT} & \boxed{5} \\ \text{MARKING} & \boxed{6} \end{bmatrix} \\ \text{CONT} & \boxed{7} \\ \text{CONX} & \boxed{8} \end{bmatrix} \\ \text{NONLOCAL} & \boxed{9} \end{bmatrix} \\
\text{QSTORE} & \boxed{10} \\
\text{RETRIEVED} & \boxed{11}
\end{bmatrix} \mapsto
$$

$$
\begin{bmatrix}
\text{PHON} & \boxed{1} \\
\text{SYNSEM} & \begin{bmatrix} \text{LOCAL} & \begin{bmatrix} \text{CAT} & \begin{bmatrix} \text{HEAD} & {}_{verb}\begin{bmatrix} \text{PRD} & + \\ \text{VFORM} & \boxed{2} \\ \text{AUX} & \boxed{3} \\ \text{INV} & \boxed{4} \end{bmatrix} \\ \text{SUBCAT} & \boxed{5} \\ \text{MARKING} & \boxed{6} \end{bmatrix} \\ \text{CONT} & \boxed{7} \\ \text{CONX} & \boxed{8} \end{bmatrix} \\ \text{NONLOCAL} & \boxed{9} \end{bmatrix} \\
\text{QSTORE} & \boxed{10} \\
\text{RETRIEVED} & \boxed{11}
\end{bmatrix}
$$

Figure 12: PRD-LLR with property transfer for verbs explicitly specified

It is clear that we do not want to ask the linguist to write down several instances of lexical rules in this way. But let us briefly note that the above example points out an important property of LLRs which will be very useful when we explain how LLRs are transformed into explicit LRs: it suffices to look at a LLR and the signature to determine which kind of objects could undergo this lexical rule in order to tell which attributes need to be transferred. No reference to lexical entries was necessary to make property transfer in the above example explicit. We only need lexical entries when an explicit lexical rule is applied to make sure that the objects satisfying the input of a lexical rule also satisfy a basic lexical entry or the output of a lexical rule. As we saw in section 2.1, figure 2, the word principle is extended to take care of this.

3.2.2 Leaving identity (case b) unexpressed

The notation for lexical rules used in most HPSG theories proposed in the literature assumes that the properties which are not explicitly changed by the lexical rule, are identical for both sets of objects related. In other words, property transfer (case b) is left unspecified.

The advantages of allowing the linguist to leave identities unexpressed follow from the disadvantages discussed in section 3.2.1 above: Firstly, the linguist only needs to express those things of direct relevance to the lexical rule. Secondly, leaving identities unexpressed saves the linguist from having to split up a lexical rule into several instances. And finally, since we are striving to provide a semantics for lexical rules as used in HPSG linguistics, as long as it is formally reasonable, we naturally try to get as close as possible to the notation assumed in the HPSG theories proposed in the literature.

Nothing was said so far about how the linguist expresses case c) and in fact this case is usually not mentioned in the HPSG literature. However, if both case b) and c) are left unspecified, it is not possible to uniquely determine whether a property of one object is supposed to be identical to that of the other object or whether that property of both objects is unrelated. To distinguish between the two cases we will need to introduce some additional notation for case c).

3.2.3 A linguistic lexical rule notation

In the last two sections we discussed two possible setups for a LLR-N and we decided that we do not want to express property transfer explicitly in the LLR-N. So now the task is to fix a syntax for the other two cases.

The case a) in which the linguist specifies which properties are changed is the easiest case. Here everything is already explicit and we can use the ordinary descriptions of the formal language of King. Case c-ii.) doesn't need any additional notation since it only amounts to saying that no inappropriate (in the technical sense) property transfer should be deduced.

So far nothing apart from ordinary descriptions was required. In fact, one could try to also express case c-i.), the intentional non-correspondence, by using certain ordinary descriptions, namely denotationally irrelevant ones. To specify that the type value of an attribute in the out-description is intended to be unrelated to the in-description, one could specify a type as value of that attributes which is less specific than the type which is appropriate for that attribute. However, since the appropriate value of an attribute depends on the type of object this attribute is on, and the intended interaction of types in the in- and out-description is already rather complex (cf., section 4.1), it seems clearer to introduce a separate notation for type unrelatedness here.

In a description language using variables instead of path-equalities, there also is a possibility to express that two attributes are not required to be reentrant (called *reentrancy elimination* in the following) using a denotationally irrelevant description. By using a variable as value of an LLR-Out attribute which is different from the variable that is the value of its corresponding LLR-In attribute, one can express that the LLR-Out attribute is not intended to be reentrant with its corresponding LLR-In attribute and the nodes it is reentrant with.[9] In case the LLR-Out attribute is no longer reentrant with any other node, the new variable will only have a single occurrence. However, proceeding in this way has the consequence that LLRs can no longer be mapped to LRs just by looking at the signature. Instead one would have to consider which kind of variables are present in the lexical entries that license word objects

[9] I believe this possibility was mentioned by Mike Calcagno in a talk on 23. June 1994 at the HPSG Workshop in Tübingen.

which can undergo a lexical rule. Since otherwise we are able to map LLRs into explicit LRs just by looking at the LLRs and the signature, having to look at other parts of the grammar to tell what the abbreviating notation means seems to be an unnecessary complication.

After this excursion into the possibilities to express case c-ii.) without adding additional LLR notation, we now show how extra LLR syntax can be introduced to do the job. Note that we do not extend the description language at the formal basis of HPSG, i.e., the King logic. We only add additional notation to the linguistic lexical rule notation which will be used by the procedure which maps LLRs into LRs. The mapping will also eliminate the extra notation so that it does not have to be interpreted in LRs.

Type elimination is noted by a $\flat$ as value of a type assignment. E.g., $\begin{bmatrix}\mathrm{X|U} & \flat\end{bmatrix}$ is used to eliminate the value of attribute U.

Regarding path equality elimination, one might be tempted to introduce $\sharp$ as a binary operator which is used just like path equality $\approx$. The notation with different subscripts, i.e., $\sharp_i$, could then be used if multiple pairs of path equalities are to be eliminated. However, there is a problem with such a notation. Take the LLR given in 13.

$$\begin{bmatrix} \mathrm{W} & \boxed{1} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \mapsto \begin{bmatrix} \mathrm{W} & \sharp_2 \\ \mathrm{X} & \sharp_2 \end{bmatrix}$$

Figure 13: A LLR with the binary $\sharp$ notation

Take the set of objects described by LLR-In.[10] LLR-Out is intended to say that there is a second set of objects, which is just like the first set but we don't want to force attributes W and X to be token identical for the second set of objects. However, because of the transitivity of path equality, having to eliminate the path equality restriction on W and X also requires us to alter the relationships of W and X to Y and Z. So it is unclear whether the LR which is abbreviated by the LLR in 13 is supposed to be the description in 14 a, b, c, or d.

$$\text{a.} \begin{bmatrix} \mathrm{IN} & \begin{bmatrix} \mathrm{W} & \boxed{1} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \\ \mathrm{OUT} & \begin{bmatrix} \mathrm{W} & \boxed{1} \\ \mathrm{X} & \boxed{2} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{2} \end{bmatrix} \end{bmatrix} \quad \text{b.} \begin{bmatrix} \mathrm{IN} & \begin{bmatrix} \mathrm{W} & \boxed{1} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \\ \mathrm{OUT} & \begin{bmatrix} \mathrm{W} & \boxed{2} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{2} \end{bmatrix} \end{bmatrix} \quad \text{c.} \begin{bmatrix} \mathrm{IN} & \begin{bmatrix} \mathrm{W} & \boxed{1} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \\ \mathrm{OUT} & \begin{bmatrix} \mathrm{W} & \boxed{1} \\ \mathrm{X} & \boxed{2} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \end{bmatrix} \quad \text{d.} \begin{bmatrix} \mathrm{IN} & \begin{bmatrix} \mathrm{W} & \boxed{1} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \\ \mathrm{OUT} & \begin{bmatrix} \mathrm{W} & \boxed{2} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & \boxed{1} \\ \mathrm{Z} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Figure 14: The four possibilities to map the LLR of figure 13 into a LR

To obtain a unique interpretation of the binary $\sharp$ notation one would need to complicate the $\sharp$ notation further or specify additional rules for its interpretation. Instead of complicating

[10]Note that the objects in the denotation of LLR-In in the modular lexical rule encoding also have to satisfy a lexical entry or the output of a lexical rule. The token identity requirement on the objects therefore can stem from either of these places. Since we want to rewrite a LLR as LRs independent of the lexical entries, we therefore cannot look at which path equalities hold between attributes in the input and do something on the basis of that. Instead, we use structure sharing between attributes in the in- and the out-specification which is possible since the signature tells us what attributes can be present. This will be discussed in section 4.1.3.

matters in this way, we introduce path equality elimination $\sharp$ as eliminating the path equality requirements with one path only. Figure 15 shows the AVM notation for eliminating the path equality requirements on attribute X.

$$\left[\text{X} \;\; \sharp\right]$$

Figure 15: Path equality elimination in AVM notation

In a sense, the unary $\sharp$ notation is less expressive that the binary one. If one wants to specify an LLR-Out for the LLR-In of figure 13 so that the LLR-Out is rewritten as either one of the four specific cases of figure 14, the path equalities which are intended to be kept for an attribute for which $\sharp$ was used to eliminate some path equalities need to be restated. It is necessary to repeat certain path equalities because it is only possible to eliminate all path equalities with an attribute and not only those holding between two attributes leaving the others as in the binary notation. Still, the notation seems to be straightforward and until there is evidence that a more expressive notation is needed we will stick to it.

$$\text{a.} \begin{bmatrix} \text{X} & \sharp\boxed{2} \\ \text{Z} & \boxed{2} \end{bmatrix} \qquad \text{b.} \begin{bmatrix} \text{X} & \sharp\boxed{1} \\ \text{Y} & \boxed{1} \end{bmatrix} \qquad \text{c.} \left[\text{X} \;\; \sharp\right] \qquad \text{d.} \left[\text{W} \;\; \sharp\right]$$

Figure 16: LLR-Outs which are used to arrive at the cases in figure 14 with a unary $\sharp$

4 Exploring a linguistic lexical rule notation

Now that we have an idea of what we need in a LLR-N, in the following we need to explore how the L-descriptions making up the LLR-N can be used in a way that makes it possible to deduce the intended explicit lexical rule descriptions.

The idea behind our formalization of a linguistic rule notation is to define a rewrite system that takes every LLR-N into fully specified lexical rule descriptions which can be given the standard set theoretical interpretation defined by King (1989). What exactly has to be done by this rewrite system? The mapping from LLRs to LRs has to fill in descriptions ensuring identity of unmentioned properties (*property transfer*) and this has to be done with respect to the extra notation added to the LLR-N to express unrelatedness (case c-i). We will call this *enriching* an LLR. The result of enriching an LLR is a LR. In terms of the description language encoding we use, enriching adds variety assignments to LR-OUT and path equalities between paths in the in-description and the corresponding paths in the out-description.

As became clear when we discussed the different setups for a LLR-N in the last section, enriching is done solely on the basis of the LLRs and the signature; it is not dependent on the lexical entries. Since LLRs are interpreted by enriching them into ordinary LR descriptions and interpreting these in the ordinary set theoretical way this allows us to assign a semantics to LLRs without looking at the lexical entries in a grammar. What makes this possible is the closed world interpretation of type hierarchies assumed in HPSG and King (1989). The signature and the closed world assumption tell us which kind of *word* objects could possibly exist (regardless of whether they are grammatical, i.e., in the denotation of the theory, or not). We already saw the closed world assumption at work in figure 12 of section 3.2.2 when we discussed what a linguist has to do if (s)he wants to specify property transfer by hand. In what follows, we use the same logical properties of HPSG to automatically enrich a description with property transfer.

4.1 Exploring the various possibilities

There are two basic types of specifications: variety assignment ($\sim$) and path equality ($\approx$). Since we are dealing with two descriptions, one as part of the LLR-Out, one as part of LLR-In, the relative position of two specifications needs to be considered. The path leading to the specification in LLR-Out can be included in that leading to a specification in LLR-In. The two specifications can be at the same place in LLR-In and LLR-Out. The path leading to the specification in LLR-Out can be an extension of that leading to a specification in LLR-In. To deal with the L-description syntax we allowed for LLR-Out, we additionally have to consider certain out-descriptions: the empty specification, the type elimination specification ($\flat$), and the path equality elimination specification ($\sharp$).

As basis of the examples discussed below, assume the following example signature:

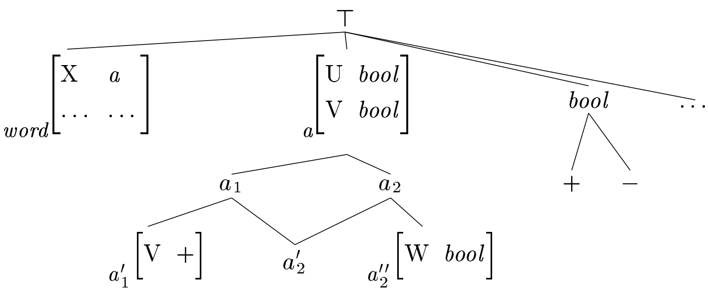

Figure 17: An example signature

4.1.1 What's the type when type meets type?

Consider the linguistic lexical rules shown in figure 18.

$$\begin{array}{lll} 1. & word & \mapsto \begin{bmatrix} \text{X} & a_2 \end{bmatrix} \\ 2. & word & \mapsto \begin{bmatrix} \text{X} & a_2' \end{bmatrix} \end{array}$$

Figure 18: LLRs with type specifications

We know by the signature that every word object that serves as input to one of these lexical rules has to have an X attribute that has some variety of a as its value. So what kind of X value do we want for the word objects that are related to the first set by one of the lexical rules in figure 18? One thing is clear: we want the objects licensed by the output of the lexical rule to be in the denotation of $\begin{bmatrix} \text{X} & a_2 \end{bmatrix}$ for the first and $\begin{bmatrix} \text{X} & a_2' \end{bmatrix}$ for the second lexical rule. What needs to be decided in addition to that is whether the set of objects is further restricted to have the value of X identical to the value of X in an object which serves as input to the lexical rule. Say, for example a word object whose X is a_2' serves as input to the first lexical rule. Are there only objects in the denotation of the output of the lexical rule that have a_2' as value of X in that case, or are there also some that have a_2'' as value of X (which would be in the denotation of $\begin{bmatrix} \text{X} & a_2 \end{bmatrix}$). Generally speaking, in the first case a type on an object undergoing a lexical rule is preserved if it is compatible with the out-specification. In the second case, the out-specification is all that matters. Say we pursue the first possibility and do preserve the type. Then, in cases where preservation of a type is not intended, adding a single type elimination specification to LLR-Out does the trick. So this is ok. On the other hand, if we do not preserve the type, having to specify transfer of the type specification by hand can force the linguist to explicitly specify many subcases of the LLR, as was discussed in section 3.2.1. As a result, a LLR-N designed to preserve a type if possible seems to be the most convenient.

LLR-N design decision 1 For every path P with value t in LLR-Out which is also defined on the objects in the denotation of LLR-In do the following in case there is no specification $\begin{bmatrix}\text{P} & \flat\end{bmatrix}$: specify the LRs so that each object having a subtype of t as value of P is related to objects having the same variety as P's value. Every other object (the value of P is not described by t) is related to objects having any subtype of t as value of P.

The effect of this design decision is that every LLR is enriched into several LRs: we obtain one LR for every variety assignment possible according to the signature and the specification in LLR-Out.

Applying the design decision 1 to the LLRs in figure 18, we obtain the following. The cases in which a variety is preserved are marked with a $\bullet$. In the other cases, the variety considered in LR-IN and the type in LLR-Out is incompatible and the specification in LLR-Out is used for LR-OUT.

$$
\begin{array}{llll}
1. & \text{a)} & \begin{bmatrix}\text{IN} & \begin{bmatrix}\text{X} & a_2'\end{bmatrix}\\ \text{OUT} & \begin{bmatrix}\text{X} & a_2'\end{bmatrix}\end{bmatrix} & \bullet \\
 & \text{b)} & \begin{bmatrix}\text{IN} & \begin{bmatrix}\text{X} & a_2''\end{bmatrix}\\ \text{OUT} & \begin{bmatrix}\text{X} & a_2''\end{bmatrix}\end{bmatrix} & \bullet \\
 & \text{c)} & \begin{bmatrix}\text{IN} & \begin{bmatrix}\text{X} & a_1'\end{bmatrix}\\ \text{OUT} & \begin{bmatrix}\text{X} & a_2\end{bmatrix}\end{bmatrix} &
\end{array}
\qquad
\begin{array}{llll}
2. & \text{a)} & \begin{bmatrix}\text{IN} & \begin{bmatrix}\text{X} & a_2'\end{bmatrix}\\ \text{OUT} & \begin{bmatrix}\text{X} & a_2'\end{bmatrix}\end{bmatrix} & \bullet \\
 & \text{b)} & \begin{bmatrix}\text{IN} & \begin{bmatrix}\text{X} & a_2''\end{bmatrix}\\ \text{OUT} & \begin{bmatrix}\text{X} & a_2'\end{bmatrix}\end{bmatrix} & \\
 & \text{c)} & \begin{bmatrix}\text{IN} & \begin{bmatrix}\text{X} & a_1'\end{bmatrix}\\ \text{OUT} & \begin{bmatrix}\text{X} & a_2'\end{bmatrix}\end{bmatrix} &
\end{array}
$$

Figure 19: LRs resulting from enriching the LLRs of figure 18

So far, so good; but what about the following cases:

$$
\begin{array}{lll}
1. & \mathit{word} \mapsto & \begin{bmatrix}\text{X|W} & +\end{bmatrix} \\
2. & \mathit{word} \mapsto & \begin{bmatrix}\text{X|V} & -\end{bmatrix} \\
3. & \mathit{word} \mapsto & \begin{bmatrix}\text{X} & \begin{bmatrix}\text{U} & \boxed{1}- \\ \text{V} & \boxed{1}\end{bmatrix}\end{bmatrix}
\end{array}
$$

Figure 20: LLRs with implicit type specifications

The problem is that in the examples in figure 20, even though no type is specified directly in LLR-Out for X, still only certain types as values for X in LLR-Out will yield a consistent description. In the first LLR in figure 20, the attribute W is only appropriate for objects of type a_2'', in the second LLR the attribute V with value "$-$" is not appropriate for objects of type a_1', and in the third LLR V is structure shared with U and, since U has "$-$" as value, again a_1' is not a possible value for X.

The solution to this problem is to first infer the types on all nodes in LLR-Out which are forced by the descriptions in LLR-Out and the signature. Then the ordinary design decision 1 can be used. Luckily, the task of inferring a variety as value of each attribute in a description has already been dealt with: The normalization algorithm of Götz (1993) and Kepser (1994) can be used to transform a description into a normal form representation in which (among other things) every attribute is assigned a variety consistent with the rest of the description (in case this is possible). The formal definition of the normal form is provided in section 5.3.

4.1.2 Paths not mentioned in the out-specification

Having dealt with type specifications, we now look at attributes which do not occur in LLR-Out even though they could be there according to the signature. In case such an attribute is also appropriate for LLR-In, we can transfer the value of the attribute in LLR-In to the corresponding attribute in LLR-Out. In our setup this is done by specifying a path equality between the path in LLR-In and the corresponding one in LLR-Out. Having such a path equality do the property transfer makes sure that all constraints on the path which have to be satisfied by an object serving as input to the lexical rule – some of which stem from LR-IN, others that aren't present in the lexical rule but stem from a lexical entry or output of another lexical rule – are present on the corresponding path in LR-OUT.

Two other things should be explained here: First, we know that if there is a path equality between two paths this means that both paths point to the same object. Therefore we also know that the value of an appropriate attribute A extending one of the paths in the path equality and the value of the same attribute A extending the other path in the path equality will also be token identical. As a result, it suffices to specify structure sharing at certain attributes as close as possible to the root, namely those attributes which do not occur in LLR-Out themselves, but directly extend paths or subpaths that are specified in LLR-Out. Note that since the number of specifications in LLR-Out is finite and the number of appropriate attributes of an object is finite, a finite number of path equalities between LLR-In and LLR-Out will suffice to specify property transfer of unmentioned attributes. Second, as we saw in section 3.2.1, one needs to watch out for subtypes of LLR-Out attribute values which have additional appropriate attributes. But since in the formal setup of King (1989) introduced in section 5.1 a type assignment is only an abbreviation for a disjunction of variety assignments and since we will deduce property transfer on the basis of a disjunctive normal form, all cases including those introducing additional appropriate attributes are explicit and nothing special needs to be done. The following design decision does the trick.

LLR-N design decision 2 Take any path P that occurs in LLR-Out. For each appropriate attribute A s.t. PA does not occur in LLR-Out, LR contains a path equality between PA in LR-OUT and PA in LR-IN in case A is appropriate for the corresponding path P in LR-IN.

Since different varieties as value of A can have different appropriate attributes, the above design decision tells us to map every LLR into a set of LRs just like it was the case in the type–type case dealt with in design decision 1. The resulting set of LRs is interpreted disjunctively. Each LR has a specific variety as value of A and in that rule the attributes appropriate for that variety are dealt with. We already saw an example for the effect of this design decision in section 3.2.1. Decision 2 states that figure 11 on p. 9 is interpreted as if it had the structure sharings defined in 12 on p. 10 specified. So by interpreting LLRs in the way fixed in design decisions 2 and 1, we have indeed *automated* what in section 3.2.1 we argued that we do not want to ask the linguist to do *by hand*.

4.1.3 The case of path equalities

After having decided what we do with unspecified attributes and type specifications, we now turn to path equalities. The first idea one might have is to look at the path equalities in

LLR-In and transfer them to the corresponding paths in LLR-Out. For example, if paths X and Y are structure shared in LLR-In then we include a structure sharing between X and Y in LR-OUT, unless one of the paths in LLR-Out carries a path equality elimination specification ♯. However, such a straightforward copying of path equalities can be problematic: Höhle (1995) shows that preserving path equalities in such a way leads to wrong results when using the Complement Extraction Lexical Rule of Pollard and Sag (1994) to extract complements from verbs bearing a Hinrichs/Nakazawa style argument raising specification. The reason has to do with the first-rest encoding of lists and the fact that a path equality cause two list positions to be structure shared, not two list elements. As a result, if one element of a list is removed in LLR-Out, the path equalities of LLR-In would have to be adjusted before copying them over to LLR-Out, to reflect the new position of the elements which were structure shared before. While such adjusting of path equalities already is a complicated, in the general case perhaps impossible task, there even is one more complication. Since consistency is assumed as application criterion, one not only has to take care of the path equalities in LLR-In but also the path equalities which occur in the lexical entries the lexical rule applies to. Note that it is not possible to determine a finite set of path equalities just by looking at the signature and the lexical rule, as was the case when we had to find out above what variety specifications or paths might occur in some kind of lexical entry. So if we pursue this idea of copying over (adjusted) path equalities, we can no longer interpret lexical rules independent of the lexical entries. I.e., in such a case it is not possible to transform the linguistic lexical rules into an explicit, interpretable form just by looking at the linguistic lexical rule and the signature.

Luckily, there is an alternative. Instead of taking path equalities form the input and enforcing corresponding path equalities in the output, we specify path equalities between LLR-In and LLR-Out. For example[11], when LLR-In contains the path equality (IN|X = IN|Y), we do not add the path equality (OUT|X = OUT|Y); instead we add (IN|X = OUT|X) and (IN|Y = OUT|Y). The path equality (OUT|X = OUT|Y) then does not have to be explicitly stated since it follows by transitivity of the path equality relation. Under such a setup, we need to watch out for one thing though: We introduce path equalities between paths in LR-IN and LR-OUT. So any specification of a path structure shared with in LR-OUT will also restrict the corresponding path structure shared with in LR-IN. By specifying path equalities for transfer in such a case, we therefore restrict the domain of objects which can undergo the lexical rule; an unwanted effect. We will therefore only transfer path equalities for those paths, that do not get specified in LLR-Out. Since this is just what we did in design decision 2, there's nothing else to do here.

LLR-N design decision 3 The transfer of path equalities is ensured by design decision 2.

Before we finish this section, one might think that we still need to cover several other cases. In the very beginning of this section we said that the relative position of two specifications needs to be considered; but so far we seem not to have addressed specifications which are not on the same attribute in LLR-In and LLR-Out. So how does a path equality or type specification on an attribute in LLR-In interact with a path equality or type specification on a different attribute in LLR-Out when it comes to enriching? In fact, this question was already answered above, but let us clarify the issue some further. Regarding the treatment of types, design decision 1 only treat types as values of the same attribute in LLR-In and LLR-Out; but the trick is that we normalize the L-descriptions first. After normalizing a description, we know for every path which varieties are possible values that are consistent with the rest of the

[11]We use a somewhat informal notation here since the formal notation will be introduced in the following section 5.1.

description. Therefore design decision 1 does all there is to do. Regarding the treatment of path equalities, design decision 2 specifies path equalities between LR-IN and LR-OUT for paths which are not specified in LR-OUT. The structure sharing between in- and out-description 'automatically' lets the attribute in LLR-Out take part in whatever restrictions are put on objects described by LR-IN, e.g., when the LR is applied. So we don't need to worry about descriptions in LR-IN. And since we only specify such transfer structure sharing for paths which are not part of LLR-Out, we do not need to worry about other specifications in LLR-Out either.

Now that we've cast our intuitions into the design of a language to be used by linguists to specify lexical rules (LLR-N), we need to formalize these intuitions.

5 The formal languages

5.1 The description language of King (1989)

As the formal basis of our approach we assume the logical setup of King (1989). As shown in King (1994), this setup provides the foundation desired for HPSG in Pollard and Sag (1994). The formal language defined in the following is a version of the one proposed by King.

5.1.1 Syntax

Definition 1 (Signature) A **signature** $\mathcal{S}$ is a triple $\langle \mathcal{V}, \mathcal{A}, \text{approp} \rangle$ s.t.

- $\mathcal{A}$ is a finite set of **attribute** names
- $\mathcal{V}$ is a finite set of **varieties** (also called species or minimal types)[12]
- approp : $\mathcal{V} \times \mathcal{A} \rightarrow Pow(\mathcal{V})$ is a total function from pairs of varieties and attribute names to sets of varieties

Everything which follows is done with respect to a signature. For notational convenience we will work with an implicit signature $\langle \mathcal{V}, \mathcal{A}, \text{approp} \rangle$. This is possible since at no point in our proposal do we have to alter the signature.

Definition 2 (Term) Let $:$ be a reserved symbol, the root symbol of a path. A **term** is a member of the smallest set $\mathcal{T}$ s.t.

- $: \in \mathcal{T}$ and
- $\tau\alpha \in \mathcal{T}$ if $\tau \in \mathcal{T}$ and $\alpha \in \mathcal{A}$

Definition 3 (Description) Let $(,), \sim, \approx, \neg, \wedge, \vee$ and $\rightarrow$ be reserved symbols. A **description** is a member of the smallest set $\mathcal{D}$ s.t.

- $\tau \sim \varphi \in \mathcal{D}$ if $\tau \in \mathcal{T}$ and $\varphi \in \mathcal{V}$
- $\tau_1 \approx \tau_2 \in \mathcal{D}$ if $\tau_1, \tau_2 \in \mathcal{T}$
- $\neg\delta \in \mathcal{D}$ if $\delta \in \mathcal{D}$

[12]For easier comparison with standard HPSG notation, one can introduce a finite join semi-lattice $\langle \mathcal{Z}, \preceq \rangle$ as type hierarchy with $\mathcal{Z} \supset \mathcal{V}$. A type assignment is then simply an abbreviation for a set of variety assignments: $\tau \sim t = \bigvee\{\tau \sim \varphi \mid \varphi \preceq t\}$ with $\tau \in \mathcal{T}$, $t \in \mathcal{Z}$, and $\varphi \in \mathcal{V}$. In the paper, we assume every type assignment to be expanded in that way. Nothing of theoretical importance hinges on this.

- $(\delta_1 \wedge \delta_2), (\delta_1 \vee \delta_2), (\delta_1 \rightarrow \delta_2) \in \mathcal{D} \qquad \text{if } \delta_1, \delta_2 \in \mathcal{D}$

Definition 4 (Theory) A **theory** Θ is a subset of $\mathcal{D}$ ($\Theta \subseteq \mathcal{D}$).

Definition 5 (Set of Literals) A **set of literals** Σ is a subset of the set of descriptions $\mathcal{D}$ ($\Sigma \subset \mathcal{D}$), s.t. each $\delta \in \Sigma$ has one of the four forms ($\tau, \tau_1, \tau_2 \in \mathcal{T}; \varphi \in \mathcal{V}$):

- $\tau \sim \varphi$
- $\tau_1 \approx \tau_2$
- $\neg\, \tau \sim \varphi$
- $\neg\, \tau_1 \approx \tau_2$

5.1.2 Semantics

Definition 6 (Interpretation of a Signature) An **interpretation** I is a triple $\langle \mathrm{U}, \mathrm{V}, \mathrm{A} \rangle$ s.t.

- U is a **set of objects**, the domain of I,
- $\mathrm{V} : \mathrm{U} \rightarrow \mathcal{V}$ is a total function from the set of objects to the set of varieties, the **variety assignment** function,
- $\mathrm{A} : \mathcal{A} \rightarrow \{\mathrm{U} \rightharpoonup \mathrm{U}\}$[13] is an **attribute interpretation** function s.t. for each $u \in \mathrm{U}$ and $\alpha \in \mathcal{A}$:
  - if $\mathrm{A}(\alpha)(u)$ is defined then $\mathrm{V}(\mathrm{A}(\alpha)(u)) \in \mathrm{approp}(\mathrm{V}(u), \alpha)$, and
  - if $\mathrm{approp}(\mathrm{V}(u), \alpha) \neq \emptyset$ then $\mathrm{A}(\alpha)(u)$ is defined.

Definition 7 (Interpretation of Terms) $[]^I : \mathcal{T} \rightarrow \{\mathrm{U} \rightharpoonup \mathrm{U}\}$ is a **term interpretation** function over interpretation $\mathrm{I} = \langle \mathrm{U}, \mathrm{V}, \mathrm{A} \rangle$ s.t.

- $[:]^I$ is the identity function on U, and
- $[\tau\alpha]^I$ is the functional composition of $[\tau]^I$ and $\mathrm{A}(\alpha)$.

Definition 8 (Interpretation of Descriptions) $[\![]\!]^I : \mathcal{D} \rightarrow \mathit{Pow}(\mathrm{U})$ is a **description interpretation** function over interpretation $\mathrm{I} = \langle \mathrm{U}, \mathrm{V}, \mathrm{A} \rangle$ s.t. ($\tau, \tau_1, \tau_2 \in \mathcal{T}; \varphi \in \mathcal{V}; \delta, \delta_1, \delta_2 \in \mathcal{D}$):

- $[\![\tau \sim \varphi]\!]^I = \{u \in \mathrm{U} \mid [\tau]^I(u) \text{ is defined and } \mathrm{V}([\tau]^I(u)) = \varphi\}$,
- $[\![\tau_1 \approx \tau_2]\!]^I = \left\{ u \in \mathrm{U} \;\middle|\; \begin{array}{l} [\tau_1]^I(u) \text{ is defined,} \\ [\tau_2]^I(u) \text{ is defined, and} \\ [\tau_1]^I(u) = [\tau_2]^I(u) \end{array} \right\}$,
- $[\![\neg\delta]\!]^I = \mathrm{U} \setminus [\![\delta]\!]^I$
- $[\![(\delta_1 \wedge \delta_2)]\!]^I = [\![\delta_1]\!]^I \cap [\![\delta_2]\!]^I$
- $[\![(\delta_1 \vee \delta_2)]\!]^I = [\![\delta_1]\!]^I \cup [\![\delta_2]\!]^I$
- $[\![(\delta_1 \rightarrow \delta_2)]\!]^I = (\mathrm{U} \setminus [\![\delta_1]\!]^I) \cup [\![\delta_2]\!]^I$

[13] We write $\{X \rightharpoonup Y\}$ for the set of partial functions from set X to set Y.

Definition 9 (Interpretation of a Theory) A theory is interpreted conjunctively. $[\![ ]\!]^I : Pow(\mathcal{D}) \rightarrow Pow(\mathrm{U})$ is a **theory interpretation** function over interpretation $\mathrm{I} = \langle \mathrm{U}, \mathrm{V}, \mathrm{A} \rangle$ s.t.

$$[\![\Theta]\!]^I = \bigcap\{[\![\delta]\!]^I \mid \delta \in \mathcal{D}\}$$

Definition 10 (Satisfiability) A theory Θ is **satisfiable** iff there is an interpretation I s.t.

- $[\![\Theta]\!]^I \neq \emptyset$

Definition 11 (Model) An interpretation $\mathrm{I} = \langle \mathrm{U}, \mathrm{V}, \mathrm{A} \rangle$ is a **model** of a theory Θ if $[\![\Theta]\!]^I = \mathrm{U}$

The definitions above define a class of formal languages which can be used to express HPSG grammars. We only list these definitions here to make it possible to follow the formal definition and interpretation of the linguistic lexical rule notation in the next sections. A discussion of the formal language of King is beyond the scope of this paper. The reader interested in such a discussion is referred to King (1994).

5.2 The syntax of the linguistic lexical rule notation

Definition 12 (Lexical Rule Signature) Every signature $\mathcal{S}$ for which the following condition holds is a lexical rule signature $\mathcal{S}_{lr}$.

- $lex\_rule \in \mathcal{V}$ and
- $\mathrm{IN}, \mathrm{OUT} \in \mathcal{A}$ and
- $\mathrm{approp}(lex\_rule, \mathrm{IN}) = \{word\}$ and
- $\mathrm{approp}(lex\_rule, \mathrm{OUT}) = \{word\}$.

Definition 13 (L-Description) Let $\flat$ and $\sharp$ be reserved symbols. With respect to a lexical rule signature $\mathcal{S}_{lr}$ let $\mathcal{T}$ be a set of terms, $\mathcal{D}$ a set of descriptions. A L-description is a member of the smallest set $\mathcal{L}$ s.t.

- $d \in \mathcal{L}$ if $d \in \mathcal{D}$ and
- $:\mathrm{OUT}\mu \sim \flat \in \mathcal{L}$ if $\mu \in \mathcal{A}^+$ and
- $:\mathrm{OUT}\mu\sharp \in \mathcal{L}$ if $\mu \in \mathcal{A}^+$.

Definition 14 (Linguistic Lexical Rule) With respect to a given lexical rule signature $\mathcal{S}_{lr}$ a linguistic lexical rule LLR is a subset of the set of L-descriptions $\mathcal{L}$ containing at least the following literals ($\varphi \in \mathcal{V}; \mu, \mu_1, \mu_2 \in \mathcal{A}^+$):

- $: \sim lex\_rule$ and
- $:\mathrm{OUT}\mu \sim \varphi$ or $:\mathrm{OUT}\mu_1 \approx :\mathrm{OUT}\mu_2$ or $:\mathrm{OUT}\mu \sim \flat$ or $:\mathrm{OUT}\mu\sharp$.

There's nothing complicated going on here. We just add the additional LLR notation by defining L-descriptions with respect to a lexical rule signature. A linguistic lexical rule then consists of L-descriptions, and for convenience sake we ask for an LLR-out containing at least one specification.

In most HPSG theories proposed in the literature, AVMs are used as descriptions instead of the term notation introduced above. AVMs can be seen as a kind of normal form representation for descriptions. We already introduced a notation for linguistic lexical rules in figure 9 on p. 8. Now that we've introduced the formal notation for linguistic lexical rules, let us illustrate the different ways in which one can write down LLRs with an example (which is not intended to say much but just show the way things are written down). We will use the notation shown in figure 21 on the left as shorthand for the AVM shown on the right, which in the formal notation defined above is expressed as shown below that.[14]

$$\begin{bmatrix} \mathrm{B} & \boxed{1} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & {}_{u}\begin{bmatrix} \mathrm{Z} & \boxed{1} \end{bmatrix} \end{bmatrix} \mapsto \begin{bmatrix} \mathrm{A} & \flat \\ \mathrm{B} & \sharp \\ \mathrm{U|V} & \flat\boxed{2} \\ \mathrm{X} & \sharp\boxed{2} \end{bmatrix} \qquad {}_{lex\_rule}\begin{bmatrix} \textsc{in} & \begin{bmatrix} \mathrm{B} & \boxed{1} \\ \mathrm{X} & \boxed{1} \\ \mathrm{Y} & {}_{u}\begin{bmatrix} \mathrm{Z} & \boxed{1} \end{bmatrix} \end{bmatrix} \\ \textsc{out} & \begin{bmatrix} \mathrm{A} & \flat \\ \mathrm{B} & \sharp \\ \mathrm{U|V} & \flat\boxed{2} \\ \mathrm{X} & \sharp\boxed{2} \end{bmatrix} \end{bmatrix}$$

$$: \sim lex\_rule \ \wedge \ :\textsc{in}\ \mathrm{B} \approx :\textsc{in}\ \mathrm{X} \ \wedge \ :\textsc{in}\ \mathrm{X} \approx :\textsc{in}\ \mathrm{Y\ Z} \ \wedge \ :\textsc{in}\ \mathrm{Y} \sim u \ \wedge$$
$$:\textsc{out}\ \mathrm{A} \sim \flat \ \wedge \ :\textsc{out}\ \mathrm{B}\sharp \ \wedge \ :\textsc{out}\ \mathrm{U\ V} \sim \flat \ \wedge \ :\textsc{out}\ \mathrm{U\ V} \approx :\textsc{out}\ \mathrm{X} \ \wedge \ :\textsc{out}\ \mathrm{X}\sharp$$

Figure 21: Three ways to write down LLRs

5.3 A normal form for L-descriptions

In section 4.1 we saw that the L-descriptions making up the LLR need to be normalized to have a consistent variety assigned to each defined attribute, which is needed for the mapping from LLRs to LRs. This section serves to introduce a normal form for descriptions. It reports work carried out in Götz (1993) and Kepser (1994). Originally, the normalization algorithm is used to determine if a given description is satisfiable.

The linguist writes down LLRs. So we want to normalize normalize L-descriptions, not simple descriptions. Since L-descriptions are a simple extension of descriptions with two additional statements for type and path equality elimination, we only need to add two trivial clauses to the description normalization algorithm of Götz (1993) to obtain an algorithm which transforms an L-description into normal form.

First, we need to introduce some additional terminology.

Definition 15 (Terms and subterms in Σ) The set $\textsc{Term}(\Sigma)$ contains all paths occurring in a set of literals Σ and their subpaths ($\tau \in \mathcal{T}; \pi \in \mathcal{A}^*; \varphi \in \mathcal{V}$):

$$\begin{aligned} \textsc{Term}(\Sigma) \ = \ & \{\tau \in \mathcal{T} \mid (\neg)\ \tau\pi \approx \tau' \in \Sigma\} \\ & \cup \{\tau \in \mathcal{T} \mid (\neg)\ \tau' \approx \tau\pi \in \Sigma\} \\ & \cup \{\tau \in \mathcal{T} \mid (\neg)\ \tau\pi \sim \varphi \in \Sigma\} \end{aligned}$$

[14]Since the path equality relation is transitive, there are several possibilities to encode the example in the formal notation. Normalization (cf., section 5.3) introduces all path equalities which can be inferred due to transitivity.

Definition 16 (Clause and Matrix)

- A **clause** Σ is a finite (possibly empty) set of literals.
- A **matrix** Γ is a finite (possibly empty) set of clauses.

Definition 17 (Interpretation of a Clause and a Matrix)

- A **clause** is interpreted conjunctively. If Σ is a clause, then $[\![\Sigma]\!]^I = \bigcap_{\delta \in \Sigma} [\![\delta]\!]^I$.
- A **matrix** is interpreted disjunctively. If Γ is a matrix, then $[\![\Gamma]\!]^I = \bigcup_{\Sigma \in \Gamma} [\![\Gamma]\!]^I$.

The conversion from L-descriptions to its normal form proceeds in two steps. First, the L-description is transformed into disjunctive normal form, i.e., where all negations are pulled in and the disjuncts are on the top level. The resulting matrix Γ is a finite set, each element of which represents one disjunct. Each disjunct is a clause which consists of a finite set of literals. Since the transformation into disjunctive normal form is a rather standard procedure, we simply assume its existence here. Second, the resulting matrix is normalized. We start with a declarative characterization of what it means for an L-description to be in normal form.

Definition 18 (Normal Clause) A set Σ of literals is **normal** iff the following conditions hold ($\tau, \tau_1, \tau_2 \in \textsc{Term}(\Sigma); \varphi, \varphi_1, \varphi_2 \in \mathcal{V}; \alpha \in \mathcal{A}; \pi \in \mathcal{A}^*$)

Original:

1. $: \approx : \in \Sigma$ (root is defined)
2. if $\tau_1 \approx \tau_2 \in \Sigma$ then $\tau_2 \approx \tau_1 \in \Sigma$; (symmetry of $\approx$)
3. if $\tau_1 \approx \tau_2, \tau_2 \approx \tau_3 \in \Sigma$ then $\tau_1 \approx \tau_3 \in \Sigma$; (transitivity)
4. if $\tau\pi \approx \tau\pi \in \Sigma$ then $\tau \approx \tau \in \Sigma$; (prefix closure)
5. if $\tau_1 \approx \tau_2, \tau_1\pi \approx \tau_1\pi, \tau_2\pi \approx \tau_2\pi \in \Sigma$ then $\tau_1\pi \approx \tau_2\pi \in \Sigma$; ($\approx$ and path extensions)
6. if $\tau \approx \tau \in \Sigma$ then for some $\varphi \in \mathcal{V}, \tau \sim \varphi \in \Sigma$; (exhaustive typing)
7. if for some $\varphi \in \mathcal{V}, \tau \sim \varphi \in \Sigma$ then $\tau \approx \tau \in \Sigma$; ($\sim$ path is defined)
8. if $\tau_1 \approx \tau_2 \in \Sigma, \tau_1 \sim \varphi_1 \in \Sigma, \tau_2 \sim \varphi_2 \in \Sigma$ then $\varphi_1 = \varphi_2$; ($\approx$ and $\sim$)
9. if $\tau \sim \varphi_1 \in \Sigma, \tau\alpha \sim \varphi_2 \in \Sigma$ then $\varphi_2 \in \text{approp}(\varphi_1, \alpha)$; (appropriateness 1)
10. if $\tau \sim \varphi \in \Sigma, \tau\alpha \in \textsc{Term}(\Sigma), \text{approp}(\varphi, \alpha) \neq \emptyset$ then $\tau\alpha \approx \tau\alpha \in \Sigma$; (appropriateness 2)
11. if $\neg\delta \in \Sigma$ then $\delta \notin \Sigma$. (no contradictions)

Additional:

12. if $\tau \sim \flat \in \Sigma$ then $\tau \approx \tau \in \Sigma$; ($\sim \flat$ path is defined)
13. if $\tau\sharp \in \Sigma$ then $\tau \approx \tau \in \Sigma$; ($\sharp$ path is defined)
14. if $:\textsc{out}\pi\alpha \approx :\textsc{out}\pi\alpha, :\textsc{in}\pi \sim \varphi \in \Sigma, \mathit{approp}(\varphi, \alpha) \neq \emptyset$ then $:\textsc{in}\pi\alpha \approx :\textsc{in}\pi\alpha \in \Sigma$; (corresponding in-paths are defined)

The algorithm which takes an L-descriptions as a DNF matrix and returns its normal form is given below as a set of rewrite rules on sets of clauses. Γ is used as variable over sets of clauses and Σ as variable over clauses. Readers interested in the formal properties of the algorithm and a discussion of the normal form are referred to Kepser (1994, section II).

Algorithm 1 (Clause Normalization) The algorithm consists of a sequence rewrite rule applications. One step of the algorithm is the application of exactly one rewrite rule. The algorithm terminates, if no rule can be applied (any more). A rule applies to a set of clauses Γ' only if the left hand side of the rule matches Γ', and if the right hand side is a valid set description under the same variable assignment. The rewrite rules are ($\varphi_1, \varphi_2 \in \mathcal{V}; \alpha \in \mathcal{A}; \pi \in \mathcal{A}^*$) :

$$
\begin{array}{rcll}
\Gamma \uplus \{\Sigma\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{:\approx:\}\} & (1)\\
\Gamma \uplus \{\Sigma \uplus \{\tau_1 \approx \tau_2\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau_1 \approx \tau_2\} \uplus \{\tau_2 \approx \tau_1\}\} & (2)\\
\Gamma \uplus \{\Sigma \uplus \{\tau_1 \approx \tau_2, \tau_2 \approx \tau_3\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau_1 \approx \tau_2, \tau_2 \approx \tau_3\} \uplus \{\tau_1 \approx \tau_3\}\} & (3)\\
\Gamma \uplus \{\Sigma \uplus \{\tau\sigma \approx \tau\sigma\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau\sigma \approx \tau\sigma, \tau \approx \tau\}\} & (4)\\
\Gamma \uplus \{\Sigma \uplus \{\tau_1 \approx \tau_2, \tau_1\sigma \approx \tau_1\sigma, & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau_1 \approx \tau_2, \tau_1\sigma \approx \tau_1\sigma, & (5)\\
\tau_2\sigma \approx \tau_2\sigma\}\} & & \quad \tau_2\sigma \approx \tau_2\sigma\} \uplus \{\tau_1\sigma \approx \tau_2\sigma\}\} & \\
\Gamma \uplus \{\Sigma \uplus \{\tau \approx \tau\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau \approx \tau, \tau \sim \varphi\} \mid \varphi \in \mathcal{V}\}, & (6)\\
 & & \text{if } \forall\varphi'.\ \tau \sim \varphi' \notin \Sigma & \\
\Gamma \uplus \{\Sigma \uplus \{\tau \sim \varphi\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau \sim \varphi, \tau \approx \tau\}\} & (7)\\
\Gamma \uplus \{\Sigma \uplus \{\tau_1 \approx \tau_2, \tau_1 \sim \varphi_1, & \longrightarrow & \Gamma, \text{ if } \varphi_1 \neq \varphi_2 & (8)\\
\tau_2 \sim \varphi_2\}\} & & & \\
\Gamma \uplus \{\Sigma \uplus \{\tau \sim \varphi_1, \tau\alpha \sim \varphi_2\}\} & \longrightarrow & \Gamma, \text{ if } \varphi_2 \notin \text{approp}(\varphi_1, \alpha) & (9)\\
\Gamma \uplus \{\Sigma \uplus \{\tau \sim \varphi\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau \sim \varphi, \tau\alpha \approx \tau\alpha\}\}, & (10)\\
 & & \text{if } \tau\alpha \in \textsc{Term}(\Sigma) \text{ and approp}(\varphi, \alpha) \neq \emptyset & \\
\Gamma \uplus \{\Sigma \uplus \{\delta, \neg\delta\}\} & \longrightarrow & \Gamma, \text{ for any positive literal } \delta & (11)\\
\Gamma \uplus \{\Sigma \uplus \{\tau \sim \flat\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau \sim \flat, \tau \approx \tau\}\} & (12)\\
\Gamma \uplus \{\Sigma \uplus \{\tau\sharp\}\} & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{\tau\sharp, \tau \approx \tau\}\} & (13)\\
\Gamma \uplus \{\Sigma \uplus \{:\textsc{out}\pi\alpha \approx :\textsc{out}\pi\alpha, & \longrightarrow & \Gamma \cup \{\Sigma \uplus \{:\textsc{out}\pi\alpha \approx :\textsc{out}\pi\alpha, & (14)\\
:\textsc{in}\pi \sim \varphi\}\} & & \quad :\textsc{in}\pi \sim \varphi, & \\
 & & \quad :\textsc{in}\pi\alpha \approx :\textsc{in}\pi\alpha\}\}, \text{ if } \mathit{approp}(\varphi, \alpha) \neq \emptyset &
\end{array}
$$

Each rewrite rule corresponds to a line in the definition of a normal clause. Line 3 of definition 18, for example, demands transitivity of path equality. The corresponding rewrite rule (3) in algorithm 1 picks out a clause with two literals expressing path equalities and adds a literal expressing the path equality resulting from transitivity, if it is not already part of the clause. Note the use of ordinary ($\cup$) and disjoint union ($\uplus$). The last occurrence of disjoint union in the rewrite rule (3) ensures that this rule will only apply, if the literal to be added was not part of the original clause, i.e., if transitivity for the two literals did not already hold in Σ.

The original normalization algorithm of Götz (1993) consists of rules 1–11. Since we are dealing with L-descriptions, in addition we have to take care of the new symbols $\flat$, and $\sharp$. For each such literal, rule (12) adds a literal defining the corresponding path. The rest of the algorithm will then ensure that each subpath is also defined and that each (sub)path is assigned the possible varieties. Finally, rule (13) ensures that for each path in the out-description the corresponding path in the in-description is introduced, if it is appropriate.

6 A Semantics for the linguistic lexical rule notation

We define an algorithm which realizes a function from lexical rules as specified by the linguist (LLR) to enriched descriptions of lexical rule objects which can be given the standard set theoretical interpretation defined in section 5.1.[15] The conversion from LLR to ordinary descriptions proceeds in two steps. First, the LLR is converted into normal form, then the normal form LLR is enriched with additional path equalities and variety assignments to encode the property transfer which is only implicit in the LLR. As a result of enriching the LLR we obtain an ordinary description, i.e., a LR, which is interpreted in the normal way.

6.1 Enriching a LLR matrix: Intro

We saw in section 5.3 what it means for a L-description to be in normal form. Now we turn to the enriching algorithm. The input to the enriching algorithm is a LLR in normal form. A normalized LLR is a matrix Γ^{llr}, a finite set, each element of which represents one disjunct. Each disjunct is a clause which consists of a finite set of literals.

Take a normalized LLR and apply the following steps:

1. For every clause Σ in the LLR matrix, define a new matrix $\Gamma = \{\Sigma\}$.
2. With each such Γ obtain an enriched matrix Γ^e by applying a sequence of rewrite rules with respect to Σ until no rules can be applied.

The transfer enriched LLR matrix Γ^{lr} is the union of all transfer enriched matrices Γ^e obtained, from which all inconsistent clauses and all literals of the form $\tau\sharp$ and $\tau \sim \flat$ have been eliminated. ($\tau \in \mathcal{T}$)

6.2 An enriching algorithm

Algorithm 2 (Transfer enriching a LLR matrix) A rule applies to a matrix Γ with respect to Σ iff the matrix matches the left hand side of the rule and the right hand side is a valid set description under the same variable assignment. ($\varphi_1, \varphi_2 \in \mathcal{V}; \alpha \in \mathcal{A}; \pi \in \mathcal{A}^*$)

The rewrite rules are:

$$\begin{array}{ll} \Gamma' & \uplus \left\{\Sigma' \cup \left\{\begin{array}{l} :\mathrm{IN}\pi \sim \varphi_1, \\ :\mathrm{OUT}\pi \sim \varphi_2 \end{array}\right\}\right\} \\ & \uplus \left\{\Sigma'' \cup \left\{\begin{array}{l} :\mathrm{IN}\pi \sim \varphi_1, \\ :\mathrm{OUT}\pi \sim \varphi_1 \end{array}\right\}\right\} \end{array} \underset{\Sigma}{\longrightarrow} \Gamma' \cup \left\{\Sigma'' \cup \left\{\begin{array}{l} :\mathrm{IN}\pi \sim \varphi_1, \\ :\mathrm{OUT}\pi \sim \varphi_1 \end{array}\right\}\right\} \tag{1}$$

$$if\, \varphi_1 \neq \varphi_2 \quad \text{and } :\mathrm{OUT}\pi \sim \flat \notin \Sigma$$

$$\Gamma' \uplus \left\{\Sigma' \uplus \left\{\begin{array}{l} :\mathrm{OUT}\pi \sim \varphi_1, \\ :\mathrm{IN}\pi \sim \varphi_2 \end{array}\right\}\right\} \underset{\Sigma}{\longrightarrow} \Gamma' \cup \tag{2}$$

$$\left\{\Sigma' \uplus \left\{\begin{array}{l} :\mathrm{OUT}\pi \sim \varphi_1, \\ :\mathrm{IN}\pi \sim \varphi_2, \\ :\mathrm{IN}\pi\alpha \approx :\mathrm{OUT}\pi\alpha \end{array}\right. \;\middle|\; \begin{array}{l} \mathrm{approp}(\varphi_1, \alpha) \cap \\ \mathrm{approp}(\varphi_2, \alpha) \neq \emptyset \end{array}\right\}\right\}$$

$$if \quad :\mathrm{OUT}\pi\alpha \approx :\mathrm{OUT}\pi\alpha \notin \Sigma$$

[15] The formal setup used in this section is modeled after the description normalization algorithm of Götz (1993) which is reported in Kepser (1994).

The above rewrite rules are a formal version of the design decisions 1 and 2. In prose, the rewrite rules do the following:

1. Check if the type on a certain in-path is compatible with that on the corresponding out-path. I.e., check if certain varieties are assigned to both the in and the out-path. If that's the case, eliminate the disjunct in which the in-path and the out-path are not assigned the same variety.
2. Introduce structure sharing between LLR-In and LLR-Out for all attributes α extending a path which is defined in LLR-Out in case α is appropriate for both the path in LLR-In and the corresponding one in LLR-Out and the path extended by α is not itself defined in LLR-Out. Note that the last conjunct includes the case that the path extended by α is specified with a path or type elimination marker.

6.3 An example

Now that we're done exploring and formalizing the LLR-N, let us end this paper with a linguistic example taken from Pollard and Sag (1994, chapter 9, p. 378), the Complement Extraction Lexical Rule (CELR). There are two reasons for looking at this example. On the one hand the signature is explicitly given by Pollard and Sag. This is necessary to understand what goes on with a linguistic lexical rule. On the other hand, the CELR is rather difficult to express without a formalized lexical rule mechanism and can cause unwanted results under some interpretations as discussed by Höhle (1995). So this makes it a good test case to see whether we've made things any clearer, even though a lot of the possibilities which we envisaged in the design of the LLR-N will naturally play no role in this particular case.

Since the different ways to encode relations in HPSG are a separate issue and we don't want to complicate the example with a junk slot encoding of append and union, we use a simplified version of the CELR. The particular instance of the CELR we discuss treats the second element on COMPS and SUBCAT for entries with an empty SLASH set instead of treating any element on COMPS and any on SUBCAT with any SLASH set. The rest of the rule we use is like in the original CELR. Figure 22 shows how in our setup the CELR can be specified by the linguist. Note that only those parts which are intended to be changed need to be mentioned at all. No type or path equality elimination is needed for this example.

$$\begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{LOC|CAT|VAL|COMPS|R} & \langle \boxed{3}[\text{LOC } \boxed{1}] \mid \boxed{5} \rangle \\ \text{NLOC|INHER|SLASH} & \{\} \end{bmatrix} \\ \text{SUBCAT|R|F } \boxed{3} & \end{bmatrix} \mapsto \begin{bmatrix} \text{SYNSEM} & \begin{bmatrix} \text{LOC|CAT|VAL|COMPS|R} & \boxed{5} \\ \text{NLOC|INHER|SLASH} & \{\boxed{1}\} \end{bmatrix} \\ \text{SUBCAT|R|F|NLOC|INHER|SLASH} & \{\boxed{1}\} \end{bmatrix}$$

Figure 22: The Complement Extraction Lexical Rule (simplified)[16]

Since no typing information is specified in LLR-Out and those attributes which have types as values that have subtypes (HEAD, NUCLEUS, RESTIND, DTRS, ...) are not mentioned in LLR-Out, all the work to map the CELR into a description is done by the rewrite rule that adds path equalities between the in- and the out-description. The LR resulting from enriching the CELR is shown in figure 23.

[16] Where tags are needed, the same tag numbers as in the original formulation of Pollard and Sag are used. F and R are used as abbreviated notation for the *ne_list* attributes FIRST and REST.

$$
\begin{bmatrix}
\textit{lex\_rule} & \\
\underline{\text{IN}} & \begin{bmatrix}
\textit{word} & \\
\text{PHON} & \boxed{@1} \\
\underline{\text{SYNSEM}} & \begin{bmatrix}
\textit{synsem} & \\
\underline{\text{LOC}} & \begin{bmatrix}
\textit{loc} & \\
\underline{\text{CAT}} & \begin{bmatrix}
\textit{cat} & \\
\text{HEAD} & \boxed{@2} \\
\underline{\text{VAL}} & \begin{bmatrix}
\textit{val} & \\
\underline{\text{COMPS}} & \begin{bmatrix}
\textit{ne\_list} & \\
\text{F} & \boxed{@3} \\
\underline{\text{R}} & \begin{bmatrix}
\textit{ne\_list} & \\
\underline{\text{F}} & \boxed{3}\begin{bmatrix} \textit{synsem} & \\ \underline{\text{LOC}} & \boxed{1} \end{bmatrix} \\
\underline{\text{R}} & \boxed{5}
\end{bmatrix}
\end{bmatrix} \\
\text{SUBJ} & \boxed{@4} \\
\text{SPR} & \boxed{@5}
\end{bmatrix} \\
\text{MARK} & \boxed{@6}
\end{bmatrix} \\
\text{CONT} & \boxed{@7} \\
\text{CONX} & \boxed{@8}
\end{bmatrix} \\
\underline{\text{NLOC}} & \begin{bmatrix}
\underline{\text{INHER}} & \begin{bmatrix}
\textit{nloc1} & \\
\underline{\text{SLASH}} & \{\} \\
\text{REL} & \boxed{@9} \\
\text{QUE} & \boxed{@10}
\end{bmatrix} \\
\text{TO-B} & \boxed{@11}
\end{bmatrix}
\end{bmatrix} \\
\underline{\text{SUBCAT}} & \begin{bmatrix}
\textit{ne\_list} & \\
\text{F} & \boxed{@12} \\
\underline{\text{R}} & \begin{bmatrix}
\textit{ne\_list} & \\
\underline{\text{F}} & \boxed{3}\begin{bmatrix}
\text{LOC} & \boxed{@13} \\
\text{NLOC} & \begin{bmatrix}
\text{INHER} & \begin{bmatrix} \text{REL} & \boxed{@14} \\ \text{QUE} & \boxed{@15} \end{bmatrix} \\
\text{TO-B} & \boxed{@16}
\end{bmatrix}
\end{bmatrix} \\
\text{R} & \boxed{@17}
\end{bmatrix}
\end{bmatrix} \\
\text{QSTORE} & \boxed{@18} \\
\text{RETR} & \boxed{@19}
\end{bmatrix} \\
\underline{\text{OUT}} & \begin{bmatrix}
\textit{word} & \\
\text{PHON} & \boxed{@1} \\
\underline{\text{SYNSEM}} & \begin{bmatrix}
\textit{synsem} & \\
\underline{\text{LOC}} & \begin{bmatrix}
\textit{loc} & \\
\underline{\text{CAT}} & \begin{bmatrix}
\textit{cat} & \\
\text{HEAD} & \boxed{@2} \\
\underline{\text{VAL}} & \begin{bmatrix}
\textit{val} & \\
\underline{\text{COMPS}} & \begin{bmatrix}
\textit{ne\_list} & \\
\text{F} & \boxed{@3} \\
\underline{\text{R}} & \boxed{5}
\end{bmatrix} \\
\text{SUBJ} & \boxed{@4} \\
\text{SPR} & \boxed{@5}
\end{bmatrix} \\
\text{MARK} & \boxed{@6}
\end{bmatrix} \\
\text{CONT} & \boxed{@7} \\
\text{CONX} & \boxed{@8}
\end{bmatrix} \\
\underline{\text{NLOC}} & \begin{bmatrix}
\textit{nloc} & \\
\underline{\text{INHER}} & \begin{bmatrix}
\textit{nloc1} & \\
\underline{\text{SLASH}} & \{\boxed{1}\} \\
\text{REL} & \boxed{@9} \\
\text{QUE} & \boxed{@10}
\end{bmatrix} \\
\text{TO-B} & \boxed{@11}
\end{bmatrix}
\end{bmatrix} \\
\underline{\text{SUBCAT}} & \begin{bmatrix}
\textit{ne\_list} & \\
\text{F} & \boxed{@12} \\
\underline{\text{R}} & \begin{bmatrix}
\textit{ne\_list} & \\
\underline{\text{F}} & \begin{bmatrix}
\textit{synsem} & \\
\text{LOC} & \boxed{@13} \\
\underline{\text{NLOC}} & \begin{bmatrix}
\textit{nloc} & \\
\underline{\text{INHER}} & \begin{bmatrix} \underline{\text{SLASH}} & \{\boxed{1}\} \\ \text{REL} & \boxed{@14} \\ \text{QUE} & \boxed{@15} \end{bmatrix} \\
\text{TO-B} & \boxed{@16}
\end{bmatrix}
\end{bmatrix} \\
\text{R} & \boxed{@17}
\end{bmatrix}
\end{bmatrix} \\
\text{QSTORE} & \boxed{@18} \\
\text{RETR} & \boxed{@19}
\end{bmatrix}
\end{bmatrix}
$$

Figure 23: The explicit LR resulting from enriching the CELR of figure 22

To make it easier to read, the path equalities which were added by the enriching algorithm bear an @ sign before the number in order to distinguish them from the tags present in the linguistic lexical rule. Also, the attributes that were part of the original linguistic specification are underlined. For each of these defined paths normalization introduced a specification $:\tau \approx :\tau$ to mark which paths are defined and variety specifications $:\tau \sim \varphi$. So, the varieties along the underlined paths in figure 23 are introduced by normalization. The path equalities marked with an @ are introduced by enriching: Along the defined paths the appropriate attributes are introduced and path equalities between paths in the in-specification and the out-specification are added for those attributes which directly extend a defined path but are not themselves defined, i.e., underlined.

Note that the element on the SUBCAT list of the output which corresponds to the one that is extracted from the COMPS list turns out to be identical to the element on the input's SUBCAT list except for the NLOC|INHER|SLASH specification. This is just what was intended but what in the absence of a formalized lexical rule mechanism was not not formally expressed in the original formulation of the CELR as discussed in footnote 37 on p. 378 of (Pollard and Sag, 1994)).

Regarding the problem of Höhle (1995), we had already mentioned in section 4.1.3 that this problem is caused by having to modify certain path equalities before 'copying them over' from the in-specification to the out-specification. Since we do no such copying, but rather specify path equalities between the in- and the out-specification this problem does not arise in our approach.

Finally, to take a place in the theory, the description in figure 23 is included as one of the disjuncts on the right-hand side of the type definition of type *lex_rule* which we saw in figure 4 on p. 5.

7 Concluding Remarks

In this paper, we've tried to shed some light on a possibility to formalize lexical rules using a standard logical basis of HPSG. First, we defined lexical rules so that they can be constrained by ordinary descriptions. Then we explored and defined a linguistic lexical rule notation which allowed us to leave certain things implicit. Finally, we showed how we can get from the linguistic notation to the explicit constraints.

We believe there are some nice properties of such an approach: First of all, apart from the mapping from the linguistic notation to explicit constraints, we did not add any additional machinery to the logic. The semantics of the linguistic lexical rules after the mapping is provided by the standard King logic which is used for all of HPSG. The advantage this has for the linguist is that when it comes down to seeing exactly what a certain linguistic lexical rule means, (s)he can always take a look at the resulting enriched, fully explicit descriptions of lexical rules in the language used to write the rest of the HPSG theory, instead of having to interpret the linguistic lexical rules directly in some kind of additional formal system.

Second, the mapping from linguistic lexical rules to explicit constraints is done independent of the lexical entries. It suffices to look at the linguistic lexical rules and the signature to determine what remained implicit in the linguistic lexical rule notation and how it can be made explicit. This is possible because HPSG is built on a type feature logic and a closed word interpretation of a type hierarchy.

Third, the approach presented is highly modular and adaptable to the linguist's needs: One can decide on the data structure for lexical rules one likes best (relations or ordinary descriptions), alter/extend the linguistic lexical rule notation in a way one likes, and alter/extend the rewrite rules which enrich linguistic lexical rules to ordinary descriptions in a way one likes. This is important until a real discussion of possibilities and linguistic consequences of various setups has shown what HPSG linguist's really want to write down and what it's supposed to mean.

And fourth, if one takes descriptions of lexical rule objects as basic encoding as we have done in the main part of the paper (and not 'proper' relations as part of the relational extension as considered in the beginning), this makes it possible to hierarchically group lexical rules and express constraints on (groups of) lexical rules. This allows us to express general principles every lexical rule has to obey, and it makes it possible to express that a group of lexical rules shares certain properties.

Finally, as mentioned in the introduction, this paper builds on ideas developed in Meurers and Minnen (1995) and can be seen as providing a formal foundation for that computational proposal. The computational treatment proposed in Meurers and Minnen (1995) can be used for lexical rules producing an infinite lexicon since it avoids expanding out the lexicon under lexical rule application. The compiler at the heart of the computational treatment transforms a set of lexical rules into a set of relational constraints on lexical entries by considering the possible interaction of the set of lexical rules and the lexicon in an abstract way. The encoding is then advanced by program transformation techniques to allow on the fly application of the relations encoding the lexical rules.

References

Aït-Kaci, Hassan. 1984. *A lattice theoretic approach to computation based on a calculus of partially ordered type structures.* Ph.D. thesis, University of Pennsylvania.

Calcagno, Mike. 1995. Interpreting lexical rules. In *Proceedings of the Conference on Formal Grammar*, Barcelona. Also in: Proceedings of the ACQUILEX II Workshop on Lexical Rules, 1995, Cambridge, UK.

Calcagno, Mike and Carl Pollard. 1995. Lexical rules in HPSG: What are they? In preparation for the Acquilex II Workshop on Lexical Rules.

Carpenter, Bob. 1992. *The Logic of Typed Feature Structures - With Applications to Unification Grammars, Logic Programs and Constraint Resolution*, volume 32 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, New York.

Copestake, Ann. 1992. *The Representation of Lexical Semantic Information.* Cognitive science research paper CSRP 280, University of Sussex.

Dörre, Jochen. 1994. Feature-Logik und Semiunifikation. Arbeitspapiere des SFB 340 Nr. 48, Universität Stuttgart.

Götz, Thilo. 1993. A normal form algorithm for king's descriptive formalisms. unpublished term paper, Universität Tübingen.

Götz, Thilo. 1995. Compiling HPSG constraint grammars into logic programs. Draft, extended version of a paper presented at CLNLP 95, Edinburgh, Scotland.

Hinrichs, Erhard and Tsuneko Nakazawa. 1994. Partial-VP and Split-NP topicalization in German: An HPSG analysis. In (Hinrichs, Meurers, and Nakazawa, 1994).

Hinrichs, Erhard W., W. Detmar Meurers, and Tsuneko Nakazawa. 1994. 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 Gert Smolka. 1988. Definite relations over constraint languages. LILOG technical report, number 53, IBM Deutschland GmbH.

Höhle, Tilman N. 1995. The complement extraction lexical rule and variable argument raising. Handout for a talk given at the HPSG Workshop held in Tübingen on 21–23. June 1995.

Keller, Frank. 1993. Encoding HPSG grammars in TFS, part III - encoding revised HPSG. ms., Universität Stuttgart.

Kepser, Stephan. 1994. A satisfiability algorithm for a logic for typed feature structures. Master's thesis, Universität Tübingen.

King, Paul. 1989. *A Logical Formalism for Head-Driven Phrase Structure Grammar.* Ph.D. thesis, University of Manchester.

King, Paul. 1994. An expanded logical formalism for Head-Driven Phrase Structure Grammar. Arbeitspapiere des SFB 340 59, University of Tübingen.

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

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

Meurers, W. Detmar. 1994. On implementing an HPSG theory – Aspects of the logical architecture, the formalization, and the implementation of head-driven phrase structure grammars. In: Erhard W. Hinrichs, 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.

Meurers, W. Detmar and Guido Minnen. 1995. A computational treatment of HPSG lexical rules as covariation in lexical entries. In *Proceedings of the Fifth International Workshop on Natural Language Understanding and Logic Programming*, Lisbon, Portugal.

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

Pollard, Carl and Ivan A. Sag. 1987. *Information-based Syntax and Semantics, Vol. 1.* Number 13 in Lecture Notes. CSLI Publications, Stanford University. Distributed by University of Chicago Press.

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