UNIVERSITÄT BIELEFELD
FAKULTÄT FÜR LINGUISTIK UND LITERATURWISSENSCHAFT

A Hierarchical Model of German Morphology in a Spoken Language Lexicon Environment

Dissertation zur Erlangung des Grades
Doktor der Philosophie (Dr. phil.)

vorgelegt von

Harald Lüngen

im Mai 2002

Gutachter:
Prof. Dr. Dafydd Gibbon, Universität Bielefeld
PD Dr. Hagen Langer, Universität Osnabrück

Danksagung

Ich möchte meinen wissenschaftlichen Betreuern, Prof. Dr. Dafydd Gibbon und PD Dr. Hagen Langer sehr herzlich und aufrichtig für ihre Bereitschaft, dieses Dissertationsprojekt über mehr als vier Jahre zu betreuen, für ihre verlässliche Unterstützung und auch für ihre Geduld danken. Trotz geographischer Distanz und verschiedener Verzögerungen waren sie immer wieder bereit, sich meiner inhaltlichen oder logistischen Probleme in persönlicher Kommunikation und E-Mail-Diskussionen anzunehmen. Die vorliegende Arbeit hat von ihren Kommentaren, Anregungen und Erklärungen außerordentlich profitiert.

Mein weiterer Dank gilt meinen Freunden und Kollegen in Bielefeld und Helsinki, die mir oft ihre Zeit gewidmet und mich durch Diskussionen, Korrekturlesen und Hilfe bei praktischen Problemen unterstützt haben: Dipl.-Inform. Benjamin Hell, Silke Kölsch M.A., Peter Kühnlein M.A., Martin Matthiesen M.A., Dr. Martina Meister, Claudia Sassen, Katriina Semple B.A., Julia Simon, Caroline Sporleder M.A., Thorsten Trippel und ganz besonders Dr. Andreas Witt.

Ich danke außerdem meinen lieben Eltern Uta und Wolfgang Lüngen dafür, dass sie immer an mich und dieses Projekt geglaubt haben.

Für verbleibende Lücken und Fehler bin ausschließlich ich verantwortlich.

Helsinki/Helsingfors, 8. Mai 2002

Contents

1. Introduction

1.1. Problems of a general theory of morphology

The goal of this thesis is to develop a comprehensive theory of the morphology of German within the grammatical framework of Head-driven Phrase Structure Grammar (HPSG).

Thus, the *feature sets*, the *type hierarchy of lexical signs*, and the *grammatical principles* that describe the morphotactics of inflection, compounding, and derivation in German, are to be defined and well-motivated linguistically. Certain problems in the description of German morphology, which have not been solved so far, or which have been solved within a different grammatical framework, or have been solved without an integration into an overall theory, will therefore be tackled in this thesis. These are, firstly, general morphological problems of the kind that a description of the morphology of any language faces, such as the architecture of the grammar and the situation of the morphological component within the grammar. It must, for example, be formalised what the interfaces to the syntax and to the phonology look like and if and how far there is interaction between these three. A question that is connected to this is what the expressive power of the formalism that is necessary and sufficient to describe the morphology is on the Chomsky hierarchy (Chomsky, 1965b).

But even the question whether an independent morphological component exists at all is controversial. In the early generative tradition represented for example by Chomsky and Halle (1968) and later revived by Lieber (1992), it is assumed that there is no autonomous morphology at all, that the syntax is responsible for arranging the *formatives* of a language in phrase structure rules and transformational rules, and that the phonology provides the spell-out of these formatives as sequences of phonemes. Those rules in the syntactic component that may be interpreted as morphological rules (such as *affix hopping*, cf. Chomsky, 1957) interact with the purely syntactic rules in such a way that it is not possible to draw a line between them and mark an autonomous morphological component.

Another tradition within generative grammar indeed favours an autonomous morphological component, which nevertheless uses the same formalism and rule types as are necessary to describe the syntax while employing a different set of lexical items, features, and principles for morphology. In these *word-syntactic* approaches, there is a clear interface to the syntactic component, usually characterised by the structures that can extend the category **Word**, which is used as the start symbol of a morphological grammar. Such an approach is taken by Selkirk (1982), Di Sciullo and Williams (1987), and Toman (1987). Later, computational word-syntactic morphologies were designed and implemented in unification-based grammar formalisms, e.g. by Ritchie et al. (1992), in the Generalized Phrase Structure Grammar (GPSG) formalism, and Antworth (1994), in a feature and unfication-enriched Two-level morphology (s.b.).

The idea that at least derivational morphology is independent from syntax was introduced by Chomsky (1970) in generative grammar. He claimed that derivational nominalisations are described in the lexicon by *lexical redundancy rules*. This approach was extended to all derivational morphology by Halle (1973), who formed the term *word formation rules (WFRs)* for the morphological rules that operate on the lexicon. The nature of WFRs was subsequently further studied and specified by the proponents of *word-based*, or *realisational* morphology, most notably in Aronoff (1986), Anderson (1992), and Aronoff (1994). A major characteristic of the WFRs is that they operate on existing lexemes (words), not morphemes, which play no role in realisational morphology.

The theory of Lexical Phonology and Morphology developed in Siegel (1979), Kiparsky (1982a,b, 1985), and Mohanan (1982) stands in the tradition of this realisational generative approach as well, but it postulates an extended lexicon with multiple ordered strata and restrictions on the application of the affixation rules and the phonological rules on each stratum.

Other recent, computational linguistics-oriented directions of scholarship postulate an independent morphology component, which must be described using an altogether different formalism than the syntax. One of these approaches is Two-level morphology (Koskenniemi, 1983a, 1985; Karttunen and Beesley, 1992, and cf. Section 2.2). Its two-level rule component and system of continutation classes correspond to finite-state automata, which are equivalent to regular grammars (type 3 grammars on the Chomsky scale). For comparison, note that transformational grammars are unrestricted production systems (type 0 grammars), and that feature-based grammars that employ recursively defined feature structures (such as HPSG, cf. Section 3) are known to be at least as powerful as context-free grammars (type 2 grammars).

In the 1990s, a number of approaches which describe morphology as a phenomenon that arises from the complex hierarchical organisation of the lexicon and its interaction with phonology have been published. Several of them employ the default inheritance lexical representation formalism DATR (Evans and Gazdar, 1996) for this purpose (Gibbon and Reinhard, 1991; Corbett and Fraser, 1993; Cahill and Gazdar, 1997, 1999; Gibbon, 1991; Bleiching, 1994), but approaches employing the formalism of strictly typed feature structures of HPSG are also available (Riehemann, 1993, 1998, 2001; Gibbon, 1997; Koenig and Jurafsky, 1994; Koenig, 1999).

Beside the architecture of the grammar and the position of morphology, the “model of grammatical description” (Hockett, 1954) to be employed in morphology has always been an object of controversy. Are affixes rules or signs? Is there such a thing as morphological constituency which corresponds to syntactic constituency? Does the morphological lexicon contain lexemes or morphemes? Is the notion of an inflectional *paradigm* an independent entity within the morphology of a language, or does it arise from the organisation of morphological schemata? Morphologists have come to different conclusions, not only for different languages but even for the different areas of morphology within one language. Inflectional endings are not considered to be suffixes in the sense of linguistic signs by many scholars. Instead, they are frequently regarded as phonological material introduced for example in word formation rules (Aronoff, 1986, 1994), morpholexical rules (Anderson, 1982, 1992), paradigm functions (Stump, 1991), or HPSG-lexical rules (Pollard and Sag, 1987). This is motivated by the fact that substantial parts of inflection in many languages show other phonological processes besides agglutinative affixation, which in such approaches can be treated identically, i.e. by the same kind of rules. Consequently, a combination of a stem plus an inflectional affix does not have an internal constituent structure in these approaches. Some scholars extend this view even to derivation, most notably the proponents of word-based morphology, who claim that there are no morphological objects other than existing *words*, and that each morphological rule operates on words. Affixes, including derivational affixes, do not exist as listemes (items listed in a lexicon) in such a theory. Again, derivational affixes can only be found as phonological strings that are concatenated with words/stems in lexical redundancy rules or schemata.

Contrary to inflection and derivation, it is quite uncontroversial that the area of compounding involves constituency and operates on lexemes (in the form of stems or words), which are linguistic signs and listed in the lexicon, regardless of whether compounding is viewed as a part of syntax or of morphology/the lexicon.

1.2. Problems of a theory of German morphology

Feature percolation and headedness in syntax have been well-studied since the advent of X-bar theory in Chomsky (1970) and Jackendoff (1977), and, consequently, the concepts of HEAD features and HEAD feature percolation play a central role in the feature-based grammar frameworks of GSPG (Gazdar et al. (1985), as the *Head Feature Convention*), and HPSG (Pollard and Sag (1987, 1994), as the *Head Feature Principle*), too. Several authors have presented a corresponding feature percolation principle for morphology, but with surprisingly different and partly contradicting results, depending on different factors such as the particular language and the subfield of morphology under scrutiny, but also on the employed model of grammatical description and the assumed feature set. (It is for example, a priori excluded that an inflectional affix may function as the head of a word when morphological analysis is conducted in a word-and-paradigm framework.) In this thesis, we will therefore carefully try to establish a consistent Head Feature Principle for the morphology of German, taking into account the feature set to be employed, and the differences of feature percolation in inflection, derivation, and compounding.

The theory developed in this thesis comprises an HPSG subtheory on nominal inflection in German. It deals amongst other things with the hierarchy of nominal inflectional classes (formerly examined outside HPSG in e.g. Bleiching (1992), Cahill and Gazdar (1999), and Bleiching and Gibbon (2000)), syncretism in the nominal inflectional paradigm (Bleiching et al., 1996), and the role of umlaut in noun inflection (Wiese, 1987; Gibbon and Reinhard, 1991).

The important fields within verbal inflection to be modelled are the hierarchy of verbal inflectional classes, the description of regular (agglutinative) inflection vs. ablaut-based, irregular inflection, the hierarchy of ablaut patterns, and the question which grammatical categories participate in verbal inflection. Verbal inflection is also interesting with respect to headedness in morphology, as the past tense forms of regular (weak) verbs is formed by adding *two* separable inflectional suffixes to a stem, cf. Section 5.2.3.

For adjectival inflection, the descriptions of the set of adjectival inflectional categories and of the inflectional syncretism is crucial, cf. e.g. Cahill and Gazdar (1997). Inflectional class plays no role for German adjectives.

What distinguishes inflection from derivation, and what do they have in common? We aim at answering this question for German and formalise the results. Within derivation, too, the problem of headedness needs to be solved, and many possibilities have been explored in previous studies. William's (1981) *Right-hand Head Rule* makes the suffix the head in a derivational suffixation, but makes the base the head in a derivational prefixation. However, it is sometimes claimed that the base must be the head in any kind of affixation. Then again, Brehmer (1985) claims that there are certain (category-changing) prefixes which are the heads of the words derived with them. We will analyse feature percolation in derivation closely to determine what is descriptively adequate. Further results of previous research in derivational morphology to be incorporated into our HPSG theory are the morphotactic properties of native vs. nonnative lexical roots and affixes, and the modelling of the lexical strata and the different types of affixes from Lexical Phonology and Morphology.

The question how to treat umlaut in derivation has also been an object of research in the past. It is well-known that the derivational relations involving umlaut are more complex than in inflection, the difficulty lying in the fact that an umlautable base is not always umlauted when combined with an umlaut-triggering derivational suffix (Zwicky, 1967; Reinhard, 1991). We will search for a solution to this problem within HPSG. Furthermore, the role of derivational alternations involving ablaut and conversion has to be clarified and properly encoded.

The description of compounding in German does not pose so many problems on the morphological level. Still, there are areas of special interest such as the status and proper classification of linking morphemes (*Fugenmorpheme*), where it has been suggested that these are largely dependent on inflectional classes (Gibbon, 1991; Langer, 1998), with the exception of `-s` with feminines (`Arbeitsamt`) and analogical formations such as `Sternenhimmel`, with a dative plural link, and the constituent structure of compounds, especially in so-called *synthetic compounds* such as

`Messebesucher` or `Bahn-Card-Besitzer`, where compounding seemingly interacts with derivation. A further challenge is presented by the class of *phrasal compounds* (`(das) Auf-Nummer-Sicher-Gehen, (die) Hin-und-Zurück-Angelegenheit`), as here, morphological rules operate on syntactic phrases.

Many insights from the theories mentioned in Section 1.1 can be formalised in HPSG in a non-contradictory and non-redundant way and will be useful in answering the questions raised in this section. We will shortly explain further why we think that HPSG is most suitable to achieve an adequate description of the morphology of German. Let us first provide some introductory words about the HPSG grammar framework.

1.3. Sign-based morphology

HPSG (Pollard and Sag, 1987, 1994) is a linguistic theory which emerged from the tradition of generative grammar and uses formal devices and notions from previous unification-based theories of grammar, especially Functional Unification Grammar (FUG, Kay, 1979), Lexical Functional Grammar (LFG, Bresnan, 1982), Generalised Phrase Structure Grammar (GPSG, Gazdar et al., 1985), and Categorial Unification Grammar, (CUG, Uszkoreit, 1986). HPSG models of grammar fragments are described in terms of well-defined mathematical objects (*feature structures*, cf. Section 3), and are represented using the notational formalism of attribute-value matrices (AVMs). Partly because of this formal background, HPSG is widely used in natural language processing, see for example the large-scale applications developed by Müller (1999, 2000), Uszkoreit et al. (1994, 2000), and Copestake (1999). Another reason why computational linguists are attracted to HPSG is that contrary to current generative approaches to grammar, it is a *monostratal* theory of grammar, which means there is only one stratum for declarative representation of models of linguistic objects, i.e. there are not different representational levels such as D-structure and S-structure, and no computationally costly transformations to mediate between these are needed.

The central linguistic object modelled by feature structures in HPSG is the *sign*. A structuralist notion of sign is adopted, i.e. a sign is a structured object which binds together a form and a meaning component. Traditionally, a linguistic sign is identified with a *word* in the sense of a *lexeme* or alternatively, a *word form*. In HPSG and related theories, all objects modelled by feature structures with PHONOLOGY and SEMANTICS attributes are signs, too, so that there are different *types* of signs. Phrases and sentences are *phrasal signs*, and lexemes are *lexical signs*. Through the DAUGHTERS (DTRS) attribute in phrasal signs, constituent structure is represented in a feature structure. There are also lexicalised phrasal signs, which may be called *idioms*. Gibbon (1997) points out that even phonemes may be regarded as a special type of sign, which have a "purely structural 'meaning' ". But also higher level linguistic units, such as utterances or texts, are signs in that sense. These types of signs are composed of signs from a lower level (or, *rank*, after Jespersen, 1933; Halliday, 1985) of linguistic description. Signs can be syntactically, semantically, and phonologically compositional. Lexical signs, and recently also phrasal signs (cf. Sag, 1997; Ginzburg and Sag, 1999), are associated with types, which are ordered in type hierarchies. Type restrictions can be represented as feature structure descriptions, too, but they are not signs. Two fundamental possible relations between signs are thus *constituency* (one sign may be embedded in another, more complex, sign), and *type subsumption*, i.e. two signs may have a common supertype (cf. also Koenig, 1999, pp.51). The formalism of typed feature structures may be complemented by lexical rules (as in Pollard and Sag, 1987), by means of which further relations between signs can be defined. They play a role especially in morphology, but their adequacy and necessity is has often been challenged.

Although HPSG is an established grammatical framework, which even includes several assumptions about universal grammar (such as the structure of a linguistic sign) and human language in general, both its formalism and its sign-based modelling conventions are quite unbiased with respect to many of the design features that distinguish the morphological theories mentioned above. It is therefore a suitable instrument for studying the morphology of German.

In fact, several different approaches to the description of morphology in HPSG have been put forward previously: In Pollard and Sag (1987) and Flickinger (1987), lexical rules are introduced as the major device to describe the inflectional morphology of English (derivation and composition are not treated in Pollard and Sag, 1987). Lexical rules can be interpreted as an item-and-process model of morphology in the sense of Hockett (1954), see Section 2.1. They have been criticised for being an alien element in the HPSG framework from a formal point of view: They are neither types nor signs nor anything which can be modelled as a feature structure, instead, they represent non-monotonic operations on feature structures. Moreover, their interpretation is not unambiguous, see Section 4.3 for a formal introduction and discussion of lexical rules.

Krieger (1993) provides an item-and-arrangement account of German derivational morphology in HPSG, using the HPSG attribute-value structures and principles to describe a classical word grammar with an integrated semantics, where morphemes are fully-fledged linguistic signs. One of the author's aims was to eliminate lexical rules from the description of morphology. This approach will be discussed in Section 4.4.

Riehemann (1993, 1998, 2001) criticises this approach because of systematic redundancies and introduces *schemata* for a description of a fragment of German derivational morphotactics in HPSG. Schemata arise as lexical types over existing derived words, giving morphemes no independent status. This approach (discussed in Section 4.4) is not easily categorised in terms of Hockett's (1954) classification.

Most current (not only HPSG-related) computational models of the lexicon and morphology are declarative in the sense that they can serve as knowledge bases for any kind of morphological processing. From a computational point of view, though, word-and-paradigm models seem to have been preferred for morphological generation applications, since in these, the access to the lexicon is via the lemmata, and the relevant information about inflectional affixes is stored immediately at the lemma entries, cf. Calder (1989) and Bleiching et al. (1996).

Item-and-arrangement approaches lend themselves more easily to morphological analysis applications, as in such tasks, morphological segmentation has to be performed first, and lexical access is done via the morphs obtained through the segmentation. Once there is a lexical entry for each morpheme where information about its combinatorial (morphotactic and morphosemantic) possibilities is stored, a point is obviously reached where a morpheme is a concept that associates a form with a meaning and must be considered to be a linguistic sign.

1.4. Goals

1.4.1. Linguistic adequacy

One goal of this thesis is to provide a descriptively adequate theory of the morphology of German. Linguistic adequacy criteria for a grammar go back to (Chomsky, 1964, p.29ff). A grammar is *observationally adequate* "if it presents the observed primary data correctly". This means that a theory of morphology is observationally adequate if it is capable of distinguishing words and non-words in a corpus of linguistic data. "A second and higher level of success is achieved when the grammar gives a correct account of the linguistic intuition of the native speaker, and specifies the observed data (in particular) in terms of significant generalisations that express underlying regularities in the language." A descriptively adequate morphology thus assigns lexicalised words as well as potential words structures and expresses part of a native speaker's morphological competence. *Explanatory adequacy* may be reached within a theory of grammar that formulates criteria for picking the best theory among several descriptively adequate theories. According to Chomsky and Halle (1968), such criteria are based on "external evidence" from neighbouring fields such as psycholinguistics and cognitive science, e.g. whether a morphology can explain phenomena of child language acquisition.

But we also aim at something that might be called *computational adequacy.* The morphology will be described in a formalism for which several implementations exist. Thus it can be used as *lingware* in a linguistic analysis or generation system, which may be a component in a more complex system, such as an automatic speech recognition system, or a machine translation system (see Section 1.4.2). It is in fact one of the tasks of computational linguistics to provide linguistically well-motivated grammars for such systems, and in turn, a successful computational implementation represents a proof of the well-formedness of a theory.

The theory developed in this thesis is also intended to be *comprehensive*, including the areas of inflection, derivation and compounding for all word classes. With the help of this, we will also seek an answer to the question, whether it is necessary to use a feature-based grammar formalism for an adequate description of German morphology at all, as it has been suggested previously, that less powerful grammar formalisms such as finite state automata suffice for the description (of at least the main areas) of morphology (Koskenniemi and Church (1988), but see also Carden (1983)).

The Verbmobil subproject *Lexicon and Morphology* (see Section 1.4.2) dealt with a lexicon containing phonological and surface-form related morphological information about words and sub-word units to be used in speech processing. This project, though, is situated on the boundary between speech processing (dealing with sub-word units) and language processing (dealing with words and linguistics units larger than words), so our morphology is supposed to be augmentable with further types of linguistic information, such as semantic constraints on morphological construction types.

Moreover, the thesis will provide results that have an impact on a general theory of morphology across languages. However, we will be careful in the formulation of proposals in that direction, as we believe that principles of universal grammar/morphology can only be formulated as a consequence of comparing the grammars/morphologies of many languages, which must be ideally be provided in identical frameworks and especially formalisms first. This also implies that this thesis will provide a test for previously made universal or typological claims within the HPSG framework.

1.4.2. Speech applications of a morphological theory

This thesis emerged from the author's work in the automatic speech-to-speech translation project *Verbmobil* between 1995 and 2000, in the subprojects called *Lexicon and Morphology* and *Generation of pronunciation dictionaries.* Verbmobil ran for eight years between 1993-2000, in two phases 1993-1996 and 1997-2000. It is documented comprehensively in Wahlster (2000b). Verbmobil's final deliverable was a "speaker-independent and bidirectional speech-to-speech translation system for spontaneous dialogs in mobile situations" (Wahlster, 2000a, p.3). The system is able to process a vocabulary of 10157 German words (full word forms). This is quite a high number compared with other systems with a similar task. However, the number is very low in view of the number of actual words and lower still with respect to the potential number of words of the German language; it only makes sense under the condition that Verbmobil is to be used in a limited domain, viz appointment scheduling and travel planning. Generally, the ratio between full words form types and stem (lemma) types in German texts, varies between 3:1 and 5:1, due to inflectional variation depending on the text type. English, in contrast, has a word form variation factor of only slightly ≥ 1 (Gibbon and Lüngen, 2000). German also differs from English in the higher rate of word formations (derivatives and compounds) in the vocabulary. In addition, these word formations are consistently transcribed as single words in transcriptions of speech, whereas in English transcriptions, they are very frequently orthographically segmented by blanks or hyphens, resulting in some inconsistency in the available figures, cf.

- `einundzwanzig` vs. `twenty-one`
- `Reisebüro` vs. `travel agency`
- `nachzuschlagen` vs. `to look up`

Moreover, a rather large percentage of the *out-of-vocabulary items* encountered by the system in test runs are compounds, derivatives, and inflectional forms of stems that are already included in the vocabulary. It is well-known that this percentage increases with increasing vocabulary.[1]

One of the ways to cope with this fact is to inventorise the morph(eme)s instead of or in addition to the words of a language with rich morphology in the lexica of speech recognition systems and to perform morph recognition instead of word recognition. Words are known to be composed of morphs and can therefore still serve as the interface between speech components and language components in a speech recognition architecture enhanced by a morphological processor. The question of how to include which kind of morphological knowledge in the speech recognition process to gain the best results is still an object of research, and several experimental word recognition systems dealing with this problem have been implemented and evaluated in recent years, e.g. Geutner (1995), Berton et al. (1996), Lüngen et al. (1996), Althoff et al. (1996), Althoff (1997), Strom and Heine (1999), and Pampel (1999) in the Verbmobil context.

Whereas current word recogniser dictionaries frequently consist of simple pronunciation tables, it is obvious that a more elaborately structured lexicon is needed to include morphotactic, morphographemic, and morphophonological properties of word forms. Such a lexicon has been developed during Verbmobil Phase I (1993-1996), see Gibbon and Ehrlich (1995), and Bleiching et al. (1996). The extensional coverage of this core lexicon for speech processing is the vocabulary of the transcriptions of the Verbmobil German dialogue corpus, which amounted to 11398 lemmata in the year 2000. Its intensional coverage comprises components especially needed for speech processing, i.e. orthographic and phonemic surface form, lexical prosodic and syllable structure, morphological segmentation and structure, with pointers to morphological constituents and their properties. Its lexicon model is based on ILEX (Integrated Lexicon with Exceptions, Gibbon (1991, 1997)), but we will provide an HPSG model for this lexicon in this thesis. The HPSG theory of morphology developed in this thesis was employed in the morphological analyser and lexical acquisition program MCLASS (implemented in Prolog). The MCLASS system is presented in Section 6.

[1]cf. e.g. Geutner (1995), Matthiesen (1999), Mengel (1999).

1.5. Method and structure of the thesis

We will first examine the state of the art of morphological theory. Section 2 deals with current generative models and models of computational morphology. The formal foundations of the formalism of typed feature structures, which is used in HPSG, are presented in Section 3. We will then have a closer look at previous approaches to morphology and the lexicon within HPSG. The emphasis will be on studies of German morphology (as well as syntax as far as it is relevant), but also other languages are taken into account when it is expedient (Section 4). The central part of this thesis is then Section 5, where the morphology of German is studied with respect to particular sub-areas such as headedness, prefixation, or nominal inflection. We also take various previous approaches to these topics into consideration and evaluate them critically. The results of these evaluations are made compatible by formulating them in terms of typed feature structures, i.e. feature appropriateness specifications, the subsumption relations between lexical types, and principles of morphological constituency. Some principles of HPSG such as the interaction of the Head Feature and Marking Principle, or the structure of the attribute SYNSEM are also evaluated critically as a consequence of the analysis of German. In Section 6, the lexical acquisition program MCLASS, which uses the HPSG theory of German morphology for its DCG morphotactics, is presented.

As linguistic data for the analyses, we partly reuse examples from the literature, partly provide examples introspectively, but for the most part we take words from the Verbmobil dialogue corpora in the form of the morphologically annotated Bielefeld lexical database (Lüngen et al., 1998; Gibbon and Lüngen, 2000).

1.6. Basic terminology and notes on typography

One goal of this thesis is to provide exact definitions of central terms such as morph, morpheme, lemma, paradigm, and stem within the morphological theory to be developed. However, we want to discuss previous approaches already before the introduction of the actual theory with the help of these concepts. In these discussions, we generally adopt the meanings these terms have in classical structuralist studies such as Bloomfield (1933), or as they are presented modern textbooks (Spencer, 1991, p.3ff). The following is an overview of morphological terms and what they are supposed to denote (as well as what they are supposed *not* to denote) in this thesis:

Morph: A minimal meaning-bearing unit of language. Morphs have surface representations in terms of phoneme or grapheme strings, thus they can be segmented in running text. We also subsume under morphemes elements which might not be considered meaning-bearing by some scholars, such as the `-ig` in `Helligkeit`, or the `-et-` in `theoretisch` because we consider them to have a structural meaning, for example that a whole series of word forms shares them (cf. `theoretisieren,` `pathetisch, apologetisch, Apologet`), or that stress assignment is sensitive to them. In other words, we claim that the word forms `Einigkeit` and `Helligkeit` should have the same morphological structure, e.g. that they are constituted by a sequence of root+suffix+suffix, although `-ig` seems to be an adjective-forming suffix only in `Einigkeit`.

A *morpheme* denotes a class of morphs that share the same meaning or function but differ in their distributional and/or surface form properties. Thus morphemes are more abstract entities, and additional symbols that represent morphophonemes (morphologically conditioned classes of phonemes) and morphographemes may be employed to represent their phonological and graphemic *underlying* forms. *Allomorph* is then a relational term defined with the help of morph and morpheme: A morph A which is a realisation of the morpheme B is an allomorph of B. Thus, `sprach` and `sprich` are orthographic allomorphs of `//sprEch//`.

Free morphs are those that can appear as a word form (s.b.) in their own right (`Büro, war, aber`), whereas *bound* morphs are morphs that only appear as a subpart of a word form (`-e, -iv, werb-`).

That part of a word form that is left when all affixes are stripped off, is called a *root* morph. A class of root morphs with the same meaning forms a root morpheme. Not all root morphs in German are free forms. A *base* is that part of a word form to which any other morph is added. An *affix* is a bound morph(eme) that must be affixed to a base (s.b.) to form a new base. In German, an affix must always be affixed either only from the left (a *prefix*) or only from the right (a *suffix*). (So-called *circumfixes* may be analysed as two morphs, a prefix and a suffix.)

A form that may combine with an inflectional suffix to form a word form is called a *stem*. *Base* and *stem* are not always distinguished in the literature, but (Spencer, 1991, p.461, footnote 10), for example, makes the distinction in the sense in which we make it here. In the discussion of previous studies of morphology, however, we sometimes use *stem* in the sense it was originally used by the author(s), which often corresponds to our *base*.

A *lemma* is a class of word forms belonging to the same paradigm. A lemma is also viewed as a class of stems that belong to the same paradigm. We consider a *lexeme* to be a lemma associated with a specific meaning.

A *word form* is any form that may occur freely, i.e. as a syntactic atom. The term *word* is used in this thesis sometimes for *lemma* and sometimes for *word form*. Which one is meant is either unambiguous from the context, or considered irrelevant for the argumentation. Note that in the presentation of our HPSG morphology, *word* receives the specific meaning of denoting the lexical type that subsumes word forms.

A *paradigm* is the set of word forms that bear certain morphosyntactic descriptions and are associated with one lemma. (So far the definitions of ‘paradigm’ and ‘lemma’ are cyclic, but they won't be in the morphological theory we are going to present.)

Among the basic morphological operations, *inflection* is the one by means of which certain morphosyntactic categories are marked on a stem, i.e. the declination of nominals, and the conjugation of verbs. In *derivation*, however, a new base is formed from one base, involving the modification of syntatic and semantic properties of the original base. Finally, in *compounding* two bases are combined to form a new base.

All other central terms, especially those that receive a technical meaning in HPSG and in our HPSG morphology are defined when they are introduced.

As *italic font* is used for type symbols in HPSG, which occur quite frequently in this thesis, we cite word forms, stems and morphs throughout this thesis in `typewriter font`. Sometimes a form may still be cited in italics, this means that it represents the respective more abstract entity (morpheme or lemma).

Finally, note that in examples of lexical entries, we transcribe all surface orthographic forms, phonological forms, and morphological boundaries according to the Verbmobil lexical conventions for spelling and pronunciation, (see Gibbon, 1995) but in the remaining examples (mostly those in running text) we do not make use of the TeX German umlaut notation, i.e. we spell `Füße` and not `F"u"se`. For phonological tanscriptions, the Verbmobil conventions imply the use of the SAMPA phonetic symbols for German, the relevant version of which is found in the same source.

2. Some previous approaches to morphology

2.1. Two models of grammatical description

In the article “Two Models of Grammatical Description”, Hockett (1954) compares *Item-and-Arrangement* (IA) models of grammar with *Item-and-Process* (IP) models of grammar. Hockett had recognised that one or the other kind of underlying models lay behind the work of different American structuralists. He also mentioned a third, independent model of linguistic description called *Word-and-Paradigm* (WP). These models, apart from WP, can be applied to the description of any level of grammar, though Hockett uses examples only from the domains of morphology and morphophonology.

In an IP model, the difference between two partially similar forms is frequently described as a process which yields one form out of the other.[2] Thus, the form `baked` is derived from the form `bake` by means of a process called PAST TENSE FORMATION. The same process is responsible for derivation of `took` from `take`. Different *markers* are appropriate in this process for each of the roots `bake` and `take`, explaining the fact that in one case the actual phonological operation is a suffixation and in the other it is an apophony.[3] A grammar then consists of a list of the simple forms to which processes can be applied, and a detailed description of all the processes. What would count as an affix in an IA approach is not a morpheme, not even a linguistic sign in an IP approach. It is simply a marker which has no meaning and no independent status outside a process.

In an IA model, however, the basic ingredients of the grammar are a list of morphemes and an inventory of constructions with a detailed description of each construction. Morphemes thus have an independent status, they are associated with a form and a meaning and they are listed in a lexicon, thus, they are signs. They exist independently of the rules (constructions), into which they are embedded.

Thus, one of the main differences between the two models is the status of morphemes, especially affixes, which are absent in an IP model. Another crucial difference is that an IA model seems to involve only one kind of phonological operation, which is concatenation; operations like mutation or deletion are excluded since functional and semantic categories have to be associated with identifiable and segmentable items. For this reason, in strict IA-models, additional concepts such as zero allomorphs, portmanteau morphs, or prosodic morphemes must sometimes be introduced.

[2]Cf. (Harris, 1939, p.199).

[3]In Spencer (1991), it is falsely claimed that Hockett regards the above two different past tense formations as the outputs of two different processes. A process, as Hockett explains it, is rather defined by the forming of one grammatical, functional or semantic category, and different phonological operations involved in a process are regarded as different *markers* of the process, which depend on the *root* that is the input to the process.

A classical transformational grammar as presented by Chomsky (1965a) incorporates both models: the base component, characterised by phrase structure rules, follows an item-and-arrangement model of grammar whereas the transformational component conforms to an item-and-process model.

The Word-and-Paradigm model of morphology briefly mentioned by Hockett was first discussed in detail by Robins (1959) and further developed in Matthews (1974). It was designed as an answer to some shortcomings of both the IA and the IP model of morphology. The latter are adequate in cases where there is a one-to-one correspondence between morphological form and morphological function, i.e. in which one function can be identified with exactly one affix or one process. However, this is probably never the case across all morphological processes to be found in one language, though agglutinating languages such as Finnish or Turkish can come quite close to this. Cases of one-to-many or many-to-one correspondences between form and function have been handled by remedies such as portmanteau morphs and zero allomorphs, or complex processes and processes without markers in IA and IP models, but when it comes to many-to-many correspondences as in the inflectional morphology of highly fusional languages such as Latin (cf. Figure 1)[4], IA and IP models fail to capture important generalisations.

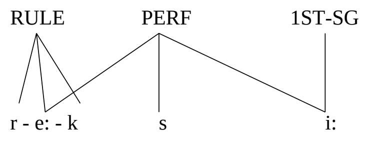


Figure 1: Many-to-many correspondences between form and function in Latin `rexi` ("ruled").

The basic idea in the WP approach is to formulate these generalisations at the level of the *morphosyntactic word*, i.e. in the description of fully inflected word forms where morphosyntactic functions and phonological exponents can be stated totally independently of each other if necessary. In order to achieve this, the notion of a *paradigm* as a linguistic object is introduced.

A paradigm is the set of all declined or conjugated word forms that are associated with one particular lemma. A paradigm class is then a class of lemmata for which the phonological exponents for expressing the morphosyntactic categories are the identical.

Consequently, the different declension and conjugation classes characteristic of fusional languages like Greek, Latin, or Sanskrit can easily be described and related to each other. Another result is that inflectional syncretism can be described adequately in this model, abounding also in languages like German or Old English.

In computational morphology, the WP model has been adopted and implemented in

[4]after Spencer (1991).

one or the other variation by Calder (1989), Bleiching et al. (1996), Cahill and Gazdar (1997), and Cahill and Gazdar (1999).

2.2. Two-level morphology

The core of the two-level model is actually an alternative to traditional *phonological* rewrite rules as used for example in the SPE (Chomsky and Halle, 1968). It had been discovered earlier that such rewrite rules correspond to finite-state transducers, given some additional restrictions that were linguistically perfectly acceptable (cf. Johnson, 1972; Karttunen, 1983). The operation of composing large cascades of rewrite rules to one single transducers turned out to be computationally unfeasible in many cases, though, because the resulting transducers would become too large. Koskenniemi's (1983) solution was the invention of the so-called two-level rules, which still correspond to finite-state transducers but apply in parallel. The rules must be written such that each surface/lexical (the two levels) character pair is seen by all transducers at the same time. Two-level rules were successfully applied to describe morphophonological phenomena such as Finnish consonant gradation, triggered amongst other things by morphological border symbols in the rule contexts. The description of the morphotactics within Two-level morphology relies otherwise on a system of *continuation classes*, i.e. lexica of morph categories. Each morph entry containts a pointer to a continuation class. Both components, the two-level rule component and the system of continuation classes, correspond to finite-state automata.

Finnish morphotactics is relatively simple, though (e.g. all derivation and inflection is exclusively suffixing), and the model is not designed to express morphological constituency. With the German word form $\texttt{[un[[denk]}_V\texttt{bar]}_{ADJ}\texttt{]}_{ADJ}$, for example, the continuation class approach fails, because when `un-` is prefixed to `denk`, it is not known that an adjective base follows as the suffix `-bar` is not seen in the immediate continuation lexicon.[5]

The fact that it forms adjectives from adjectival bases is nevertheless an important morphological constraint on `un-`prefixation, which one would want to express in the morphology. The semantic interpretation of compounds, or stress assignment to compounds in German also rely on a representation of morphological constituency (cf. Bleiching, 1991) and the two-level model is simply not designed for such tasks (cf. also Sproat, 1992, p.152f).

In a Two-level morphology, the morphotactics is described by IA means: The only possible operation is concatenation, and other operations such as ablaut are described

[5]Morphological "long-distance" dependencies like this may be described by splitting the possible continuations after `un-`, but such a treatment fails to capture the generalisation that `un-` is a deadjectival adjective-forming prefix.

by introducing different allomorphs to be concatenated under certain conditions. The morphophonology, expressed in the actual two-level rules relating the representations on the surface and the lexical level, corresponds to an IP model, if we consider the items to be the lexical forms. If we consider the feasible pairs of lexical and surface form (morpho-)phonemes to count as the items in question, the model conforms to an IA phonology. The evaluation of the two-level rule model with respect to an IA vs. IP approach remains somewhat unclear, since the rules apply always in parallel and, unlike in the case of SPE phonological rules, it is not possible to consider the morphophonological process described by one two-level rule isolated from the rest of the rules.

2.3. Morphology in generative grammar

2.3.1. Word syntax

Selkirk (1982) proposed an approach to morphology called *Word syntax.* Adopting the Government and Binding model of syntax by Chomsky (1981), she designed a model of morphology using means familiar from generative syntactic theory. These means are the formalism of context free phrase structure rules, a version of X-bar Theory to constrain the set of possible PS-rules, and a theory of headedness in morphology. In her approach, the maximal projection in a Word syntax is the interface to the sentence syntax, i.e. it is identical with the zero level projection in sentence syntax, the lexical category, that is, the *Word.* While syntax and morphology share the descriptive formalism and the important principles of X-bar Theory, Selkirk emphasises the autonomy of Word Syntax, in which categories and additional principles distinct from those to be found in sentence syntax are employed. All Word syntax is incorporated in the lexicon, the morphological rules serve both as lexical redundancy rules and to assign meaning and structure to newly created words. The morphological base component consists of the PS-rule part and of a morpheme dictionary comprising a sub-dictionary of free morphemes plus a list of affixes. In addition to the non-terminal category *Word*, there are two other categories, the non-terminal *Root*, and the terminal *Affix.*

Lexical entries for affixes include the following (cf. Selkirk, 1982, p.64):

- Category, including type (always *Affix*), syntactic category features, and diacritic features
- Subcategorisation frame
- Semantic functions
- Phonological representation

A difference between morphology and syntax is that in morphology, recursion in the PS-component is highly constrained in that a *Root* cannot dominate a *Word.* This means that no structures parallel to a $\bar{\mathrm{V}}$ dominating an NP in syntax are possible in morphology.[6]

Selkirk's general X-bar schema for affixation looks as follows:[7]

2.1
$$\begin{array}{llcll}
\textit{(a)} & \text{Word} & \longrightarrow & \textit{Affix} & \textit{Word} \\
\textit{(b)} & \textit{Word} & \longrightarrow & \textit{Word} & \textit{Affix} \\
\textit{(c)} & \textit{Root} & \longrightarrow & \textit{Affix} & \textit{Root} \\
\textit{(d)} & \textit{Root} & \longrightarrow & \textit{Root} & \textit{Affix} \\
\textit{(e)} & \textit{Word} & \longrightarrow & \textit{Root} &
\end{array}$$

The terminal symbols 'Root' and 'Affix' may be expanded by lexical insertion. The rules under 2.1 are to be viewed as informal versions of the following:

2.2
$$\begin{array}{llcll}
\text{(a)} & \mathrm{X} & \longrightarrow & \mathrm{Y}^{\mathrm{af}} & \mathrm{X} \\
\text{(b)} & \mathrm{X} & \longrightarrow & \mathrm{Y} & \mathrm{X}^{\mathrm{af}} \\
\text{(c)} & \mathrm{X}^{\mathrm{r}} & \longrightarrow & \mathrm{Y}^{\mathrm{af}} & \mathrm{X}^{\mathrm{r}} \\
\text{(d)} & \mathrm{X}^{\mathrm{r}} & \longrightarrow & \mathrm{Y}^{\mathrm{r}} & \mathrm{X}^{\mathrm{af}} \\
\text{(e)} & \mathrm{X} & \longrightarrow & \mathrm{X}^{\mathrm{r}} &
\end{array}$$

In the latter set of rules, we can firstly see that all elements of Word syntax not only belong to the morphological categories *Word*, *Root*, and *Affix*, but also to the syntactic categories N, A, V, and P, which is what X and Y may stand for. Thus, as in X-bar theory for sentence syntax, more general, category-independent rule schemata replace a larger set of single rules. The second rule under 2.2, for example, describes among other things the following, more specific rules generating derivatives of adjectives:

2.3
$$\begin{array}{llcll}
\text{(a)} & \mathrm{N} & \longrightarrow & \mathrm{A} & \mathrm{N}^{\mathrm{af}} \\
\text{(b)} & \mathrm{V} & \longrightarrow & \mathrm{A} & \mathrm{V}^{\mathrm{af}} \\
\text{(c)} & \mathrm{A} & \longrightarrow & \mathrm{A} & \mathrm{A}^{\mathrm{af}}
\end{array}$$

Secondly, in 2.2, we can see that in an affixation according to Selkirk, the *head* is always the rightmost element, as the left-hand side (LHS) of a rule is always of the same category X as the right-hand element of the right-hand side (RHS). Selkirk discusses headedness at length, considering arguments from derivation as well as from compounding and inflection. The topic of headedness in morphology will be introduced and discussed in detail in Section 5.2 in this thesis.

A third aspect of Selkirk's general rule schemata is that they include results from the theory of Lexical Phonology and Morphology (LPM) developed in e.g. Siegel (1979), Mohanan (1982), Kiparsky (1982a), and Kiparsky (1985). Selkirk points out that class

[6](cf. Spencer, 1991, p.198).

[7](cf. Selkirk, 1982, p.95).

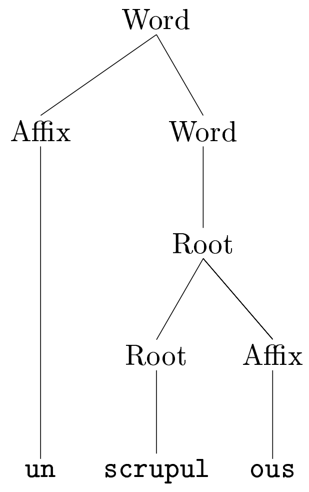


Figure 2: Tree structure of `unscrupulous` according to Word syntax

I affixes from LPM are those that can only be attached to the category *Root* in the Word syntax, again yielding *Root*. Class II affixes can be attached both to *Word* and to *Root*, yielding *Word*. The latter possibility is only indirect, as *Word* may expand to *Root*. The Word affixes and the Root affixes are in principle disjoint, and the restriction whether they attach to Root or Word is expressed in the subcategorisation frames of the affixes. Thus, the categories *Root* and *Word* remodel the domains of level I and level II affixation of LPM, respectively, while level I affixation is generated before level II affixation, which is also in accordance with LPM. Root and Word also play a role in compounding and inflection: English native compounding is always a concatenation of two *Word*, and inflectional suffixes are generated as sisters of *Word*, not of *Root*, cf. (Selkirk, 1982, p.53f and 71ff).

Selkirk's Word syntax is a comprehensive item-and-arrangement morphology for English within the framework of generative grammar. Moreover, it is a demonstration of the descriptive power of the Word-Syntax/item-and-arrangement approach to morphology and gave rise to a number of elaborations of the theory, such as Toman (1987) and Lieber (1992). Still, Selkirk's Word syntax is designed to account for concatenative morphology only (which seems to be sufficient for English productive morphology). Forms like *geese* and *goose* are evidently regarded as different morphemes, the fact that they are morphologically related is not accounted for by Word Syntax.

In view of later feature-based item-and-arrangement morphologies as e.g. in Ritchie et al. (1992) or Krieger (1993), one can say for the that Selkirk's approach also could be enhanced with a more elaborate theory of morphological features and types, which could introduce some more generalisations. Categories may be explicated as (complex) bundles of (typed) features, for example. The rules in 2.1, for example, are not so much informal versions of those in 2.2, rather the categories occurring in them are named after the values of different features appropriate for the categories *Root*, *Word*, or *Affix*. Selkirk's account of feature percolation from morphological heads (Selkirk, 1982, p. 74f.) could be formulated as a precise Head Feature Principle (cf. Section 5.2 in this thesis). Moreover, with a suitably large set of features, the category *Root* could certainly be done away with, as all restrictions could be expressed via subcategorisation of feature-value pairs.[8]

2.3.2. Deconstructing morphology

The aim of Lieber (1992) is to demonstrate that there are no independent rules or principles of morphology but that everything that usually falls under the heading of morphology can be described using the principles familiar from syntactic theory in the form of the GB framework Chomsky (1981). Lieber therefore explicitly opposes the *Lexicalist Hypothesis* introduced in Chomsky (1970) which states that the rules of morphology and the rules of syntax do not interact. The Lexicalist Hypothesis forms the background of major studies in generative morphology such as Selkirk (1982), and Di Sciullo and Williams (1987).[9]

The lexicon in Lieber (1992) is the inventory of so-called *listemes*, including the following:

- Affixes: e.g. *-ize*
- Roots: bound morphemes such as *path* (as in *psychopath, pathology*)
- Words: e.g. *run, enter*
- Lexicalised (complex) words: e.g. *transmission*

The following components are included in a lexical entry:

- the syntactic category
- a phonological representation
- the Lexical Conceptual Structure (LCS, after Jackendoff, 1990)
- the Predicate Argument Structure (PAS), giving the mapping between LCS and syntactic structure

Two sample lexical entries are given in Figures 3 and 4.

[8]Selkirk rejects such a "diacritic" analysis for English, though, cf. (Selkirk, 1982, p.112ff).

[9](cf. Lieber, 1992, p.12).

run	[V------]
	[r1n]
LCS:	$[_{\text{Event}}\ \text{GO}([_{\text{Thing}}\quad], [_{\text{Path}}\quad])]$
PAS:	x

Figure 3: Lexical Entry for *run* according to (Lieber, 1992, p.22).

-ize	[N,A------]V
	[ayz]
LCS:	$[\ \text{CAUSE}\ ([_{\text{Thing}}\quad], [\text{BE (LCS of base)}])]$
PAS:	x

Figure 4: Lexical Entry for *-ize* according to (Lieber, 1992, p.22).

Lieber gives the example of English possessive marking and phrasal compounding (i.e. not exactly the core cases) to illustrate the non-separability of grammar into a syntactic and a morphological component. In these, the morphological processes of affixation and compounding operate on syntactic structures, as in the following examples:

2.4 [*Never-come-back*]*-Airlines*

2.5 [*The Wife of Bath*]*'s (Tale)*

On prosodic and particularly on distributional and semantic grounds, we must come to the conclusion that in the first example, the VP *never come back* is compounded with the noun *airlines*, and in the second example, the possessive marker *-s* is suffixed to the NP *Wife of Bath.* In these examples, the stipulated boundary between the morphological component and the syntactic component of grammar is disregarded. Thus it seems to be doubtful whether there is such a partition of grammar at all. During the rest of her study, Lieber (1992) tries to prove step by step that the GB-principles of syntax work for morphology as well. The first of these principles is X-bar theory. The X-bar schema of syntax is slightly revised so as to fit both syntactic and morphological constructions:

2.6 $X^{n} \longrightarrow \ldots X^{n,n\text{-}1} \ldots$

The main innovation is that projections of X in the RHS of the schema need not longer be one bar-level lower than the projection of X in the LHS. The schema thus allows for direct recursion familiar from the X-bar schema devised by Selkirk (1982), cf. Section 2.3. Closely connected to X-bar theory is Head theory, as X is the head in the X-bar schema, occurring on both sides of the arrow in 2.6. Williams (1981) had proposed a theory of headedness and feature percolation in morphology, known as the *Right-hand*

Head Rule (RHR). This was was revised in Di Sciullo and Williams (1987) by introducing the notion of *relativised head*, for which the percolation principle subsequently became called the *Relativised Right-hand Head Rule (RHR).* Lieber, however, claims that heads in morphology are predictable using the head principles from syntax, which state that heads are either initial or final with respect to complements, specifiers, and modifiers, (see Section 5.2 for a definition of these) according to studies by Stowell (1990) and Travis (1990). The latter claims that the property of being head-initial or head-final is a parameter of universal grammar which must be set one way by each language.

Lieber claims that the Licensing Conditions for Heads in English syntax are the following (Lieber, 1992, p.49f.):

1. Heads are initial with respect to complements.
2. Heads are final with respect to specifiers.
3. Heads are final with respect to modifiers.

She claims that these conditions are met in English morphology as well. Her argumentation depends heavily on what is identified with complements, specifiers, and modifiers in the field of morphology, the account of which seems somewhat problematic, especially her claim that a stem in a derivation must be the specifier (p.54). Synthetic compounds such as *bus driver* are obvious counterexamples to the claim that Heads are initial with respect to complements in English word formation, as *bus* is the complement of *drive* in the example. For synthetic compounds, Lieber claims that on the level of D-structure, the construction of *bus driver* is indeed left-headed, and is only later reversed by the transformation *move-α*.

With further examples from other languages such as French and Tagalog (p.64ff) it is argued similarly that Head and X-bar theory work in morphology just as they do in syntax.

In this study, we do not want to adopt Lieber's line of argumentation, since she develops her theory within a transformational framework, whereas we aim at a description of morphology within a monostratal theory of grammar. Although Lieber provides independent motivations for ascribing the correct head directionality in synthetic compounds to D-structure, the recourse to D-structure seems arbitrary and to be done because a desired analysis is impossible on S-structure. We do not want to decide whether there are separate morphological and syntactic components of grammar at this point of the present thesis, instead we want to develop a description of the morphology of German within a unification based theory of grammar and to judge a posteriori whether it contains constraints that are to be viewed as exclusively morphological or not.

Nevertheless, Lieber's subsequent analysis of feature percolation and inheritance will be of great interest for the present thesis. In her analysis, she tries to make explicit what remains somewhat unclear in the studies of her predecessors, namely what components lexical entries have, what (kinds of) features there are, and how and why exactly they do or do not percolate. This can be regarded as an important step in the development of a Head Feature Principle for Morphology in the sense of HPSG.

Firstly, Lieber (p.80ff) distinguishes between *morphosyntactic* and *diacritic* features. Morphosyntactic features are those that play a role in syntax (in e.g. syntactic agreement) and are usually provided by inflectional morphology. In German these features would be syntactic category, case, number, gender, strong/weak adjective declension, person, and also tense and mood for verbs. It is argued that in morphological constructions, these features percolate from a head to its dominating constituent. Secondly, what it is exactly that percolates is a so-called *categorial signature* which can be interpreted as a well-typed feature structure containing the morphosyntactic specifications of a morphological construction, as in it it is precisely defined for each syntactic category what features it contains and what their possible values are.

$$\begin{bmatrix} \text{N} \\ +/- \ \text{Plural} \\ +/- \ \text{Case}_\text{i} \\ +/- \ \text{Case}_\text{j} \\ +/- \ \text{Fem} \\ +/- \ \text{Masc} \\ +/- \ \text{I} \\ +/- \ \text{II} \end{bmatrix}$$

Figure 5: Categorial signature for nouns after (Lieber, 1992, p.89)

Inflectional suffixes "do not carry complete categorial signatures", they just contribute value specifications to categorial signatures of the stems to which they are attached. Consequently, they cannot be heads.

> Only stems, bound bases, and derivational affixes will have full categorial signatures. Inflectional affixes will be marked only with individual features for which they contain specified values. In derivational word formation the value for a feature of a head morpheme will supersede or override that of an inner morpheme. Features from inflectional morphemes can never override features from their bases, but can only fill in values unspecified in the categorial signatures of their bases. Inflectional word formation is therefore **additive** in a way that derivational word formation is not. A corollary of this is that while derivational affixes may or may not be heads of their words, inflectional affixes will never be heads. (Lieber, 1992, p.112)

This is a nice anticipation of the distinction between the Head Complement Schema vs. the Head Marker Schema within HPSG (cf. Section 5.2.3).

The other kind of feature, diacritics, is mentioned by both Selkirk (1982) and Lieber (1992), but the term is never properly defined. Lieber cites [+/− strong] and [+/− latinate] as diacritic features, and from the examples we conclude that diacritics are those purely morphotactic features like paradigm features, which have no meaning for syntax at all. According to (Lieber, 1992, p.80f.), diacritic features never percolate, they are only referred to in morphological subcategorisation constraints.

3. Formal foundations of typed feature structures

In HPSG, the modelling domain for linguistic objects are *typed feature structures*, or *attribute-value structures.* In most introductions to feature-based theories of grammar, such as Pollard and Sag (1994), Kiss (1995), Witt and Müller (2002), the important distinction between the *modelling domain* (the set of feature structures) and the *description formalism* is made. To better clarify the status of AVMs, we make a three-fold distinction between the *modelling domain*, the *description formalism*, and its possible *notations*: Feature structures are structured mathematical objects, they are defined in terms of set and function theory (cf. Definitions 3.1 and 3.8). *Descriptions of feature structures* are provided in terms of logical formulas, i.e. a feature structure is described through a sentence in a formalism that has its own syntax and an interpretive semantics. The latter is a mapping of descriptions onto objects or sets of objects in the domain of feature structures. Finally, *AVMs* such as in Figure 6, or *DAG notations* such as in Figure 7 are different notations for feature structures or descriptions of feature structures. They contain graphic elements and are supposed to be more easy to read and write for humans than logical formulas. These notations stand in an isomorphic relationship to the descriptions of feature structures so that linguists can develop linguistic theories using the AVM notation without needing to know details about the logics of feature structures and their descriptions. Nevertheless reasoning is done, and proofs are conducted usually over descriptions of feature structures, and not over AVMs.

In the following, we introduce feature structures and possible relations between them as well as the crucial concepts of *types* and *inheritance hierarchies.* We follow Keller (1993) in the presentation of the logical formalism for the description of feature structures, which was originally introduced by Kasper and Rounds (1986). The formal foundations of typed feature systems were originally introduced by Carpenter (1992). The view of typed feature structures as total models of linguistic objects that is prevalent in current HPSG is based on King (1989).

3.1. Ordinary feature structures

Let L be a non-empty set of feature labels and A be a non-empty set of atomic values.

Definition 3.1 (Feature structure)[10] *A feature structure (over L and A) is a quadruple* $\langle Q, q_o, \delta, \pi \rangle$ *where*

- Q *is a non-empty, finite set of states*
- q_0 *is the root state*

[10]cf. (Keller, 1993, p.21).

- *$\delta : (Q \times L) \to Q$ is a partial transition function*
- *$\pi : Q \to A$ is a partial assignment function such that $\forall q \in Q, \forall l \in L$: if $\pi(q)$ is defined, then $\delta(q, l)$ is undefined.*

Additionally, a feature structure is required to be *connected*, i.e. every state $q \in Q$ must be *reachable* from q_0 via a sequence of transitions in δ.[11]

In this definition, terminology from automata theory is partly used (*state, transition function*, cf. (Hopcroft and Ullman, 1979, p.16f.)). We could just as well call the states *nodes*, and the transition function the set of *arcs*, as is done in (Carpenter, 1992, p.36).

$$\begin{bmatrix} \textsc{phon} & \textit{?Un} \\ \textsc{morph} & [\textsc{subcat Adj}] \\ \textsc{syn} & [\textsc{cat Adj}] \end{bmatrix}$$

Figure 6: Feature structure in AVM notation

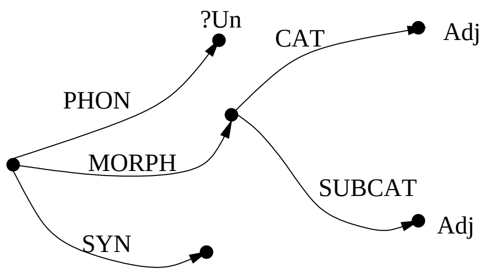


Figure 7: Feature structure in DAG notation

Example 3.2 (A feature structure) *Let* PHON, MORPH, SUBCAT, SYN, CAT $\in L$ *and* ?Un *and* ADJ *be atomic values in A. The AVM in Figure 6 represents a feature structure $\mathcal{A} = \langle Q, q_0, \delta, \pi \rangle$ where*

1. $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$
2. $\delta(q_0,\textsc{phon}) = q_1$
   $\delta(q_0,\textsc{morph}) = q_2$
   $\delta(q_0,\textsc{syn}) = q_3$
   $\delta(q_2,\textsc{subcat}) = q_4$
   $\delta(q_3,\textsc{cat}) = q_5$

[11]For a formal definition of reachability, see page 34.

3. $\pi(q_1) = ?Un$
$\pi(q_4) = \textsc{Adj}$
$\pi(q_5) = \textsc{Adj}$

The same feature structure is displayed in the alternative DAG notation in Figure 7. The transitions, labelled by feature structure labels, are easily recognisable. The 'final' states are labelled by the atoms. The names of the states cannot be found in the graph, but these are irrelevant anyway. Each single state is indicated by one (unnamed) node.

We would like to talk about a *path* and its *value* within a feature structure. A path is a sequence of labels that reflects a transition from one state to another:[12]

Definition 3.3 (Path, Value, reachable) *Let $\mathcal{A} = \langle Q, q_0, \delta, \pi \rangle$ be a feature structure, and let q be a state in Q, p be a sequence of labels in L^*. Define $\delta^* : (Q \times L^*) \to Q$ as follows:*

- *$\delta^*(q,p) = q$ if p is the empty path*
- *$\delta^*(q,p) = \delta^*(\delta(q,l),p')$ if $p = lp'$ is non-empty ($l \in L$ and $p' \in L^*$ are uniquely determined through p)*

A path *in $\mathcal{A}$ is thus an element in $Paths(\mathcal{A}) = \{p \in L^* \mid \exists\ \delta^*(q_0,p)\}$.*

Let q' be another state in Q. If $\delta^*(q,p) = q'$ for some path p then the state q' is reachable *from state q.*

The value *$\mathcal{V}(\mathcal{A},p)$ of a given path $p \in Paths(\mathcal{A})$ is defined as $\mathcal{V}(\mathcal{A},p) = \langle Q', \delta^*(q_0,p), \delta', \pi' \rangle$, where $Q' \subseteq Q$ is the subset of the states of $\mathcal{A}$ which are reachable from the new start state $\delta^*(q_0,p)$, and δ' and π' are the restrictions of δ and π respectively to states in Q'.*[13]

The value $\mathcal{V}(\mathcal{A},p)$ is thus again a feature structure. As notational device, we use '|' as a delimiter for labels in a path. Thus, the feature structure shown in Figure 6 contains the paths PHON, MORPH, MORPH|SUBCAT, SYN, and SYN|CAT. The value of the path MORPH is a feature structure represented by the AVM '[SUBCAT ADJ].' The value of MORPH|SUBCAT is the atomic symbol 'ADJ', which is considered a feature structure with only one state q_0, an empty δ, and with π being fully defined by the assignment $\pi(q_0) = \textsc{Adj}$.

By definition 3.1, feature structures are distinguished from *trees* such as the one in Figure 2 by the fact that the transitions (arcs, edges) are labelled. A second difference is that feature structures may be *re-entrant.* Informally, re-entrancy (or *structure sharing*) describes the existence of token-identical values for different paths in one feature

[12]cf. (Keller, 1993, p.23f).
[13]cf. (Keller, 1993, p.24).

$$\begin{bmatrix} \text{PHON} & \textit{/?Ungu:t/} \\ \text{DTRS} & \begin{bmatrix} \text{AFFX} & \begin{bmatrix}\text{SUBCAT} & \begin{bmatrix}\text{SCAT} & \boxed{1}\ \textsc{adj}\end{bmatrix}\end{bmatrix} \\ \text{BASE} & \begin{bmatrix}\text{SCAT} & \boxed{1}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 8: Example of AVM notation of a feature structure with token-identical values

structure. In AVM notation, this is marked through square boxes containing indices which are placed before the values and which are called *tags*. Identical tags denote token identity. Figure 8 contains such a re-entrancy, and, to emphasise token-identity, the respective value 'Adj' is indicated only once. Figure 8 displays a typical example of grammatical facts that are modelled by structure sharing: in a complex sign (one that has a DTRS attribute), the subcategorisation information of one of the daughters shares its structure with one of the other daughters present.

Formally, a feature structure $\mathcal{A}$ is re-entrant, if there exist $q, q_1, q_2 \in Q$ in $\mathcal{A}$, l_1, $l_2 \in L, l_1 \neq l_2$ such that $\delta(q_1, l_1) = q$ and $\delta(q_2, l_2) = q$. In other words, a feature structure is re-entrant if it contains a state q that can be reached through transitions via two different paths. The idea of one path with a branch that re-enters it at a particular node is illustrated best in the DAG notation in Figure 9 of the AVM example in Figure 8.

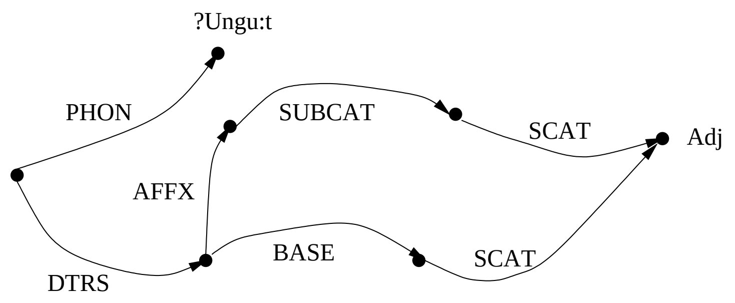


Figure 9: Re-entrant feature structure in DAG notation

One view of feature structures is to regard them as *partial* models of linguistic objects. If we have a feature structure $\mathcal{A}$ it will always be possible to find a feature structure that contains the information modelled in $\mathcal{A}$ but which contains more information in the form of additional feature-value specifications i.e. assignments for the function δ. In that case, $\mathcal{A}$ is said to *subsume* $\mathcal{B}$.

Definition 3.4 (Subsumption)[14]. *Let* $\mathcal{A} = \langle Q, q_0, \delta, \pi \rangle$ *and* $\mathcal{A}' = \langle Q', q_0', \delta', \pi' \rangle$ *be two feature structures. Then* $\mathcal{A} \sqsubseteq \mathcal{A}'$ *(*$\mathcal{A}$ *subsumes* $\mathcal{A}'$*,* $\mathcal{A}'$ *extends* $\mathcal{A}$*) just in case there exists a mapping* $h : Q \rightarrow Q'$ *which meets the following conditions:*

[14](Keller, 1993, p.24f.)

- $h(q_0) = q_0'$
- *if* $\delta(q,l)$ *is defined, then* $\delta'(h(q),l) = h(\delta(q,l))$
- *if* $\pi(q)$ *is defined, then* $\pi'(h(q)) = \pi(q)$

In definition 3.4, h is a homomorphism from Q to Q' under which the structure defined over L and A through δ and π is preserved for Q'. It is not excluded that δ' and π' contain additional mapping assignments to those required by h, and this is what constitutes the intuitive sense of feature structure subsumption: $\mathcal{A}'$ is more informative than $\mathcal{A}$.

Two feature structures, $\mathcal{A}$ and $\mathcal{B}$ are (subsumption-)equivalent, if $\mathcal{A} \sqsubseteq \mathcal{B}$ and $\mathcal{B} \sqsubseteq \mathcal{A}$; they are *incomparable* if neither $\mathcal{A} \sqsubseteq \mathcal{B}$ nor $\mathcal{B} \sqsubseteq \mathcal{A}$.

(Keller, 1993, p.25) points out that equivalent feature structures are not necessarily identical, as they may have differently named states. For some considerations it is therefore necessary to deal with equivalence classes of feature structures, it is e.g. the domain of equivalence classes of feature structures on which a weak partial order is induced through '$\sqsubseteq$'.

To introduce the operation of *unification* on the set of feature structures, we first define the concepts of feature structure *compatibility* and *minimal unifiers*: Two feature structures $\mathcal{A}$ and $\mathcal{A}'$ are *compatible* just in case $\exists\mathcal{B} : \mathcal{A} \sqsubseteq \mathcal{B}$ and $\mathcal{A}' \sqsubseteq \mathcal{B}$. $\mathcal{B}$ is called a *unifier* of $\mathcal{A}$ and $\mathcal{A}'$. $\mathcal{B}$ is called *minimal* just in case if $\mathcal{B}'$ is another unifier then $\mathcal{B} \sqsubseteq \mathcal{B}'$.

Definition 3.5 (Unification) *Let* $\mathcal{A}, \mathcal{A}'$ *be feature structures. Define*

- $\mathcal{A} \sqcup \mathcal{A}' = \mathcal{B}$ *if* $\mathcal{A}$ *and* $\mathcal{A}'$ *are compatible such that* $\mathcal{B}$ *is their minimal unifier*

The unification of two (compatible) feature structures $\mathcal{A} \sqcup \mathcal{A}'$ is thus a feature structure that combines the pieces of information contained in $\mathcal{A}$ and those in $\mathcal{A}'$. In HPSG, unification of feature structure occurs when two signs are combined to form a complex sign. This is a situation such as represented in the AVM in Figure 8. The two feature structures that are the value of DTRS|AFFIX and DTRS|BASE are originally two different signs. The values of DTRS|AFFIX|SUBCAT|SCAT and DTRS|BASE|SCAT are unified in Figure 8, which is indicated through the tag $\boxed{1}$.

Finally we want to point out that Definition 3.1 does not exclude feature structures that contain cycles. A feature structure $\mathcal{A}$ is cyclic if it contains a non-empty path p and a state q such that $\delta^*(q,p) = q$, i.e. a path that leads to its own begin state.

(Karttunen, 1984, p.24) remarked on cyclic feature structures that "it has not been shown, that there are phenomena in natural languages that involve circular structures". But the semantics of one reading of the sentence in Figure 10 could be modelled by a feature structure such as represented by the AVM in Figure 10, cf. (Carpenter, 1992, p.38).

$$\boxed{1}\begin{bmatrix}\text{REL} & \textsc{false}\\ \text{ARG} & \boxed{1}\end{bmatrix}$$

Figure 10: Example of an AVM representing a cyclic feature structure

$$\begin{bmatrix}\text{AGR} \neg \begin{bmatrix}\text{PER} & 3\\ \text{NUM} & \textsc{sg}\end{bmatrix}\end{bmatrix}$$

Figure 11: AVM containing a negative constraint

3.6 This sentence is false.

In this thesis, we employ cyclic feature structures in the description of the kind of mutual subcategorisation between umlauted bases and umlauting suffixes, cf. Section 5.5.4.

3.2. Feature logics

Why have a distinction between feature structures and descriptions of feature structures at all? Feature structures, as defined above have the advantage that they can be defined in terms of basic mathematical concepts, i.e. sets and ordered tuples (relations and functions). On the other hand, according to (Carpenter, 1992, p.51), the logical language of descriptions provides "a way to talk about feature structures; the language can be displayed linearly one symbol after another and can thus be easily used in implementations." Moreover, a description language is a kind of metalanguage that allows us to formulate negative and disjunctive constraints on feature structures (the information contained in feature structures as defined so far is *conjunctive* in the sense in which sets are *col*-lections of objects). A description may thus denote classes of feature structures, apart from particular feature structures. This is definitely desirable when describing models of linguistic objects, as is illustrated by the examples in the AVMs representing descriptions of feature structures in Figures 11 and 12. The AVM in Figure 11 describes the class of feature structures that contain agreement information other than *3rd person singular*. It could thus be the specification of the agreement information provided by the English verb form *drive* (as opposed to *drives*) which is a single word form standing for different grammatical words (cf. Section 5). The AVM in Figure 12 could represent a description of the agreement information of the German article *die* which can be alter-

$$\begin{bmatrix}\text{AGR} \begin{bmatrix}\text{GEN} & \textsc{fem}\\ \text{NUM} & \textsc{sg}\end{bmatrix} \vee [\text{NUM } \textsc{pl}]\end{bmatrix}$$

Figure 12: AVM containing a disjunctive constraint

natively *plural* or *feminine singular*. In these examples, it is the domain of grammatical words that is modelled by feature structures.

The characteristics of feature structure descriptions are defined in the language of Rounds-Kasper logic (Kasper and Rounds (1986)). Here we introduce a slightly modified version of the syntax presented in (Keller, 1993, p.26). A well-formed formula or description with respect to the set of labels L and the set of atoms A is defined as follows:[15]

- $\top$ is a description
- $\bot$ is a description
- if $a \in A$, then a is a description
- if $l \in L$ and ϕ is a description, then $l : \phi$ is a description
- if ϕ and ψ are descriptions, then $(\phi \wedge \psi)$ is a description
- if ϕ and ψ are descriptions, then $(\phi \vee \psi)$ is a description
- if $p_1, p_2 \in L^*$, then $p_1 \doteq p_2$ is a description

In a description, the operator ':' has higher precedence than '$\vee$' and '$\wedge$', and brackets are omitted when this causes no ambiguities. $\top$ is the trivial description (satisfied by any feature structure), and $\bot$ is the 'inconsistent description' (which is never satisfiable). A path equation $p_1 \doteq p_2$ states that the values of the paths p_1 and p_2 are token-identical.

Example 3.7 (A feature structure description) *Let* PHON, DTRS, AFFIX, SUBCAT, SCAT, BASE *be feature labels* $\in$ *L*, 'ADJ', '/?Ungu:t/' *be atomic feature values* $\in$ *A. The following formulas are well-formed descriptions:*

1. (PHON: */?Ungu:t/* $\wedge$
   (DTRS: (AFFX:SUBCAT:SCAT: ADJ $\wedge$ BASE:SCAT: ADJ)))
2. (DTRS:AFFX:SUBCAT:SCAT $\doteq$ DTRS:BASE:SCAT)

Both descriptions are *satisfied* by the feature structure represented by the DAG in Figure 9. Formally, a feature structure $\mathcal{A} = \langle Q, q_o, \delta, \pi \rangle$ satisfies a description ϕ ($\mathcal{A} \models \phi$) under the following conditions:[16]

- $\mathcal{A} \models \top$
- $\mathcal{A} \not\models \bot$
- $\mathcal{A} \models a \Leftrightarrow \pi(q_o) = a$

[15]cf. also (Carpenter, 1992, p.52).

[16](Keller, 1993, p. 26).

- $\mathcal{A} \models (l : \phi) \Leftrightarrow \mathcal{V}(\mathcal{A}, l)$ is defined and $\mathcal{V}(\mathcal{A}, l) \models \phi$
- $\mathcal{A} \models (\phi \vee \psi) \Leftrightarrow \mathcal{A} \models \phi$ or $\mathcal{A} \models \psi$
- $\mathcal{A} \models (\phi \wedge \psi) \Leftrightarrow \mathcal{A} \models \phi$ and $\mathcal{A} \models \psi$
- $\mathcal{A} \models (p_1 \doteq p_2) \Leftrightarrow \delta^*(q_0, p_1) = \delta^*(q_o, p_2)$

The satisfaction relation '$\models$' is monotonic with respect to the subsumption relation. That is, if $\mathcal{A} \models \phi$ and $\mathcal{A} \sqsubseteq \mathcal{B}$, then $\mathcal{B} \models \phi$. Via the interpretation of feature structure descriptions, it also follows that if $\mathcal{A}$ and $\mathcal{B}$ are compatible and if $\mathcal{A} \models \phi$ and $\mathcal{B} \models \psi$ then $\mathcal{A} \sqcup \mathcal{B} \models (\phi \wedge \psi)$, i.e. feature structure unification corresponds to the conjunction of descriptions.

A whole calculus for drawing inferences over feature structure descriptions can be defined, which is based on the proofs for equivalences that hold between expressions of the Rounds-Kasper language, cf. (Keller, 1993, p.28f.). Algorithms for the unification of feature structure descriptions (including disjunctive descriptions) are based on such a calculus, cf. Kasper (1987), and Eisele and Dörre (1988).

In the definition of feature structure descriptions given above, the negation operation occurring in Figure 11 is not included. This is because the interpretation of the negation of feature structure descriptions poses difficulties, i.e. it is non-trivial to define the satisfaction relation for negated descriptions without satisfaction becoming non-monotonic with respect to subsumption. Monotonicity is a desirable quality of the calculus for reasons of algorithmic efficiency, though. Several solutions have been put forward for defining a suitable interpretation for negative descriptions, (cf. Keller, 1993, p.31ff). One of them lies in the extension of the modelling domain with a system of types and feature appropriateness definitions, which is desirable for several other reasons and has thus been incorporated into the HPSG framework. With the help of these *typed feature structures,* it is possible to draw the linguistically relevant distinction between a feature-value specification that is unknown in a description and an altogether irrelevant feature.

3.3. Typed feature structures

A typed feature structure is distinguished from an ordinary feature structure through the specification of a type symbol denoting a type in a type system. Types serve "to organise feature structures into natural classes" (Carpenter, 1992, p.11). More precisely, the type associated with a feature structure states what feature labels are allowed to occur in the feature structure in so-called *appropriateness conditions.* A feature structure containing feature labels that are not appropriate for it is said to be not *well-typed.*

A type system is simply a set *Type* of type symbols partially ordered by subsumption. As types are atomic, the subsumption relation may be defined extensionally for *Type*.

For instance, let $sign, phrasal\text{-}sign, lexical\text{-}sign, word, morpheme \in \mathsf{Type}$, and define the subsumption relation as follows:

- $sign \sqsubseteq lexical\text{-}sign$
- $sign \sqsubseteq phrasal\text{-}sign$
- $lexical\text{-}sign \sqsubseteq word$
- $sign \sqsubseteq word$
- $lexical\text{-}sign \sqsubseteq morpheme$
- $sign \sqsubseteq morpheme$
- $lexical\text{-}sign \sqsubseteq lexical\text{-}sign$
- $phrasal\text{-}sign \sqsubseteq phrasal\text{-}sign$
- $sign \sqsubseteq sign$
- $word \sqsubseteq word$
- $morpheme \sqsubseteq morpheme$

We say that *sign* subsumes *lexical-sign*, that *sign* is a supertype of *lexical-sign*, that *lexical-sign* inherits from *sign*, and that *lexical-sign* is a subtype of *sign.* A type system is alternatively called an *inheritance hierarchy* (of types). The inheritance hierarchy above may be depicted as a graph (Figure 13).

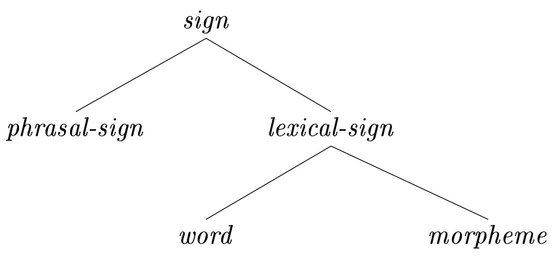


Figure 13: A simple type hierarchy

(Carpenter, 1992, p.11) remarks that inheritance specifications (as well as the links in Figure 13) resemble the *ISA* links in knowledge representation networks such as KL-ONE (Brachman and Schmolze (1985)). More precisely, "the full subsumption relation can be inferred as the transitive and reflexive closure of the relation determined by the ISA links" (Carpenter, 1992, p.12).

An inheritance hierarchy Type is called *consistent* if its types share a common subtype σ, an *upper bound* with respect to subsumption, i.e. for every $\tau \in \mathsf{Type}$, $\tau \sqsubseteq \sigma$ holds. σ is called the *least upper bound* or *join* of Type if additionally $\sigma \sqsubseteq \rho$ for every upper bound $\rho \in \mathsf{Type}$. It is required that every consistent subset of an inheritance hierarchy have a join. The join of the empty set (which is a consistent subset of Type) is defined as $\bot$, which is thus required to be an element of any type hierarchy.

The join of a set of types with two elements α, β is written $\alpha \sqcup \beta$.

The definition of feature structures is extended to *typed feature structures* by associating each feature structure with a type in *Type*:

Let L be a non-empty set of feature labels and *Type* be an inheritance hierarchy.

Definition 3.8 (Typed feature structure)[17] *A typed feature structure over* L *and* Type *is a quadruple* $\langle Q, q_0, \delta, \theta \rangle$ *where*

- Q *is a non-empty, finite set of states that are reachable from* q_0
- q_0 *is the root state*
- $\delta : (Q \times L) \to Q$ *is a partial* transition function
- $\theta : Q \to$ Type *is a total function from states to types, the* type assignment*, or* typing function

In comparison with the definition of ordinary feature structures given in Figure 3.1, the type assignment function θ replaces the function π which assigned atomic values to states. Moreover, θ is a total function, assigning *every* state in a typed feature structure a type. But the notion of atomic values can be preserved by identifying atomic values with the lowest types in the inheritance hierarchy, i.e. those that have no subtypes (other than themselves) and no appropriate feature labels (cf. Pollard and Sag, 1994, p.19). An example of the description of a typed feature structure in AVM notation representing the derivational suffix *-keit* is given in Figure 15. It is associated with the type *keit*, and the values of all features contained are typed, too. The value of MORPH|DER|HEAD|LINK, for example, is typed to be *link*.

Figure 14 shows a feature structure in DAG notation, whose nodes are labelled by *types* and whose arcs are labelled by attributes.

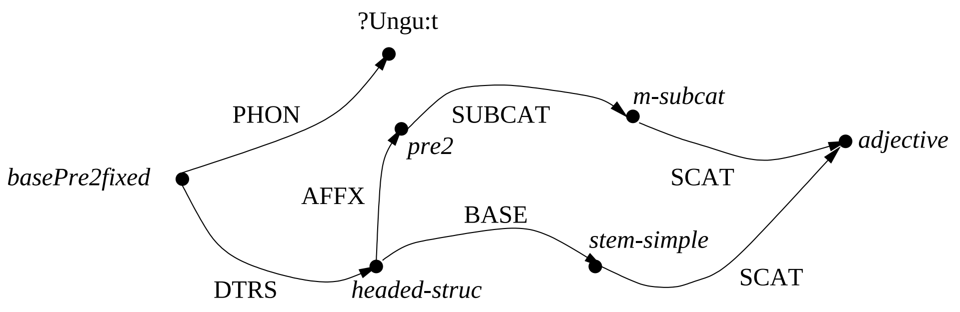


Figure 14: DAG notation of a typed feature structure with types as nodes and feature labels as arcs

[17]cf. (Keller, 1993, p. 36), (Carpenter, 1992, p.36).

$$
{}_{\textit{native-n-suffix}}\begin{bmatrix}
\text{PHON} & \textsc{kaIt} \\
\text{ORTH} & \textsc{keit} \\
\text{MORPH} & {}_{\textit{n-morph}}\begin{bmatrix}
\text{HEAD} & {}_{\textit{n-morph-head}}\begin{bmatrix}
\text{FLEX} & {}_{\textit{n-flex}}\begin{bmatrix}\text{SUFFIXCLASS} & \textit{noun-frau}\end{bmatrix} \\
\text{DER} & {}_{\textit{n-der}}\begin{bmatrix}\text{NAT} & + \\ \text{COMB} & + \\ \text{INTERF} & \{\,\}\end{bmatrix} \\
\text{UML} & {}_{\textit{uml}}\begin{bmatrix}\text{UMLB} & \bot\end{bmatrix} \\
\text{LINK} & {}_{\textit{link}}\begin{bmatrix}\text{LINKED} & - \\ \text{LINKBL} & \textit{s-lm-n}\end{bmatrix}
\end{bmatrix} \\
\text{MPHON} & {}_{\textit{suffix-mphon}}\begin{bmatrix}
\text{RBOUND} & \begin{bmatrix}\text{BND} & \textit{boundness} \\ \text{BND-ARG} & \textit{sufn} \sqcap \textit{infl} \sqcap \textit{stem-initial}\end{bmatrix} \\
\text{LBOUND} & \begin{bmatrix}\text{BND} & \textit{toBind} \\ \text{BND-ARG} & \textit{base-final}\end{bmatrix} \\
\text{ADJ} & - \\
\text{STR} & -
\end{bmatrix} \\
\text{SUBCAT} & {}_{\textit{non-uml-a-stem}}\begin{bmatrix}
\text{MORPH} & \begin{bmatrix}\text{DER} & [\text{NAT} \; -] \\ \text{UML} & [\text{UMLD} \; -] \\ \vee\ \text{UML} & [\text{UMLB} \; \bot]\end{bmatrix} \\
\text{SYN} & [\text{LOC}\ [\text{HEAD}\ [\text{MAJ}\ \textit{adj}]]]
\end{bmatrix}
\end{bmatrix} \\
\text{SYN} & \begin{bmatrix}
\text{LOC} & {}_{\textit{loc}}\begin{bmatrix}
\text{LEX} & - \\
\text{HEAD} & {}_{\textit{n-syn-head}}\begin{bmatrix}
\text{MAJ} & \textit{noun} \\
\text{GEN} & \textit{fem} \\
\text{CASE} & \textit{case} \\
\text{AGR} & \begin{bmatrix}\text{PER} & 3 \\ \text{NUM} & \textit{sing}\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

Figure 15: AVM for the suffix *keit*

The definition of the subsumption relation for typed feature structures is an extension of the subsumption definition for ordinary feature structures given in 3.4, taking into account the corresponding subsumption of associated types, expressed in the third condition below.

Definition 3.9 (Subsumption of typed feature structures) *Let $\mathcal{A} = \langle Q, q_0, \delta, \theta \rangle$ and $\mathcal{A}'$ $= \langle Q', q_0', \delta', \theta' \rangle$ be two typed feature structures over L and* Type. *Then $\mathcal{A} \sqsubseteq \mathcal{A}'$ ($\mathcal{A}$ subsumes $\mathcal{A}'$, $\mathcal{A}'$ extends $\mathcal{A}$) just in case there exists a mapping $h : Q \rightarrow Q'$ which meets the following conditions:*

- $h(q_0) = q_0'$
- *if $\delta(q, l)$ is defined, then $\delta'(h(q), l) = h(\delta(q, l))$*
- $\forall q \in Q : \theta(q) \sqsubseteq \theta'(h(q))$

In HPSG, where linguistic objects are modelled using typed feature structures, the idea is that every typed feature structure must be *well-typed*, which comprises the notion of *appropriateness* of feature labels for types. For example, a feature structure modelling a noun may contain, or lack, a feature value specification for CASE, but with respect to a verb it is known in advance that a CASE specification will never be appropriate. Similarly, we want to guarantee that the value type *past* cannot be assigned to the feature label CASE in a typed feature structure. (Carpenter, 1992, p.86) provides a formal definition of feature appropriateness, which is also adopted in (Keller, 1993, p.37):

Definition 3.10 (Appropriateness) *An appropriateness specification is a partial function* $Appropriate : L \times \mathsf{Type} \to \mathsf{Type}$ *from label-type pairs to types which meets the following conditions:*

- *(Minimal introduction)* $\forall l \in L$ *: there is a most general type* $\sigma \in \mathsf{Type}$ *such that* $Appropriate(l, \sigma)$ *is defined*
- *(Upward closure/Right monotonicity) If* $Appropriate(l, \sigma)$ *is defined and* $\sigma \sqsubseteq \tau$ *then* $Appropriate(l, \tau)$ *is defined an* $Appropriate(l, \sigma) \sqsubseteq Appropriate(l, \tau)$

Minimal introduction requires that whenever a feature label is appropriate for two types that have a common supertype, this feature must also be appropriate for that supertype. The *upward closure* property says that a feature label appropriate for a given type is also appropriate for all of its subtypes. Finally, the value of a feature label l appropriate for a type τ must be subsumed by the value of l for any supertype σ of τ at which l is defined (*right monotonicity*).

Now we have prepared the ground for formally introducing the notion of a well-typed feature structure:

Definition 3.11 (Well-typed feature structure) *A typed feature structure* $\mathcal{A} = \langle Q, q_0, \delta, \theta \rangle$ *is* well-typed *if whenever* $\delta(q, l)$ *is defined,* $Appropriate(l, \theta(q))$ *is defined and* $Appropriate(l, \theta(q)) \sqsubseteq \theta(\delta(l, q))$.

In other words, any feature-value specification in a well-typed feature structure must be appropriate for the type of that feature structure. In HPSG, the reversal of this requirement must also hold: Whenever a feature is appropriate for a given type, each feature structure that is assigned this type must actually contain a value specification for that feature. Formally, a typed feature structure $\mathcal{A} = \langle Q, q_0, \delta, \theta \rangle$ is *totally well-typed* if it is well-typed, and if $\delta(q)$ is defined in $\mathcal{A}$ if for $l \in L$ and $q \in Q$, $Appropriate(l, \theta(q))$ is defined.

This implies that for every type an appropriate specification exists that tells us what features are appropriate for the type, and of what type the values for these features have to be. (Carpenter, 1992, p.77) remarks that "our appropriate specifications carry out the task originally delegated to co-occurrence restrictions in GPSG". These appropriateness specifications are also called a *feature declarations.*

Feature declarations may be represented in AVM notation. Figure 16 depicts the feature declarations for the type *native-n-suffix.*

$$
{}_{native\text{-}n\text{-}suffix}\left[\begin{array}{ll}
\text{SURF} & {}_{surf}\left[\begin{array}{ll}\text{PHON} & \textit{phon-string}\\ \text{ORTH} & \textit{orth-string}\end{array}\right]\\
\text{MORPH} & {}_{n\text{-}morphology}\left[\begin{array}{ll}
\text{HEAD} & {}_{n\text{-}morph\text{-}head}\left[\begin{array}{ll}
\text{FLEX} & {}_{n\text{-}flex}[\text{SUFFIXCLASS}\ \textit{n-suffixclass}]\\
\text{DER} & {}_{n\text{-}der}\left[\begin{array}{ll}\text{NAT} & +\\ \text{COMB} & \textit{boolean}\\ \text{INTERF} & \textsc{set-of}(\textit{interfix})\end{array}\right]\\
\text{UML} & {}_{uml}\left[\begin{array}{ll}\text{UMLD} & \textit{boolean}\\ \text{UMLB} & \textit{suffix}\end{array}\right]\\
\text{LINK} & {}_{link}\left[\begin{array}{ll}\text{LINKED} & \textit{boolean}\\ \text{LINKBL} & \textsc{set-of}(\textit{lm})\end{array}\right]
\end{array}\right]\\
\text{MPHON} & {}_{suffix\text{-}mphon}\left[\begin{array}{ll}
\text{RBOUND} & \left[\begin{array}{ll}\text{BND} & \textit{boundness}\\ \text{BND-ARG} & \textit{sufn} \sqcap \textit{infl} \sqcap \textit{stem-initial}\end{array}\right]\\
\text{LBOUND} & \left[\begin{array}{ll}\text{BND} & \textit{toBind}\\ \text{BND-ARG} & \textit{base-final}\end{array}\right]\\
\text{ADJ} & \textit{boolean}\\
\text{STR} & -
\end{array}\right]\\
\text{SUBCAT} & {}_{base}[\text{SYN}\ {}_{syn}[\text{LOC}\ {}_{local}[\text{HEAD}\ \textit{syn-head}]]]
\end{array}\right]\\
\text{SYN} & \left[\begin{array}{ll}\text{LOC} & {}_{local}\left[\begin{array}{ll}\text{LEX} & -\\ \text{HEAD} & {}_{n\text{-}syn\text{-}head}\left[\begin{array}{ll}\text{MAJ} & \textit{noun}\\ \text{GEN} & \textit{gender}\\ \text{CASE} & \textit{case}\\ \text{AGR} & {}_{agr}\left[\begin{array}{ll}\text{PER} & \textit{person}\\ \text{NUM} & \textit{number}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]\ \textsc{syn}
\end{array}\right]
$$

Figure 16: AVM for type *native-n-suffix*

With the new definition of the subsumption relation for typed feature structures in Definition 3.9, the definition of unification for typed feature structures remains as it was given in Definition 3.5. It turns out that the type of the unification of two typed feature structures $\mathcal{A}$ and $\mathcal{B}$ (written $\mathcal{A} \sqcup \mathcal{B}$) which are associated with the types τ and σ, respectively, equals the join of τ and σ (written $\tau \sqcup \sigma$), cf. (Carpenter, 1992, p.47).

As to the semantics of typed feature structures, type symbols have to be accommodated in the definition of the satisfaction relation for ordinary feature structures given on page 38. The conditions under which a typed feature structure $\mathcal{A} = \langle Q, q_0, \delta, \theta \rangle$ satisfies a type σ are defined as follows:[18]

- $\mathcal{A} \models \sigma \Leftrightarrow \sigma \sqsubseteq \theta(q_0)$

(Carpenter, 1992, p.53ff) provides a calculus of equivalences which characterise the behaviour of the description language for typed feature structures. The descriptions of the totally well-typed feature structures are *partial* which permits amongst other things an inference from a type to appropriate feature labels and from the existence of a feature label to an appropriate value for that feature. That is, since all feature structures are totally well-typed, some features may be omitted in a description of a feature structure, for example, when they are not relevant for a point under discussion. In Figure 15, the value of MORPH|SUBCAT actually has a lot more features than only SYN, but since it is marked for the type *base*, the features not represented can be recovered by looking at the appropriateness specification for *base* and all of its supertypes.

Finally, the feature structures employed in HPSG are supposed to be *sort-resolved*, that is all the types occurring in them have to be maximal (most specific) with respect to the type hierarchy. In the feature structure for the suffix *keit* (Figure 15), for example, the value of MORPH|HEAD|FLEX is typed to *n-flex*, and not to *flex* as in Figure 16. If we had used *flex* in Figure 15, we would have falsely asserted that a feature structure of type *v-flex*, which is another subtype of *flex*, were also an admissible value of HEAD|HEAD|FLEX, which it is not. Thus, sort-resolving requires the type of the value of MORPH|HEAD|HEAD to be of type *n-flex* for the suffix *keit*.

[18]cf. (Keller, 1993, p.38), (Carpenter, 1992, p.53).

4. HPSG approaches to morphology

4.1. Empirical domain: The structure of the sign

A fundamental linguistic entity modelled by feature structures in HPSG is the *sign.* In Pollard and Sag (1994), a sign is modelled as a feature structure with the attributes PHON (phonology) and SYNSEM (syntax/semantics), representing the property of having both a form and a meaning. Thus, in HPSG, a two-dimensional, saussurean sign concept is proposed. In comparison with the earlier Pollard and Sag (1987), where a sign was a feature structure with three attributes, PHON, SYN, and SEM, the structure of a *sign* has been revised.

$$
{}_{\textit{sign}}\begin{bmatrix} \text{PHON} & \textsc{list-of}(\textit{phon-string}) \\ \text{SYN} & \textit{syn} \\ \text{SEM} & \textit{sem} \end{bmatrix}
$$

Figure 17: AVM for type *sign* after Pollard and Sag (1987)

$$
{}_{\textit{sign}}\begin{bmatrix} \text{PHON} & \textsc{list-of}(\textit{phon-string}) \\ \text{SYNSEM} & \textit{synsem} \end{bmatrix}
$$

Figure 18: AVM for type *sign* after Pollard and Sag (1994)

The collapsing of SYN and SEM into SYNSEM has been undertaken apparently for two reasons: Firstly, to emphasize the close dependencies between the syntax and semantics of a sign, which, unlike in transformational theories of grammar, can be represented in parallel on one representational level, i.e. within one feature structure. The second reason is more technical: In (Pollard and Sag, 1987, p.143f.), a language-universal *locality principle* was formulated, which constrained the theory of subcategorisation over feature structures. It said that "*the* SUBCAT *elements of lexical signs specify values for* SYNTAX *and* SEMANTICS *but crucially not the attribute* DAUGHTERS". In other words, a lexical sign may constrain features of types of signs it combines with, but never features of constituents of these. In the revised version, subcategorisation is no longer for lists of feature structures of type *sign*, but for lists of feature structures of type *synsem* (which is the type of the value of the attribute SYNSEM of *sign*). Thereby a subcategorisation for DAUGHTERS is automatically excluded, the locality of subcategorisation follows from the structure of the sign, it no longer needs to be explicitly formulated. Unfortunately, this way constraints on the PHON value of a subcategorised element can no longer be formulated, either; this issue will be discussed from the viewpoint of morphology in Section 5.

4.2. Lexical type hierarchies

There seems to be some inconsistency in the HPSG literature as to the use of the terms *type* and *sort*. (Kiss, 1995, p.44f.) makes an explicit distinction between them and defines *sorts* as elements of the description language (i.e. the italic labels in AVMs), whereas he defines *types* to be elements of feature structures. As was explained above, feature structures are total objects, which means that the type of a feature structure cannot have any subtypes, and a hierarchy cannot be constructed over types. Only the maximal (most specific) sorts in a sort hierarchy correspond to types. Thus, in the hierarchy developed in Section 5, only the sorts directly above the morph entries (i.e. *pre1, pre2, part, prenn* etc.) would correspond to types according to Kiss (1995).

Pollard and Sag (1987) used the term *type* for the labels in feature structures and switched to *sort* in Pollard and Sag (1994). Still, many current authors use *type* when talking about types as they are explained above, especially a hierarchy over entries in the lexicon is usually called a *lexical type hierarchy*, notably in Sag (1997), Riehemann (1998), Sag and Wasow (1999), Müller (1999), and not a *lexical sort hierarchy.* In view of this, we will refrain from making a distinction between *sort* and *type* and usually use *type* (except when talking about sort-resolvedness, as the term *type-resolved* is not found in the literature). We intend a *type* to be a part of a feature structure, and *type symbols* to represent them in AVMs. The type symbol may be found on the left of the AVM outside the outer square brackets and justified with the bottom of the AVM, or, alternatively, on the first line of an AVM close to and inside the leftmost square bracket.[19] It is always written in italics. We will call a lexical type in the sense of Kiss (1995) (a maximal subsort) a *maximal (lexical) type.*

As we have seen above, types are used to define appropriateness conditions on feature structures, but also to define classes of feature structures. In particular, types and type hierarchies are employed to structure the lexicon in HPSG. Words are grouped together according to certain properties they have in common. Common properties are encoded as feature specifications and associated with a particular type, and the actual lexical items inherit their feature specifications from such a type. This way, redundancy is removed from the lexical representations, as ideally each property (attribute-value specification) needs to be stated only once in the whole hierarchy (cf. Flickinger, 1987, p.7).

In Section 5 we introduce a set of maximal types serving to model lexical entries for German morphs and morphemes, one of which was displayed in Figure 16. The maximal types are the types right above the dotted arcs in Figure 65. The central question when constructing a type hierarchy over a set of maximal lexical types with feature appropriateness specifications is how should types be grouped to common supertypes, or conversely, how should a type be divided into subtypes. There are two possibilities: Firstly, a type T may be divided into the subtypes $T_1, T_2, ..., T_n$ according to different

[19]Due to certain Latex typesetting constraints, we make use of both notations in this thesis.

value types of one or more of the attributes appropriate for it. Such an attribute or set of attributes may then be called a *dimension* and the resulting subtypes are said to *partition* the type T, since they stand in an exclusive disjunction relation to each other and exhaust T, i.e. an instance of type T must either be of type T_1, $T_2, \ldots$, or of type T_n. Often, a type can be partitioned along several dimensions at once, leading to multiple orthogonal inheritance. That is, if a type T can be partitioned along the two dimensions D_1 and D_2, these partitions lead to subtypes $T_1, \ldots, T_n$ (by D_1) and $S_1, \ldots, S_n$ (by D_2). The set of maximal types under this hierarchy will then be a subset of the set of all joins $\{T_i \sqcup S_j\}$ where $i, j \in \{1, \ldots, n\}$. The type-partitioning dimensions $D_1, D_2, \ldots$ are often notated as nodes in the subsumption graph (though they are not types themselves) and set in square boxes. An example of this is given in figure 19 where two type-dividing dimensions, MAJOR and NAT, lead to 3×2 possible maximal types such as *native-noun*, which corresponds to the join *noun* $\sqcup$ *native.*

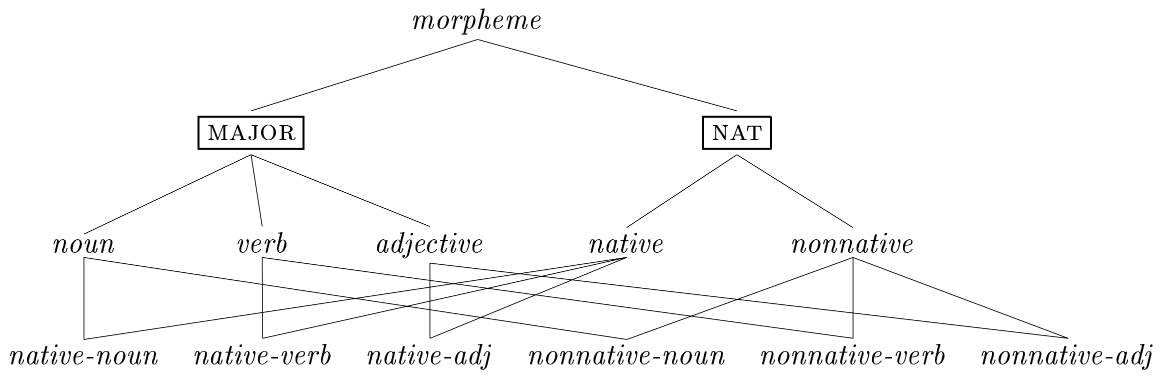


Figure 19: Partial type hierarchy for *morpheme*

A second possibility of type dividing is the following: The types $T_1, T_2, \ldots$ and T_n may constitute subtypes of one type T, if different sets of attributes are appropriate for them in addition to the common set of attributes appropriate for type T. In Figure 20, the *d(erivational)-affix* is a subtype of *morph* because in addition to the feature specifications of *morph*, the feature MORPH|SUBCAT is appropriate for it. Thus, the partitioning of the type *morpheme* into *root* and *d-affix* is along the criterion of absence vs. presence of the attribute MORPH|SUBCAT. (Since the absence of a feature may alternatively be encoded by a special value of that feature, the second kind of type dividing can always be converted into a type division of the first kind, cf. Sporleder (1999))

Both principles of subdividing a type may occur jointly and multiply; in most cases there will be one outstanding feature which suffices as the type-discriminating dimension but whose values may automatically entail type restrictions on some other features and the introduction of several new features. The type of the value of MORPH|HEAD, for example, is always covarying with the (atomic) value of SYN|LOC|MAJ.[20] A dimension

[20]For remarks on the principles of subdividing types and constructing type hierarchies in the HPSG

need thus not necessarily correspond to one single feature, but may be a conjunction of several feature value specifications, as exemplified in Krieger's (1993) affix hierarchy (see Section 4.4).

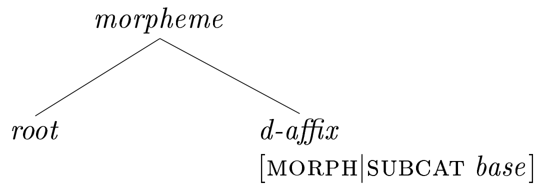


Figure 20: Another partial type hierarchy for *morpheme*

In Figure 65, we provide a more elaborate version of a morph type hierarchy with the types motivated linguistically in this thesis. In this hierarchy, subtypes introduce constraints in the fashion described above. The hierarchy is not totally free of redundancy since multiple inheritance was not employed *wherever* possible, (an inheritance hierarchy of feature structures is considered to be totally free of redundancy if each feature-value specification occurs exactly once (cf. Flickinger, 1987, p.7), as it would have become impossible to depict such a hierarchy in a decipherable graph.

4.3. Lexical rules

A second device that has been employed in HPSG to remove redundancy from the lexicon are *lexical rules* (Pollard and Sag (1987); Flickinger (1987); Copestake and Briscoe (1996); Sag and Wasow (1999); Müller (1999); Briscoe and Copestake (1999)). Whereas lexical types are used to factor out shared information between lexical entries, lexical rules serve to state systematic dependencies between (types of) lexical entries, theoretically regardless of whether these share information, or not. Typically, the inflectional morphology of a language is encoded through lexical rules: all the forms of a morphological paradigm with their respective syntactic and semantic specifications are related to one base form, or lexeme lexical entry, via general lexical rules. This way, e.g. all past tense verb forms are related to their lexeme entry by the lexical rule displayed in Figure 21.

(Calcagno, 1995, p.12) examines the formal properties of lexical rules. He points out that, while "a [HPSG] grammar seeks to describe feature structures [(...)] with descriptions [...] a lexical rule seems to be a statement about descriptions and not about feature structures." More precisely, a lexical rule defines a relation between two sets of (lexical) feature structure descriptions, i.e. a set of ordered pairs of descriptions.

literature, see e.g. (Pollard and Sag, 1987, p.198ff), (Flickinger, 1987, p.17ff), (Riehemann, 1993, p.55f), (Müller, 1999, p.15f), (Sag and Wasow, 1999, p.4 and 174). In some of the sources, only the first of the two principles described above is emphasised.

$$
{}_{\textit{lexeme}}\begin{bmatrix} \text{PHON} & \boxed{2} \\ \text{SYN} & [\text{HEAD } \textit{verb}] \\ \text{SEM} & [\text{RESTR } \boxed{1}] \end{bmatrix} \longrightarrow {}_{\textit{word}}\begin{bmatrix} \text{PHON} & \text{F}_{\textit{PAST}}\,(\boxed{2}) \\ \text{SYN} & [\text{HEAD } [\text{FORM } \text{\textsc{fin}}]] \\ \text{ARG-ST} & \langle\, [\text{CASE } \text{\textsc{nom}}], \ldots \,\rangle \\ \text{SEM} & \begin{bmatrix} \text{INDEX} & \boxed{3} \\ \text{RESTR} & \boxed{1} \oplus \left\langle \begin{bmatrix} \text{RELN} & \text{\textsc{t-precede}} \\ \text{ARG1} & \boxed{3} \\ \text{ARG2} & \text{\textsc{now}} \end{bmatrix} \right\rangle \end{bmatrix} \end{bmatrix}
$$

Figure 21: Past tense verb lexical rule (after Sag and Wasow, 1999, p.195)

The relation defined by the rule in Figure 21 is somewhat complex, as several general assumptions about the interpretation of lexical rules are made. The rule expresses that for all lexical entries that are subsumed by the description on the LHS of the rule, a second lexical entry exists that is characterised by the changes in the RHS of the rule. The RHS contains only those paths of this second entry that are either altered with respect to the paths in the LHS, or added to the paths in the LHS of the rule. The second entry is supposed to additionally contain all the paths of the LHS that are not altered in the lexical rule. Thus, for every entry of type *lexeme* that has verbal head features, a second entry exists,

- whose type is subsumed by the type *word*
- whose phonology is a function of the phonology of the original entry
- which also has verbal HEAD features, one of which is [FORM FIN]
- whose first argument is realised by a phrase type that is specified for [CASE NOM]
- which additionally introduces a temporal precedence relation in its semantics between the time of the state or event expressed already in the original entry and the temporal specification 'NOW'.

Lexical rules have also been applied to derivation, valence alteration (such as the standard example of passive formation) and lexical semantic type shift phenomena (see e.g. Şehitoğlu and Cem, 1996).

(Pollard and Sag, 1987, p.209) already point out two possibilities for the general interpretation for lexical rules: "Lexical rules can be viewed from either a declarative or a procedural perspective: on the former view, they capture generalizations about static relationships between members of two or more word classes; on the latter view, they describe processes which produce the output form from the input form." Subsequent authors have taken one or the other perspective. For example, (Flickinger, 1987, p.131) takes the declarative/relational/symmetrical/non-directional point of view, whereas Sag and Wasow (1999) take the procedural/functional/asymmetrical/directional perspective.

Under the latter perspective, the second lexical entry mentioned above is meant to be newly created by the lexical rule in Figure 21.

For the most part, morphological lexical rules correspond to an item-and-process approach to morphology, as the procedural interpretation of lexical rules suggests. Typically, affixes are introduced in the lexical rules as additional phonological material in RHS of the rule, e.g. in the function F_{PAST} in Figure 21. F_{PAST} may be defined differently for different input domains (e.g. regular vs. irregular verbs) and thus introduces different markers (in the sense of Hockett, 1954, cf. Section 2.1), cf. Section 2.1 for one and the same rule (or, process).

(Koenig, 1999, p.27ff) summarises the major drawbacks of lexical rule approaches. Firstly, when lexical rules were introduced to unification-based approaches to generative grammar in Bresnan (1982), they were seen as the lexical device to take over the function of certain transformations, which were supposed to be an inadequate means of grammatical representation. (The other major device to replace transformations is structure sharing.) But as lexical rules are unrestricted relations between (descriptions of) feature structures, they still resemble transformations (on lexical signs). They have all the non-declarative characteristics of transformations: They have to be extrinsically ordered (for example, some device has to be found to prevent the lexical rule given in Figure 21 to apply to its own output, which is not excluded intrinsically). According to (Koenig, 1999, p.43ff), sets of descriptively adequate lexical rules may even result in genuine ordering paradoxes. And the example of Latin verbal morphology (Koenig, 1999, p.39f.) shows that the introduction of stem-forming suffixes each in a single lexical rule (which is descriptively economic) leads to intermediate representation levels which do not correspond to well-formed words. Moreover, in morphology, subregularities and exceptions abound, and cases can be shown where lexical rules have to be marked with exception features, or alternatively be themselves ordered in their own default hierarchy (Copestake and Briscoe (1992)). All these are arguments that demonstrate the inelegance of lexical rules within an HPSG approach. We believe that they have been introduced in Pollard and Sag (1987) because they seemed to be an convenient method to describe the rather closed domain of English inflectional morphology. Meanwhile, in several approaches, complex morphological processes involving morphological constituency have been represented without recourse to lexical rules.

4.4. Constraint-based derivation without lexical rules

4.4.1. Krieger 1993

Krieger's (1993) study starts with a criticism of lexical rules. He points out that lexical rules as introduced in Pollard and Sag (1987) are not feature structures, nor are they types. If represented using AVMs, they have the form $\text{AVM}_1 \longmapsto \text{AVM}_2$, i.e. they are mappings between feature structures. They are thus an alien element to the rest of the HPSG formalism where operations on feature structures and relations between linguistic types are the major devices to express grammatical information. Since lexical rules are proposed to be the formal means for expressing morphological information, similarities between syntax and morphology might not be obvious in the first place. An additional procedural mechanism has to be provided to deal with lexical rule application. Moreover, a description using lexical rules is said to lack declarativeness as these seem to be unidirectional.[21]

Alternatively, an approach to morphology using only feature structures is advocated. Krieger proposes only two kinds of lexical signs: *word* and *affix,* where word may be complex or not. There is only one morphotactic rule for derivation, or rather one morphological immediate dominance (ID) schema, the *morphological affix-word rule*, or, MAWR, which is displayed in Figure 22.

$${}_{\textit{complex}}\left[\begin{array}{ll} \text{SYN} \mid \text{LOC} \mid \text{LEX} & + \\ \text{DTRS} & {}_{\textit{affix-word-structure}}\left[\begin{array}{ll} \text{AFFIX} & \textit{affix} \\ \text{WORD} & \textit{part-of-speech} \end{array}\right] \end{array}\right]$$

Figure 22: The *morphological affix-word rule,* MAWR according to Krieger (1993)

In the MAWR (Figure 22), we can see that derived words have a binary structure expressed through two paths leading to the daughters DTRS|AFFIX and DTRS|WORD. It is postulated that the feature structure under AFFIX is always the head of such a structure though no reason is given for this setting. A morphological Head Feature Principle (MHFP) is given, but there is no explicit enumeration of HEAD features in morphology, but it can be concluded that the intended HEAD features are those enumerated in Pollard and Sag (1987). There is one remark concerning headedness in (Krieger, 1993, p.22, footnote 1):

> One might argue that only suffixes can be regarded as heads and prefixes should be given the status of a modifier (the syntactic category of the compound word [sic!] is determined by the free word which is the head in this case). However, under this assumption, we have to work with two ID rule schemata, one for prefixes and one for suffixes.

This is indeed a problem, but certainly not sufficient for considering prefixes heads, and on the contrary, this thesis adopts the position that prefixes are not heads in morphology, cf. Section 5.2.

[21]Krieger thus disregards the declarative point of view mentioned already by Pollard and Sag (1987).

$${}_{complex}\begin{bmatrix} \text{SYN}|\text{LOC}|\text{HEAD} & \boxed{1} \\ \text{DTRS}|\text{AFFIX}|\text{SYN}|\text{LOC}|\text{HEAD} & \boxed{1} \end{bmatrix}$$

Figure 23: The *Morphological Head Feature Principle,* MHFP, after Krieger (1993)

In the MHFP displayed in Figure 23, the syntactic HEAD features of the mother are token-identical with the HEAD features of the affix daughter, establishing the affix daughter as the head of a derived word. (Note that in addition to the established syntactic HEAD features motivated by percolation, we propose a separate set of *morphological* HEAD features equally motivated by feature percolation in Section 5.2.3).

Krieger's MHFP may not be very well motivated, but his *morphological subcategorisation principle* (MSCP) certainly is. Two kinds of subcategorisation are recognised in the area of derivational morphology:

1. morphological subcategorisation: "an affix looks for the right feature structure to bind"
2. syntactic subcategorisation: "the subcategorisation info of the new complex word (its sentential subcategorisation) directly comes from the syntactic subcategorisation by means of structure sharing."

The suffix *-bar* is a good example to illustrate this. Regularly, it combines with transitive verbs to form adjectives, i.e. it is morphologically subcategorised for transitive verbs, such as *ausstoßen.* On the other hand, the sentential subcategorisation properties of the resulting adjectives are already present in the *-bar*-suffix: The subject of the resulting adjective is the patient of the base verb, i.e. the subject of *ausstoßbar* is the patient of *ausstoßen.*

Krieger later formulates a revised MSCP which allows for mutual morphological subcategorisation of *word* and *affix.* This way, exceptions to regular affixation can be encoded as subcategorisation properties of the stems. It is, for example, a morphological subcategorisation property of the transitive verb lexeme *sehen* not to be combinable with *-bar.* A consequence of this is that *word* as well as *affix* are specified for the feature MORPH|SUBCAT.

Another innovation in Krieger (1993) is that principles and rules, originally presented in Pollard and Sag (1987) as implications between feature structures (in the form $\text{AVM}_1 \Rightarrow \text{AVM}_2$), are reformulated as types which are integrated in the subsumption lattice of the lexical type hierarchy. This stresses the identification of word formation rules with generalisations over lexicalised complex words, a notion elaborated largely in Riehemann (1998).

As affixes are regarded as lexical items, Krieger (1993) provides a lexical hierarchy

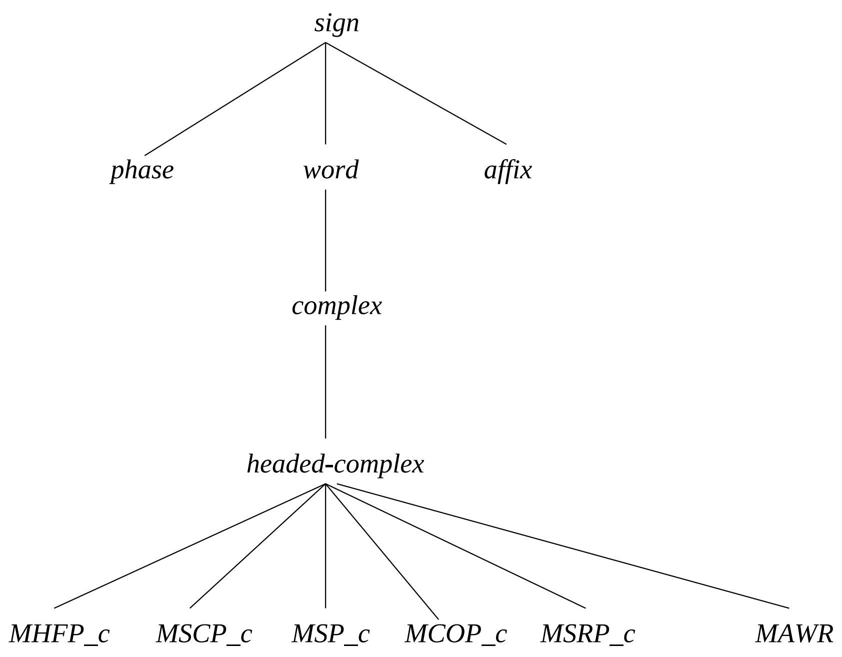


Figure 24: Part of the type lattice for morphological complex words in Krieger (1993)

of affixes. There are five dimensions along which affixes are cross-classified and which are particularly interesting for the present thesis.

- $\boxed{\text{POS}}$: encodes whether the affix is a prefix or a suffix
- $\boxed{\text{CAT}}$: encodes the category of the base and of the result of the affixation
- $\boxed{\text{SUBCAT}}$: encodes how the affix operates on the subcategorisation information of the base
- $\boxed{\text{SEM}}$: encodes the semantics of the affix, e.g. arity of predicates/operators.
- $\boxed{\text{BIND}}$: encodes the morphological subcategorisation information of the affix

These dimensions are not necessarily encoded in single features, but mostly by combinations of feature-value specifications and type restrictions on values.

4.4.2. Riehemann 1993, 1998, 2001

Riehemann (1998) introduces a new HPSG approach to derivational morphology, called *Type-Based Derivational Morphology* (TBDM).[22] The main motivations came from the analysis of large morphological data, especially *-bar*-suffixations in newspaper corpora.

[22]Riehemann's 2001 dissertation on idioms in HPSG contains a chapter on derivational morphology largely identical to the theory presented in Riehemann (1998).

Riehemann found that the *-bar*-suffixation data included regularities, subregularities, and exceptions among the lexicalised as well as among the newly created formations.

Traditional approaches assume a productive (lexical or phrase structure) rule stating that *-bar* is subcategorised for transitive verbs (cf. Toman, 1987).

Consequently, there are

1. new formations conveniently captured by the rule, such as *bestellbar, faxbar*
2. lexicalised words also captured by the rule. With these, the question arises, whether they are listed in the lexicon or whether they are processed by the morphotactic rule each time they are used. Examples are *bemerkbar, vermeidbar*
3. lexicalised words not captured by the rule, e.g. *sichtbar, fruchtbar.* These must be treated as exceptions and are listed in the lexicon.
4. new *-bar*-formations not captured by the rule, e.g. *-bar*-formations not based on transitive verbs such as the notorious *unkaputtbar.*

Riehemann observes that almost none of the lexicalised words not captured by the rule (mentioned under 3. above) are fully exceptional but rather have some exceptional properties as well as other regular properties. She provides a classification of these words on account of their exceptional properties:

1. phonologically exceptional:
   - formations with dropping of *-ig* in the stem:
     entschuldbar, from the verb *entschuldigen*[23]
2. semantically exceptional:
   - formations with an additional aspect of meaning: *eßbar*, '**safely** edible'
   - formations with the intensional dimension of obligation instead of possibility: *zahlbar*
   - formations lexicalised in only one particular sense: *haltbar*
3. syntactically exceptional:
   - formations from verbs with dative objects: *unentrinnbar*
   - formations from verbs with prepositional objects *verfügbar*
   - formations from intransitive verbs *haltbar, brennbar*

[23]It is not necessary to view *entschuldbar* as being derived from *entschuldigen* by the removal of *-ig.* We would instead propose that the base *entschuld* bears the complete verbal meaning of *entschuldig-* already, and that *-ig* is an affix that carries no meaning.

For all of the above subregular classes of lexicalised *-bar*-formations, parallel non-lexicalised examples can be found as well, i.e. formations that do not correspond to the productive rule for new formations. “These adjectives are formed from verbs which do not have an object in accusative case, thereby violating what is seen in traditional word-syntactic approaches as the ‘subcategorization requirements’ of the affix.”, e.g. from verbs with dative objects (*unausweichbar, unentziehbar, unwiderstehbar*), from verbs with prepositional objects (*unzweifelbar, verzichtbar, zugreifbar*) from intransitive verbs (*unausbleibbar, ungerinnbar, verrottbar, verheilbar*)

Riehemann wants to account for two kinds of relations that have been neglected in previous approaches to derivational morphology: Firstly, the relations between regular formations, and different kinds of subregular cases and exceptions. Secondly, the relations between lexicalised complex words and non-lexicalised, ad-hoc formations, more precisely the contribution of lexicalised words to productive word formation. Riehemann denies that Word syntax or lexical rule approaches are able to describe these relations adequately.

In Type-Based Derivational Morphology, there is a schema expressing the fact that there is a class of words, ending in the suffix *-bar*, that have transitive verbs as their morphological basis. It also states how the syntax and semantics of the verb relates to that of the adjective. For example, the accusative object of the verb is linked with the subject of the adjective, and the semantics of the verb reappears within the scope of the possibility operator in the semantics of the adjective.

$$ {}_{\textit{reg-bar-adj}}\left[\begin{array}{ll}
\text{PHONOLOGY} & \boxed{1} + \textit{bar} \\
\text{MORPH-B} & \left\langle \left[\begin{array}{ll} \text{PHON} & \boxed{1} \\ \text{SYNSEM} \mid \text{LOC} & {}_{\textit{trans-verb}}\left[\begin{array}{ll} \text{CAT} \mid \text{VAL} \mid \text{COMPS} & \langle\, \textbf{NP [acc]:}\ \boxed{2}\, \rangle \oplus \boxed{3} \\ \text{CONT} \mid \text{NUC} & \boxed{4} \left[\begin{array}{ll} \text{ACT} & \\ \text{UND} & \boxed{2} \end{array}\right] \end{array}\right] \end{array}\right] \right\rangle \\
\text{SYNSEM} \mid \text{LOCAL} & \left[\begin{array}{ll} \text{CATEGORY} & \left[\begin{array}{ll} \text{HEAD} & \textit{adj} \\ \text{VALENCE} & \left[\begin{array}{ll} \text{SUBJ} & \langle\, \textbf{NP:}\ \boxed{2}\, \rangle \\ \text{COMPS} & \boxed{3} \end{array}\right] \end{array}\right] \\ \text{CONTENT} \mid \text{NUCLEUS} & \left[\begin{array}{ll} \text{RELN} & \diamond \\ \text{ARG1} & \boxed{2} \\ \text{ARG2} & \boxed{5} \end{array}\right] \end{array}\right]
\end{array}\right] $$

Figure 25: Schema for the fully productive, regular *-bar*-adjective, cf. Riehemann (1998)

No morphological constituent structure is represented in such a schema but it is said that, “although affixes are not explicitly represented as part of the structure, the stems are, which makes the internal structure of complex words recoverable”. In TBDM, affixes appear only in the phonology, consequently, they are not signs and have thus neither an independent status outside a schema, nor can they be the head of a complex word.[24] This is what TBDM has in common with item-and-process/ lexical rule approaches. But whereas lexical rules cannot be modelled as single feature structures, schemata can and are associated with lexical types which are ordered in a type hierarchy to treat subregularities and exceptions as well. In fact "there are also schemata for the other (subregular) patterns", and "all schemata serve primarily to organize the existing lexicon, and are only secondarily used to form new words", i.e. "rules are seen as generalizations emerging from existing words." Riehemann's hierarchy of -*bar*-adjectives is given in Figure 26.

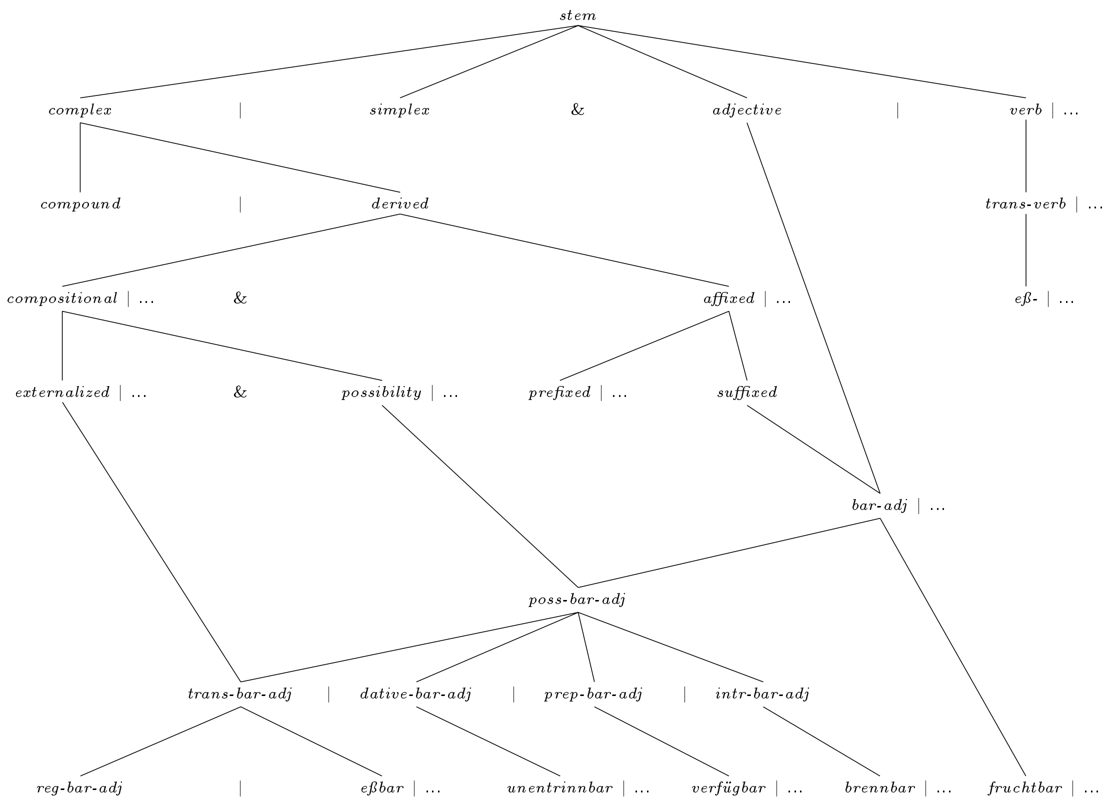

stem
complex | simplex & adjective | verb | ...
compound | derived
trans-verb | ...
compositional | ... & affixed | ...
eß- | ...
externalized | ... & possibility | ... prefixed | ... suffixed
bar-adj | ...
poss-bar-adj
trans-bar-adj | dative-bar-adj | prep-bar-adj | intr-bar-adj
reg-bar-adj | eßbar | ... unentrinnbar | ... verfügbar | ... brennbar | ... fruchtbar | ...

Figure 26: Hierarchy of -*bar*-adjectives after Riehemann (1998)

For the types in Figure 25, a closed world interpretation is assumed, i.e. there are not any more subtypes than those that are explicitly mentioned. That is the reason why the schema *reg-bar-adj* describing the fully transparent non-lexicalised words is on one level with the various lexicalised -*bar*-adjectives.

It could be argued that a pure word-syntactic approach like the one by Krieger (1993)

[24]Riehemann (1998): "One could of course think of an entire TBDM schema as being an unusual kind of 'lexical entry' for the suffix. But the proposed analysis has the advantage of generalizing readily to non-affixal morphology.".

which also includes a hierarchically organised lexicon and in addition hierarchically organised morphological principles can capture the subregularities and exceptions quite as well, and that actually no improvement is achieved through the TBDM approach. But Riehemann points out that in Krieger's approach, the morphological hierarchy is duplicated: One the one hand, a lexical type hierarchy for complex words with their subregularities and exceptions is described, one the other hand, for every lexicalised subtype of *-bar*-adjectives, a separate reading of a *-bar*-suffix with its appropriate subcategorisation specifications has to be introduced into the affix hierarchy, which is used in productive and semi-productive word formation. These two separate hierarchies arise essentially from of the same linguistic data and obscure the fact that productive derivational morphology is based on analogy to existing words. Riehemann, however has incorporated the notion of word formation as analogy in her approach in a non-redundant way.

Contrary to Krieger's (1993) approach, which actually has been implemented, Riehemann (1998) gives us almost no idea of how computational morphological processing could be performed using a morphotactics and a lexicon as suggested by her. There is one footnote with a hint at how a parser could proceed, "As a first approximation for increasing robustness in an computational system one could say that if a word cannot be found in the lexicon, all word-formation schemata with more than a number of subtypes can be tried."

But how do we have to conceive this 'trying' of schemata? The form of the affix is the only fixed surface information in a schema, i.e. only the affix can be used as a lookup key for a schema. For identifying *-bar* in an input word form like `verrottbares`, morphological segmentation has to be performed first, if a bottom-up strategy is chosen. Then the most specific schema (the one lowest in the lexical type hierarchy) containing the concatenation of a stem with *-bar* and unifying with the stem *verrott* has to be found. In other words, the representation `bar` is used to access the matching schema in an inventory of schemata in a systematic way (considering the subsumption relation between schemata). But we consider this being tantamount to looking up the properties of the affix in an affix lexicon. In fact, it leads us to suspect that Krieger's (1993) approach is a compiled-out version of TBDM, the latter being more compact than the former, which contains some systematic redundancies. However, these redundancies seem to be advantageous for morphological parsing.

5. An HPSG theory of German morphology

5.1. The structure of simplex and complex signs

We consider morphemes to be linguistic signs, which in terms of Pollard and Sag (1987) means that they are appropriate for the feature PHON(OLOGY) as well as SEM(ANTICS). For our theory, we follow Gibbon (1997) and Gibbon (2000) and replace PHON by the feature SURF(ACE) which contains both phonological and orthographic information and is structured as shown in Figure 27.

$${}_{\textit{sign}}\left[\begin{array}{ll} \text{SURF} & \left[\begin{array}{ll}\text{PHON} & \textit{phon-string}\\ \text{ORTH} & \textit{orth-string}\end{array}\right]\\ \text{SYN} & \textit{syntax}\\ \text{SEM} & \textit{semantics}\end{array}\right]$$

Figure 27: The structure of a *sign*

We will not say much else about SURF, but we want to mention here that we believe that for phonology, eventually representational structures that are more complex than lists will be needed.

In the following, we will discuss amongst other things syntactic features that play a role in morphology, but we omit semantic features, and that is one of the reasons why in our theory, we want to keep the distinction between SYN(TAX) and SEM(ANTICS) made in Pollard and Sag (1987). The other reason is that we want morphological subcategorisation to be able to refer to SURF properties of signs which is excluded under the SYNSEM approach of Pollard and Sag (1994), cf. Section 4.1.

For lexical signs, we also introduce a MORPH(OLOGY) feature as in Krieger et al. (1993), though with slightly different implications. In the present approach, the main idea is that everything stored under MORPH (namely morphologically combinatorial properties) need not be known to the syntax component. For example, the syntax is interested in the morphosyntactic specifications of a word form as a syntactic atom (cf. Di Sciullo and Williams, 1987, p.46ff), e.g. that is is a first person singular present indicative verb, but not whether it is a weak or a strong verb.

Likewise, the feature MORPH|MPHON is appropriate for *morpheme* (Figure 28). The idea is that the two-level morphophonological rule component, which is responsible for constructing the SURFACE attributes of morphologically complex signs, will need the features values under MORPH|MPHON to compute the correct surface strings. The value of MPHON is structured as shown in Figure 28.

Within *mphon* (the value type of MPHON), information about whether a morpheme can/must/must not be bound to its right or left side is stored under the paths

$${}_{morpheme}\left[\begin{array}{ll} \text{MORPH} & \left[\begin{array}{ll} \text{HEAD} & [\text{DER}\ [\text{NAT}\ \textit{boolean}]] \\ \text{MPHON} & {}_{mphon}\left[\begin{array}{ll} \text{RBOUND} & \textit{boundness} \\ \text{LBOUND} & \textit{boundness} \\ \text{STR} & \textit{boolean} \end{array}\right] \end{array}\right] \end{array}\right]$$

Figure 28: Constraints for the type *morpheme*

RBOUND|BOUND and LBOUND|BOUND, and information about what morphological subtype of sign the morpheme must be bound to is stored under RBOUND|BND-ARG, see Figure 29.

$${}_{boundness}\left[\begin{array}{ll} \text{BOUND} & \textit{bound} \\ \text{BND-ARG} & \textit{morpheme} \end{array}\right]$$

Figure 29: The structure of *boundness*

Consequently, the type *bound* is partitioned as shown in Figure 30. Boundness features were formerly employed by Lehmann (1990). They are an extension of the structuralist distinction between *free* and *bound* morphemes. The former are morphemes which may occur freely as syntactic atoms, the latter are morphemes that need to be bound to a base, i.e. can never occur freely. Originally, bound morphemes were supposed to be only affixes, but in German, clearly bound roots can be found, too, e.g. *-pliz-*, or *sprech-*.

Our possible values of BOUND should be interpreted as follows: *toBind* means that this morpheme needs another morph to its right (left) side when occurring in a word form. *Opt(ional)* means that another morpheme on its right (left) side may or may not occur within a word form, and *free* means that this morph can never combine with another morpheme on its right (left) side within the domain of a word form. Note that many lexical roots remain underspecified (have the value *bound*) on the morpheme level and can be fully specified for BOUND only on the morph level (cf. Section 5.4), which reflects the relevance of this feature for surface linear precedence properties rather than valency and immediate dominance properties.

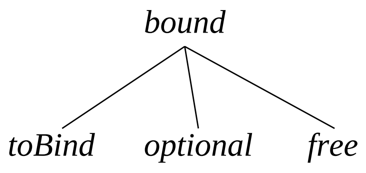


Figure 30: Partition of the type *bound*

The values of the RBOUND|BND-ARG and LBOUND|BND-ARG features are typed to *morpheme*, as shown in Figure 29. Alternatively, they might be typed to *base* and could

morpheme	as occurring in	RBOUND |BOUND	RBOUND |BND-ARG	LBOUND |BOUND	LBOUND |BND-ARG
gener_root	`gener+at+ion`	*toBind*	*suffix*	*opt*	*stem-final* $\sqcap$ *prefix*
ier_sufnn	`gener+ier#+en`	*opt*	*suffix* $\sqcap$ *infl* $\sqcap$ *stem-initial*	*toBind*	*base-final*
ver_pre1	`ver+trag#+en`	*toBind*	*base-initial*	*opt*	*stem-final* $\sqcap$ *prefix*
pliz_root	`kom+pliz+ier#+t`	*toBind*	*suffix* $\sqcap$ *infl*	*toBind*	*prenn*
sehr_root	`sehr`	*free*	$\perp$	*free*	$\perp$

Table 1: Examples of *boundness* specifications

be co-indexed with the MORPH|SUBCAT feature, but we want to employ these features to encode linear precedence relations between subtypes of *morpheme*, i.e. continuation classes in the sense of Koskenniemi (1983b).

RBOUND and LBOUND are used to build the dimension that divides the type *affix* into *prefix* and *suffix*. In contrast to MORPH|SUBCAT, they are not only appropriate for affixes, but also for lexical roots, and in particular they describe the fact that many nonnative lexical roots (such as *gener-* in Table 1) do not constitute independent stems without the addition of an affix, cf. Section 5.3. Examples of RBOUND/LBOUND specifications of morphemes can be found in Table 1.

In Figure 28, the next feature that is appropriate for all kinds of morphemes, is MORPH|HEAD|DER|NAT(IVE). This *boolean*-valued feature models two of the lexical strata posited in the theory of Lexical Phonology and Morphology (Siegel, 1979; Mohanan, 1982)). It has also been introduced as a morphological feature in the GPSG-based morphological theory by Ritchie et al. (1992) (called LAT(INATE)). The following kinds of morphological behaviour in German are explained by the feature NAT in our theory:

1. Nonnative affixes do not combine with native lexical roots, cf.

   ```
     Absurd+ität
     In+komp+at+ibil+ität
   * Falsch+ität
   * Un+ver+ein+bar+ität
   ```

   This is achieved by constraining nonnative prefixes and suffixes to be subcategorised only for [NAT −] bases, cf. Section 5.5.1.
2. Only a certain subgroup of native suffixes can combine with nonnative bases, namely those native suffixes specified as [COMB +].
3. Nonnative lexical roots are frequently bound morphs, they need either a nonnative prefix or a nonnative suffix or both to form a complete stem (something that can be inflected), cf. possible combinations with the nonnative root *pliz*:

   ```
   kom+pliz+ier#+t
   im+pliz+it
   ```

```
  re+pliz+ier#+en
  Kom+pliz#+e
* im+pliz
* pliz+ier#+t
* pliz
```

(The '*' here refers to potential stemhood.)

4. The positioning of lexical stress in derivations is sensitive to the difference between native and nonnative affixation, the tendency being that the last heavy syllable is accentuated, unless it is a, or contains a [NAT +] suffix, cf.

   `m'an+Saft` *(Mannschaft)* vs.
   `pro:ble:m+'a:t+Ik` *(Problematik)*

A NATIVE feature for stress assignment in words was also suggested by (Pampel, 1991, p.30).

morph	as in the word form	NAT
absurd_root	`Absurd+ität`	−
falsch_root	`Falsch+heit`	+
ität_suf	`Absurd+ität`	−
kom_prenn	`kom+pliz+ier#+t`	−
pliz_root	`kom+pliz+ier+#+t`	−
ver_pre1	`ver+miss#+en`	+

Table 2: Examples of specifications for NAT(IVENESS)

In complex lexical signs, the NAT features should of course percolate, and we chose to group it as a (derivational) HEAD feature and let it be shared between the mother and the head daughter constituent according to the MHFP. For one thing, the line of argumentation in Döpke and Walmsley (2000) leads us to the conclusion that NAT must be a HEAD feature, and moreover such a treatment has shown to yield satisfactory results from our lexical acquisition system (Section 5).

The last feature which is supposed to be appropriate for all kinds of morphemes is MORPH|MPHON|STR(ESS), denoting potential lexical stress. It is based on a distinction originally drawn by Féry (1986) between stress-bearing and stress-neutral affixes, and a similar feature is used in Steinbrecher (1995) for the classification of affixes. It is e.g. responsible for the distinction between the class I and class II native prefixes of German (the types *prefix1* and *prefix2* in Figure 65 in Section 5.3), i.e. the NAT dimension is definitely not sufficient to distinguish all types of prefixes. Bleiching (1992) employs a three-valued feature STRESS to distinguish between stress-bearing, stress-neutral, and stress shift-causing affixes. The members of the latter class seem to be those suffixes (inflectional and derivational) that are appropriate to be subcategorised for nominal stems of the *noun-doktor* class (which comprises nouns that end in the suffix *-or* and three minor nonnative suffixes listed in (Pampel, 1991, p.46)). Since *noun-doktor* is a possible value of the SUFFIXCLASS feature in our morphology (Figure 60), a third feature value *stress-shift-causing* for STR would be redundant and is thus abandoned in our theory.[25]

Like NAT, STR is *boolean*-valued.

We thus employ STR in analogy to Steinbrecher (1995), but extend it to lexical roots and inflectional suffixes i.e. it is appropriate for all subtypes of *morpheme*. *Root* is always [STR +] and *infl* is always [STR −], and the STR specifications for different derivational affixes will be discussed in Section 5.5.1.

The value of STR for each morpheme is input to a stress assignment component, which assigns stress values to the syllables of a morphologically complex word. (Further input to the stress assignment component are a word form's internal constituent structure (cf. Bleiching, 1991, p.20f.) and its syllable structure representation (cf. Pampel, 1991, p.38f.).

5.2. Heads and subcategorisation

Do complex words have heads, and if so, what are they? What are the HEAD features in morphology? And does HPSG's Head Feature Principle for syntax work for morphology as well, or are different principles to be found in morphology?

The concept of *head* was introduced to generative grammar through X-bar theory (Chomsky, 1970; Jackendoff, 1977). In this subtheory of generative grammar, the X-bar schema[26] given in 5.1 is devised as a generalisation over traditional phrase structure rules for syntax such as displayed in 5.2.

5.1 $X^{n} \longrightarrow \ldots X^{n-1} \ldots$

[25]In fact, the marked property of words containing the suffix *-or* should be that *-or* is unstressed when it occurs word-finally instead of considering some suffixes which may be attached to it to be stress shift-causing, because all non-final occurrences of *-or* do conform to the general rule of stress assignment for nonnative suffixes and words which contain them.

[26](Jackendoff, 1977, p.30).

	(a)	NP	$\longrightarrow$	(Det) (A) N
	(b)	NP	$\longrightarrow$	N
	(c)	VP	$\longrightarrow$	V
	(d)	VP	$\longrightarrow$	V (NP)
5.2	*(e)*	VP	$\longrightarrow$	V (NP) (PP)
	(f)	AP	$\longrightarrow$	A
	(g)	AP	$\longrightarrow$	A (AP)
	(h)	AP	$\longrightarrow$	A (PP)
	(i)	PP	$\longrightarrow$	P NP

In each of the rules in 5.2, there is an obligatory lexical element, which is also responsible for the name of the phrase as a whole, e.g. an NP is called **N**P because N is its obligatory element. In other words, rules like

5.3 NP $\longrightarrow$ V VP

where no N occurs on the RHS, are linguistically impossible, but not formally excluded in traditional PS-grammars. In the X-bar schema, the obligatory element is the *X*, and it is referred to as the *head* of the phrase. X^{max} is the maximal projection of X, and, depending on the number of embedded complements and specifiers of X, there may also be constituents that are intermediate projections X^n (max $\geq$ n $\geq$ 0) (or intermediate bar-levels) of X (cf. Figure 31). In an instantiation of the X-bar schema, X is bound to N, A, V, or P, and the categories NP, AP, VP, PP are replaced by the projections $\bar{X}$, $\bar{\bar{X}}$, and so forth. The X-bar schema thus expresses what NPs, APs, VPs, and PPs have in common, removing redundancies from traditional phrase structure grammars. It is important to note that things like N and NP are no longer atomic symbols of the grammar. The idea that e.g. N as well as NP have nominal characteristics is expressed by the fact that all kinds of NPs are projections of N and bear the feature specifications [N+, V−].[27] N is thus not a category, but a feature, and the feature specification [N+] is not only appropriate for nouns, but also for adjectives (and, presumably, for pronouns).[28]

In general, X and X^n share further feature specifications. In studies of the 1980s such as Selkirk (1982) and Di Sciullo and Williams (1987), this sharing is called *feature percolation*. Feature percolation is a reflection of the compositionality principle, i.e. the fact that a sentence or complex word form has properties that are a function of the properties of its parts. A Head Feature Principle is thus a generalisation that predicts for each complex construction type what kind of constituent contributes which properties to the construction. The term *percolation* (as opposed to *sharing*, which is generally

[27]We are aware that this is debatable, especially in the case of NPs, where alternative analyses as DPs (e.g. Stowell, 1990) have been proposed, and cf. Netter (1994) for DP analyses for German in HPSG.

[28](Jackendoff, 1977, p.31).

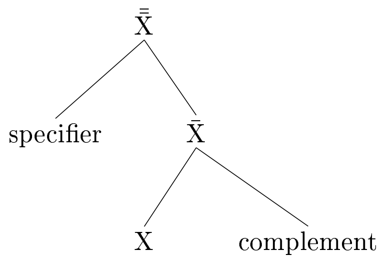


Figure 31: X-bar schema after (Jackendoff, 1977, p.17)

used in feature-based grammars such as HPSG) reflects a procedural point of view in that the features of an X (N, A, V, or P) are listed in the lexicon, and 'percolate' from the X in a generated tree structure when the head is retrieved by a lexical insertion rule. An example of feature percolation in syntax can be seen in Figure 32.

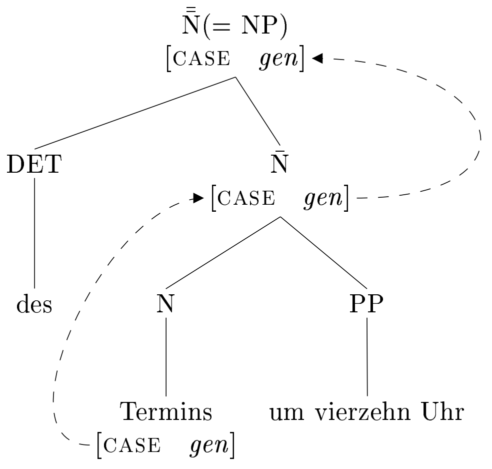


Figure 32: Tree structure and feature percolation in $\bar{\bar{N}}$

What is the motivation at all that an NP (i.e. $\bar{\bar{N}}$) shares feature specifications with its head N? In a constituent grammar (which HPSG certainly is, as opposed to a dependency grammar), generally only maximal projections can be subcategorised for, e.g. NPs. But in many languages certain characteristics of the NP manifest themselves only on the head of that NP, most notably the syntactic category N itself, but also, for instance, a case specification. Case is genuinely a morphosyntactic property pertaining to nouns, but since it is NPs that are subcategorised for by other signs, case specifications must be properties of NPs, too.

Note that in a dependency theory of grammar this poses no problem in the first place, as the head (or *nucleus*) in a dependency structure is always found on top of a tree (cf. Mel'cuk, 1988, p.23). Hence HEAD feature percolation may also be considered a device to incorporate information contained in dependency grammar analyses into phrase structure grammar analyses.

But there are further arguments for HEAD feature percolation to maximal projections. In several languages, e.g. German and Finnish, in NPs case must also be marked on determiners and adjectives as a consequence of agreement which holds within the NP. In that sense, in an NP like 'des dringenden Termins', we can talk about the whole NP being marked for genitive case.

So far, we have introduced the notion of the head of a syntactic phrase as

- the obligatory element of that phrase
- the element from which grammatical features percolate to the phrase as a whole
- the element which would be considered the head in dependency theory

Hudson (1987) elaborates on the importance of the head concept in a theory of feature percolation in syntax, reviewing an earlier article by Zwicky (1985). He also comes to the conclusion that, in addition to being the source for feature percolation in syntax, the head is

> suited to bringing together [...] six distinct notions, which we can [...] take as (more or less) independent properties of the head of a construction: it is the semantic functor, the morphosyntactic locus, the subcategorizand, the governor, the distributional equivalent and the obligatory element.[29]

From the viewpoint of HPSG, we can basically agree with these assumptions,[30] though we think that the prominent criterion for headedness is the possibility to formulate a consistent Head Feature Principle. We will use Hudson's list as an additional guideline in search of the heads and HEAD features in morphology and their representation in HPSG in Sections 5.2.1-5.2.3.

First, let us take a look at the *syntactic* Head Feature Principle (HFP) of HPSG as introduced in (Pollard and Sag, 1987, p.58) and again in (Pollard and Sag, 1994, p.34). In Figure 33, we present the HFP as a constraint on a construction type called *headed-phrase*.

The HFP of HPSG is a declarative reformulation of earlier percolation principles, especially the (also declarative) Head Feature Convention of GSPG (Gazdar et al., 1985, p.94f.). It says that the HEAD features of a phrase are identical with the HEAD features

[29](Hudson, 1987, p.124).

[30]We can agree with these assumptions maybe with the exception of the question whether N is actually the obligatory element, the distributional equivalent, and the main morphosyntactic locus of an NP (Hudson's (1987) claims rest on a DP analysis).

$$\textit{headed-phrase} \Rightarrow \begin{bmatrix} \text{SYN|LOC|HEAD} & \boxed{1} \\ \text{DTRS} & {}_{\textit{headed-structure}}\big[\text{HEAD-DTR|SYN|LOC|HEAD} \ \boxed{1}\big] \end{bmatrix}$$

Figure 33: Head Feature Principle

of its head daughter, without any indication of copying and directionality. The HFP is considered a principle of universal grammar. Much of its impact depends on the definition of the HEAD features which have to be inventorised language-specifically. For English, (Pollard and Sag, 1994, p.396ff) give the following partition of the type *head*, which is provided here in the form of a type inheritance graph, including declarations of appropriate features where they are introduced (Figure 34). *Head* is the supertype of all types that may be values of the feature HEAD.

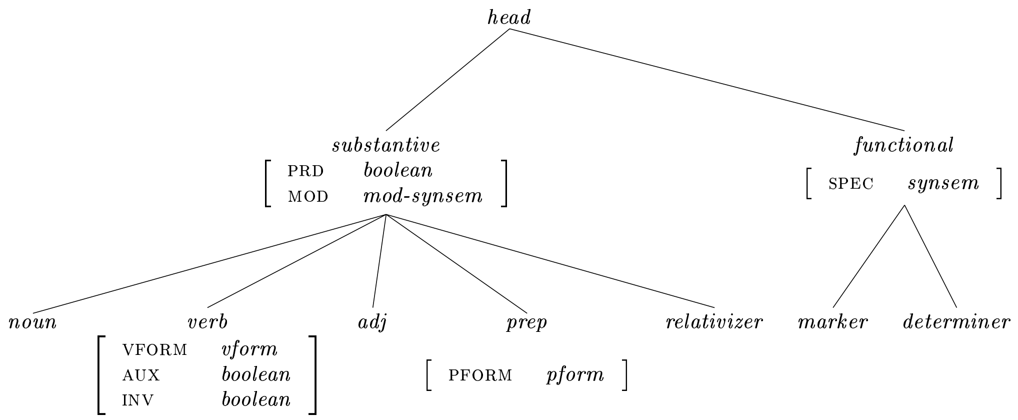


Figure 34: Partition of the type *head* after (Pollard and Sag, 1994, p.396ff)

In (Pollard and Sag, 1987, p.67f.), MAJOR (the syntactic category) is also listed among the HEAD features. In the fully typed feature system of Pollard and Sag (1994), the feature MAJOR and its possible atomic values N,V,A,P, AND ADV, are replaced by the possible (non-atomic) value types of HEAD, as displayed in the hierarchy in Figure 34.

Now that we have a notion of a.) what heads in syntax are and b.) what the HEAD features in syntax are, we proceed to examine morphological constructions in German with respect to the following questions:

- Does one of the constituents of a morphological construction bear functions similar to the ones defining heads in syntax as well as to those listed by Hudson (1987)?
- Are some or all of the features that Pollard and Sag (1994) and others list as HEAD features involved in morphological constructions, too? And if so, can some sort of HFP be postulated for morphology as well?

- Apart from the syntactic HEAD features (which may play a role in morphology, too), are there other features that are candidates for being HEAD features in morphology?

For the following head analyses, we assume that there are four general morphological construction types that involve concatenation in German (i.e. all other possible morphological concatenative construction types are subtypes of these, cf. Section 5.3):

1. Compounds (Figure 37)
2. Derivational prefixations (Figure 35)
3. Derivational suffixations (Figure 36)
4. Inflectional suffixations (Figure 38)

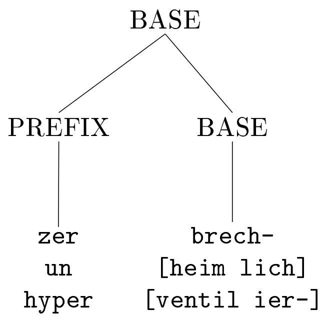


Figure 35: Constituent structure of derivational prefixations

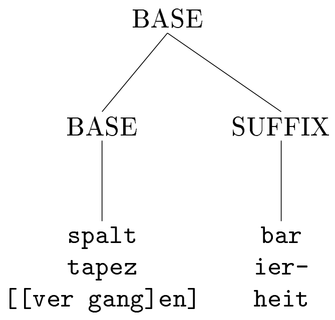


Figure 36: Constituent structure of derivational suffixations

5.2.1. Headedness in compounding

Let us first examine the most uncontroversial construction type with respect to headedness, that is, compounding. The right-hand constituent in a compound is commonly

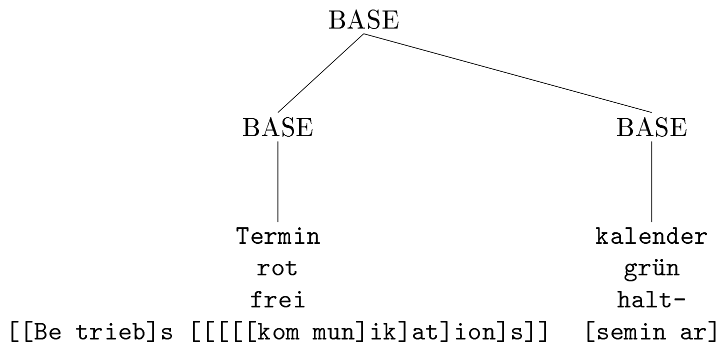


Figure 37: Constituent structure of compounds

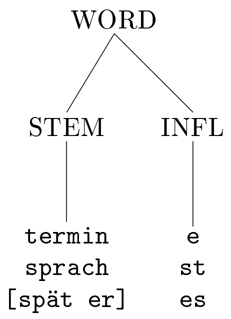


Figure 38: Possible constituent structure of inflected words

regarded as its head, and it fulfils almost all of the criteria mentioned by Hudson (1987). We will briefly go through them here, clarifying their meaning and applying them to compound constructions in German.

In order to establish the left or the right constituent of a compound as its *obligatory element*, one should be able to formulate a proposition like *In order for a morphological construction like the one in Figure 37 to be a compound, it must at least consist of the left (the right) constituent.* But if the constituents are simplex bases, such a proposition can never be true, since a simplex word is not a compound by definition. If one of the constituents is a compound itself, then either of the alternative propositions may be true, depending on whether the compound is the left or the right constituent. Thus, looking for the obligatory element in a compound reveals nothing about headedness in that compound.

Whenever the two constituents of a compound are semantically *a functor and one of its arguments*, the right constituent is indeed represented by the functor.[31] There are two subtypes of compounds where this can be observed. The first type of compound is where

[31]Hudson (1987) uses the term *functor* in the sense of *logical predicate.*

the right constituent is the nominalisation of a verb, such as `Termin#verschiebung`, or `Doktoranden#treffen`. The functor-argument structure of this type of compound can be represented as $p(x_1, ..., x_n)$ where $n \geq 1$ and p always represents the right constituent, and the left constituent is represented by one of the x_i. It is shown by the fundamentally different semantic interpretations of examples like `Abschluß#sitzung` vs. `Sitzungs#abschluß` that the right constituent is always the position to be interpreted as the functor.

The second type of compound expressing functor-argument structures is the more general case where the right constituent is a relational noun (not necessarily a nominalisation), such as e.g. `Hälfte`, or `Ende`. In the example `April#hälfte`, we are dealing with a semantic structure like $p(x)$ where p represents *Hälfte*, x represents *April*, and $p(x)$ represents the compound *Aprilhälfte* as a whole.

The strategy for identifying the head of a compound by looking for the semantic functor cannot be applied, though, when the two constituents are not a functor and one of its arguments, as is the case with many compounds, e.g. `April#scherz, Espresso#bar, Video#raum`. Even if one of the constituents is a nominalisation or a relational noun, the other constituent need not be an argument of the functor represented by the nominalisation or relational noun as e.g. in `Donnerstags#treffen`.

On the other hand, before coming to the conclusion that the semantic head must be the functor in a functor-argument construction, Hudson (1987) agrees with the notion that "in a combination X + Y, X is the 'semantic head' if, speaking very crudely, X + Y describes a kind of the thing described by X", which was originally suggested by (Zwicky, 1985, p.4). This is definitely true for all determinative compounds (*tatpuruṣa*), which form the majority of compounds in German and which are the general and productive semantic compound type in German. Other, minor, semantic types of compounds in German are possessive compounds (*bahuvrīhi*, like `Dumm#kopf`), and copulative compounds (*dvandva*, like `Import-Export`), these are *not* semantically right-headed (Spencer, 1991, p.310ff).

When the two constituents of a compound are a subcategorisand and one its subcategorised elements, the right constituent is indeed always the subcategorisand. The argumentation is quite parallel to the case of semantic functors, obviously because the semantic functor and the subcategorisand are the semantic and the syntactic side, respectively, of the same thing. Likewise, typical examples are compounds, where the right constituent is a nominalisation.

According to Hudson (1987), *government* is distinct from subcategorisation in that in government one element controls certain morphosyntactic features of another element, i.e. a verb governs an NP by specifying that the CASE feature of that NP have *accusative* as its value, for example. Subcategorisation is rather about the presence or absence of complements and (in case of presence) the number and type of complements a verb has.

Nevertheless, in HPSG syntax, government is realised most of the time by constraints on subcategorisation lists.

In German compounds, nothing can be found that resembles (syntactic) government, as the shape of one element is in no way determined by the other. The type of linking morpheme that occurs in some compounds is determined solely by the left constituent, i.e. the one to which it is suffixed, and never by the right (the other) constituent.

The right constituent of a compound is definitely the *morphosyntactic locus* i.e. the place where morphosyntactic categories are realised. The fact that inflectional affixes are always suffixes in German,[32] is already a strong hint that the right element in a morphological construction is the morphosyntactic locus of that construction. In addition, if morphosyntactic categories are expressed by morphological operations other than affixation, such as vowel mutation, this also always affects the right constituent of a compound. Even in the case of dvandvas or bahuvrihis, where there is no unique semantic head, the morphosyntactic locus is the right constituent.

From a morphological point of view, the morphosyntactic categories are those features specifications that are expressed by inflection and that can be enumerated in a paradigm (see Section 5.3). They are defined differently for the different parts of speech. We consider lexical roots such as `sprach` and `termin` as specified for the syntactic HEAD features shown in Figure 39 on account of their behaviour as syntactic atoms. An alternative analysis would be to consider morphological bases as not appropriate for the syntactic HEAD features. The syntactic HEAD features would then have to be introduced by a rule (a lexical rule or a null-affixing schema) for compounds and non-compounds alike. A rule with the same kind of effect on the feature specifications on a compound is needed anyway for combining non-null inflectional suffixes with bare stems. (Koenig, 1999, p.142ff) represents inflection along these lines, but we refrain from such a representation lest zero affixes have to be stipulated.[33]

We present the nominal and verbal morphosyntactic categories as the types *s-vhead* and *s-nhead* in Figure 39, and we argue here, that they are HEAD features in compounding.

There is a second set of potential HEAD features, which is responsible for *morphotactic* behaviour. These features encode nativeness (NAT), umlauting properties (UML), the property of possibly taking linking morphemes in compounding (LINK), and, most important, inflectional class properties for inflection (STEMCLASS, SUFFIXCLASS, and VROOT, cf. Section 5.5.2). The idea of features constraining the combinatorial potential of ele-

[32]If the participle-forming prefix `ge-` were considered an inflectional affix, it would be a counterexample to this claim. On constructional and prosodic grounds, however, we treat it as a a derivational class I prefix.

[33]In Koenig's theory, zero affixes pose not so grave a problem, as inflectional morphemes are denied the status of signs in the first place.

$$
{}_{\textit{s-vhead}}\left[\begin{array}{ll}
\text{MAJ} & \textit{verb}\\
\text{TENSE} & \textit{tense}\\
\text{MOOD} & \textit{mood}\\
\text{FIN} & \textit{finiteness}\\
\text{AGR} & {}_{\textit{agreement}}\left[\begin{array}{ll}\text{PER} & \textit{person}\\ \text{NUM} & \textit{number}\end{array}\right]
\end{array}\right]
\qquad
{}_{\textit{s-nhead}}\left[\begin{array}{ll}
\text{MAJ} & \textit{noun}\\
\text{CASE} & \textit{case}\\
\text{GENDER} & \textit{gender}\\
\text{AGR} & {}_{\textit{agreement}}\left[\begin{array}{ll}\text{PER} & \textit{person}\\ \text{NUM} & \textit{number}\end{array}\right]
\end{array}\right]
$$

Figure 39: Verbal and nominal syntactic HEAD features

ments is parallel to subcategorisation in syntax: The fact that the verbal root `sprach` is specified as [VROOT: *nahm*] and [SUFFIXCLASS: *sprechen*] licenses it to be combined with the inflectional suffixes suitable for the past tense roots of the *nehmen*-verbs, but prevents it e.g. from combining with the present suffixes for the same class. All compounds formed with `sprach` as their right constituent (e.g. `frei#sprach`, `los#sprach`) have the same feature specifications, thus these features are HEAD features as well.

$$
{}_{\textit{m-verb}}\left[\begin{array}{ll}
\text{FLEX} & {}_{\textit{vflex}}\left[\begin{array}{ll}\text{SUFFIXCLASS} & \textit{vsuffixclass}\\ \text{STEMCLASS} & \textit{vstemclass}\\ \text{VROOT} & \textit{vroot}\end{array}\right]\\
\text{DER} & \left[\text{NAT}\ \ \textit{nat}\right]\\
\text{UML} & \textit{uml}\\
\text{LINK} & \textit{link}
\end{array}\right]
\qquad
{}_{\textit{m-noun}}\left[\begin{array}{ll}
\text{FLEX} & {}_{\textit{nflex}}\left[\text{SUFFIXCLASS}\ \ \textit{nsuffixclass}\right]\\
\text{DER} & \left[\text{NAT}\ \ \textit{nat}\right]\\
\text{UML} & \textit{uml}\\
\text{LINK} & \textit{link}
\end{array}\right]
$$

Figure 40: Verbal and nominal morphological HEAD features

The above described compounding behaviour leads us to introducing separate feature structures for morphological and syntactic HEAD features. This is justified for more general reasons as well:

1. The morphological HEAD features are not relevant in syntax, i.e. in a recognition architecture where a separate morphology component is situated before the syntax and semantics component, all morphological sub-feature structures may be deleted as soon as the morphology is "finished".
2. Within the domain of morphology itself, morphological and syntactic HEAD features have a different status in inflectional affixation: Morphological HEAD features need not percolate at all, and syntactic HEAD features are passed according to the Marking Principle (not the HFP), as will be more elaborately argued in Section 5.2.3.

Returning to headedness in compounding, both morphological and syntactic HEAD features are passed from the right constituent of a compound to the compound as a whole, as can be seen from examples `frei#sprach` and `Arbeit#+s#termin` which share syntactic and morphological head properties with their right-hand constituents.

$$\left[\begin{array}{ll}
\text{SURF}|\text{PHON} & \texttt{/fr'aI\#Spr''a:x/} \\
\text{SYN} & \left[\text{LOC}\left[\text{HEAD}\ \boxed{1}\right]\right] \\
\text{MORPH} & \left[\text{HEAD}\ \boxed{2}\right] \\
\text{DTRS} & \left[\begin{array}{ll}
\text{NONHEAD-DTR} & \left[\text{SURF}|\text{PHON}\ \texttt{/fr'aI/}\right] \\
\text{HEAD-DTR} & \left[\begin{array}{ll}
\text{SURF}|\text{PHON} & \texttt{Spr'a:x} \\
\text{SYN} & \left[\text{LOC}\ \left[\text{HEAD}\ \boxed{1}\ {}_{s\text{-}vhead}\left[\begin{array}{ll}
\text{MAJ} & \textit{verb} \\
\text{TENSE} & \textit{past} \\
\text{MOOD} & \textit{indicative} \\
\text{FIN} & \textit{finite} \\
\text{AGR}_{\ agreement} & \left[\begin{array}{ll}
\text{PER} & 1 \sqcap 3 \\
\text{NUM} & \textit{singular}
\end{array}\right]
\end{array}\right]\right]\right] \\
\text{MORPH} & \left[\text{HEAD}\ \boxed{2}\ {}_{m\text{-}vhead}\left[\begin{array}{ll}
\text{FLEX}\ {}_{vflex} & \left[\begin{array}{ll}
\text{SUFFIXCLASS} & \textit{sprechen} \\
\text{STEMCLASS} & \textit{nehmen} \\
\text{VROOT} & \textit{nahm}
\end{array}\right] \\
\text{DER} & \left[\text{NAT}\ \textit{nat}\right] \\
\text{UML} & \textit{uml} \\
\text{LINK} & \textit{link}
\end{array}\right]\right]
\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 41: HEAD feature percolation in the compound `freisprach`

Note that these feature percolation properties hold as well for compounds where the right constituent is not the *semantic* head.

As a corollary of feature percolation of such features as the major class feature MAJOR, the right constituent is certainly the *distributional equivalent* of the compound as a whole, syntactically speaking. But we have to be careful since we are looking for *morphological* distributions and environments of compounds. One (and this is the only one that we can think of) would be the kind of inflectional suffixes that go with the compound. Again as a corollary of inflectional class feature percolation, a compound takes the same inflectional suffixes as its right constituent does.

The essence of the preceding paragraphs is that according to the different criteria that have previously been considered defining criteria of heads, the right constituent in a compound *is* the head. None of these criteria leads to the idea that the left constituent could be the head of a compound in German. By looking at feature percolation in compounding, we have established the feature structures defined in Figures 39 and 40 as syntactic and morphological HEAD features. In the following section, we will see that these are HEAD features in derivation, too.

$$
\begin{bmatrix}
\text{SURF|PHON} & \text{/?ARB'AIT\#+S\#TERM''I:N/} \\
\text{SYN} & \begin{bmatrix}\text{LOC}\begin{bmatrix}\text{HEAD } \boxed{1}\end{bmatrix}\end{bmatrix} \\
\text{MORPH} & \begin{bmatrix}\text{HEAD } \boxed{2}\end{bmatrix} \\
\text{DTRS} & \begin{bmatrix}
\text{NONHEAD-DTR} & \begin{bmatrix}\text{SURF|PHON} & \text{/?ARB'AIT\#+S/}\end{bmatrix} \\
\text{HEAD-DTR} & \begin{bmatrix}
\text{SURF|PHON} & \text{/TE6.M'I:N/} \\
\text{SYN} & \begin{bmatrix}\text{LOC} & \begin{bmatrix}\text{HEAD } \boxed{1} & {}_{s\text{-}nhead}\begin{bmatrix}
\text{MAJ} & \textit{noun} \\
\text{CASE} & \textit{nom} \sqcap \textit{dat} \sqcap \textit{acc} \\
\text{NUMBER} & \textit{singular} \\
\text{GEN} & \textit{masculine} \\
\text{AGR} & {}_{agreement}\begin{bmatrix}\text{PER} & \textit{3} \\ \text{NUM} & \textit{singular}\end{bmatrix}
\end{bmatrix}\end{bmatrix}\end{bmatrix} \\
\text{MORPH} & \begin{bmatrix}\text{HEAD } \boxed{2} & {}_{m\text{-}noun}\begin{bmatrix}
\text{FLEX} & \begin{bmatrix}\text{SUFFIXCLASS} & \textit{abend}\end{bmatrix} \\
\text{DER} & \begin{bmatrix}\text{NAT} & \textit{nat}\end{bmatrix} \\
\text{UML} & \textit{uml} \\
\text{LINK} & \textit{link}
\end{bmatrix}\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

Figure 42: HEAD feature percolation in the compound `Arbeitstermin`

5.2.2. Headedness in derivation

Let us next examine derivational suffixation constructions as shown in Figure 36.

The obligatory element in a derivational construction is the base rather than affix, as the result of a combination of a base with an affix yields a new base (and not a new affix). The distributional equivalent is the base for much the same reason.

In a derivation, the derivational affix has traditionally been regarded as a functor whose semantic argument is realised by the base. Consequently, the derivational affix is often regarded as being subcategorised for particular bases (cf. Selkirk (1982), Lieber (1992), and Krieger (1993)), thus, as a subcatorisand. In HPSG representations of derivations, typically the affix operates on the representation of the base, making such changes as altering its MAJOR category, its subcategorisation requirements, and deleting or adding semantic features (cf. Şehitoğlu and Cem (1996), Krieger (1993), and Riehemann (1998)). But we have mentioned alternative analyses already: Krieger (1993) proposes mutual subcategorisation of base and affix in derivations in the case of certain subregularities (discussed in Section 4.4), and we also regard the representation of umlaut properties suggested in Trost (1993) as a form of mutual subcategorisation of base and affix. Finally, Riehemann (1998) denies affixes the status of linguistic signs, and consequently they do not have subcategorisation properties at all. In her theory, constraints between affixes and bases are expressed in the word formation schemata which represent generalisations over lexicalised complex bases, see Section 4.4.

Again, as concatenative inflection is realised through suffixation only (i.e. from the right), the morphosyntactic locus in derivational suffixation is the suffix, since it is the right element by definition.

The element that determines certain morphosyntactic features of a derivation as a whole is also the suffix. It is well-known (e.g. among learners of German) that a derivational suffix determines part of speech (MAJOR), gender (in case of a nominal derivation), and the inflectional class of words formed with them. All words formed with the suffix `-ung`, for example, are feminine nouns that inflect like the simplex word `Frau`. Likewise, umlaut properties are determined by the suffix. The nominal suffix `-tum`, for example, forms neuter nouns that inflect like the simplex noun `Fach`, complete with umlaut in the plural. The features that percolate are thus the same as in compounding with the exception of ablaut features, which play no role in concatenative derivation (there are no ablauted variants of suffixes).

There is a way in which the suffix affects the shape of the base, in the case of derivational suffixes that go with umlauted or ablauted bases. But describing umlaut using governing of bases by the suffix alone leads to many exceptions and is not sufficient as Gibbon and Reinhard (1991) and Trost (1993) have shown. As for ablaut, ablauted bases that occur with particular suffixes in derivations e.g. `gang` in the word `gangbar`, are always exceptional, and a corresponding government specification using the VROOT feature (cf. Section 5.5.2) in the subcategorisation requirements of the suffix would lead to numerous incorrect predictions.

In sum, three of seven criteria point to the suffix as the head of a derivation (semantic functor, morphosyntactic locus, feature percolation), two point to the base (obligatory element, distributional equivalent), and two remain ambiguous (subcategorisand, governor). From the viewpoint of HPSG (and the cited previous work in generative morphology), though, we consider the morphosyntactic locus and feature percolation, especially of the MAJOR feature, more important criteria to determine heads. We have already seen that the criteria of obligatory elements and distributional equivalents do not distinguish between the two constituents of a compound, and thus we assume that they play a less important role in determining heads in derivation, too. Thus, like compounds, derivational suffixations are right-headed in German. Let us next check derivational prefixation constructions for headedness.

The base is the obligatory element and the distributional equivalent in derivational prefixations for the same reason the suffix is the obligatory element and the distributional equivalent in derivational suffixations. On the other hand, the prefix is the semantic functor and the subcategorisand in derivational prefixations for the same reason the suffix is the semantic functor and the subcategorisand in derivational suffixations. There is no government in derivational prefixations, i.e. no cases where the prefix chooses allomorphic variants of the base, and (morphologically conditioned) allomorphs of prefixes do not exist.

Again, as in German inflectional affixes are affixed from the right to a base, the base is the morphosyntactic locus in derivational prefixations.

Morphosyntactic feature percolation is also from the base in derivational prefixations. For example, the derived words `er+brechen, zer+brechen, ab+brechen, unter+brechen` are all verbs and inflect exactly like the simplex verb `brechen` (i.e. they have the same MAJOR, VROOT, STEMCLASS and SUFFIXCLASS feature specifications). Prefixations with the nominal class II-prefix `un-` and the semi-prefixes `haupt-` and `erz-` such as `Un#mensch, Haupt#stadt, Erz#rivale` reveal that the feature GENDER also percolates from the base, in these cases from `Mensch, Stadt`, and `Rivale`.

A significant group of verbal prefixations, though, which result in a change of the major class category to *verb*, seem to involve no feature percolation from the base at all (cf. Figures 43, 44, and 45). There is also no obvious source of the inflectional class specifications of the derived verbs, which are always characterised as weak verbs ([STEMCLASS *fragen*] and [SUFFIXCLASS *sagen*], [SUFFIXCLASS *warten*], [SUFFIXCLASS *begeistern*], or [SUFFIXCLASS *fassen*]).

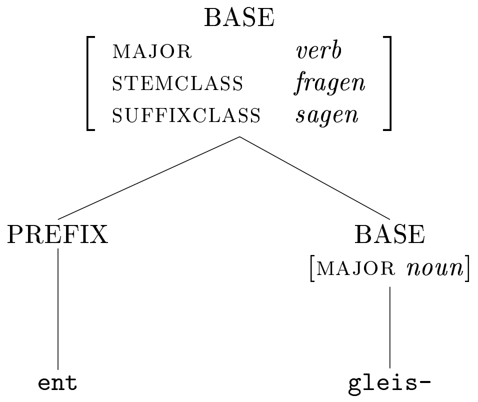


Figure 43: Absent feature percolation in the constituent structure of `entgleis-`

However, an alternative analysis of these prefixations is available, by regarding `gleis-, dumm-`, and `antrag` as (non-lexicalised) bases that have been *converted* by the morphology from nominal or adjectival to verbal before the actual prefixation.

In an analysis as exemplified by the alternative tree structure of `ent+gleis-` in Figure 46, on the level of the `ent`-prefixation, the verbal features percolate from the base just like in the examples in Figure 35 and an RHR can be established for prefixations in German as well. Such an analysis involving conversion has previously been proposed e.g. by Lüdeling (1999), who mentions further arguments in favour of it (cf. Lüdeling, 1999, p.76f.):

1. A conversion analysis is independently required for a substantial number of de-

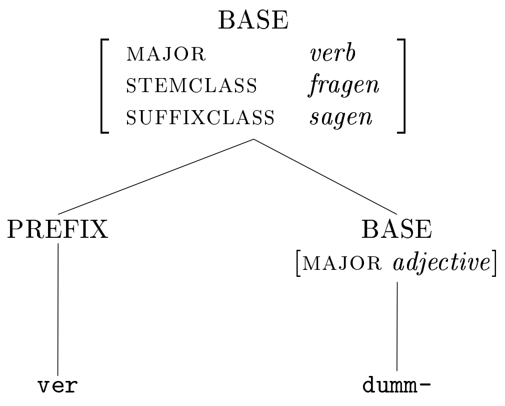


Figure 44: Absent feature percolation in the constituent structure of `verdumm-`

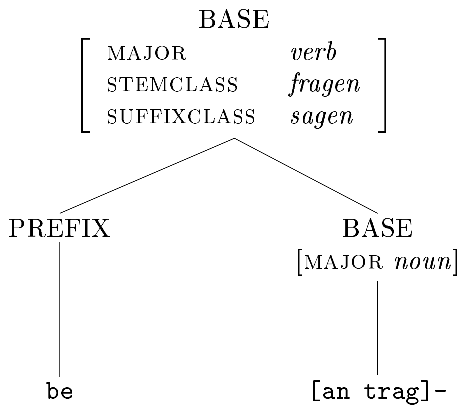


Figure 45: Absent feature percolation in the constituent structure of `beantrag-`

nominal and de-adjectival simplex verbs in German (like `haus-`, `löffel-`, `grün-`), thus no additional rule is required in the morphology..

2. If we rejected a conversion analysis, what about the 'normal' verbal prefixations such as those in Figure 35? Certainly the base should be the head in these, for the reasons given above. But then we would get different head assignments in derivations like those in Figure 35 and those in Figures 43-45. This would divide prefixations into two fundamentally different morphological construction types, which is not desirable.
3. If the base were not the head in Figure 46, the verbal information types would have to come from the prefix (the same way they percolate from suffixes in derivational suffixations), but no evidence for this can be found in German morphology.

In sum, we propose a Right-hand Head Rule for word formation in German. This

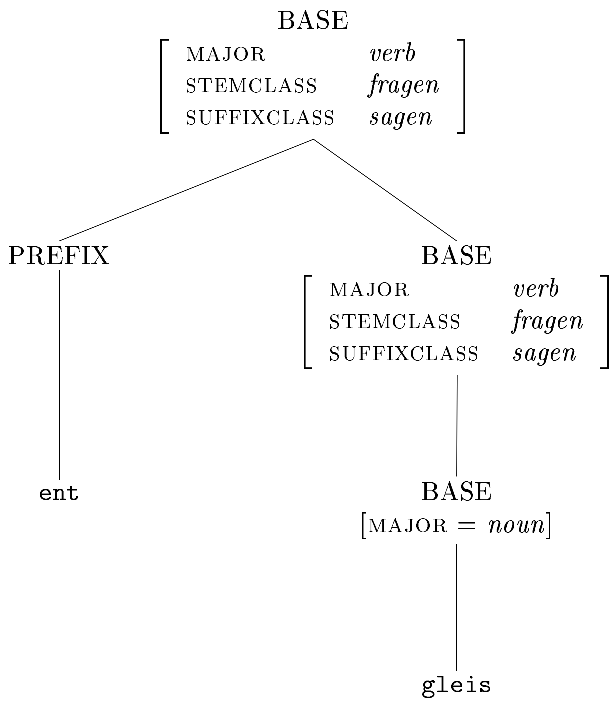


Figure 46: Alternative constituent structure of `entgleis-`

means that the HFP basically remains as it is has been introduced by Pollard and Sag (1994), only, as we have argued for distinguishing syntactic vs. morphological (HEAD) features, we must make sure that it applies to both (Figure 47). Henceforth it will be called the Morphological Head Feature Principle (MHFP).

$$
\mathit{baseComplex} \Rightarrow \begin{bmatrix} \text{SYN|LOC|HEAD} & \boxed{1} \\ \text{MORPH|HEAD} & \boxed{2} \\ \text{DTRS} & {}_{\mathit{headed\text{-}structure}}\begin{bmatrix} \text{HEAD-DTR} & {}_{\mathit{base}}\begin{bmatrix} \text{SYN|LOC|HEAD} & \boxed{1} \\ \text{MORPH|HEAD} & \boxed{2} \end{bmatrix} \end{bmatrix} \end{bmatrix}
$$

Figure 47: The Morphological Head Feature Principle (MHFP, final version)

The Continuation Schema to be introduced in Section 5.3.4 ensures that the head appears indeed as the right-hand constituent in derivation and in compounding constructions. A conversion schema naturally inherits neither from a continuation schema, nor from the MHFP (it does not have a HEAD-DTR).

5.2.3. The non-headedness of inflection

Contrary to the lexical rule approaches by Pollard and Sag (1987, 1994) and the paradigmatic approaches by Erjavec (1993) and Kathol (1999), all conducted within the HPSG framework, we assume that inflectional morphemes are signs, as they have segmentable and classifiable PHON and SYN properties. We thus follow Selkirk (1982), Lieber (1992), Di Sciullo and Williams (1987), and van Eynde (1994), who likewise propose item-and-arrangement approaches to inflectional morphology, the latter also using the HPSG framework. In Section 5.5.3, however, we will show how a paradigm theory is part of our morphology through the organisation of the type hierarchy over stems and inflected words. (We talk about *stems* and not *bases* in this section, as we have pointed out that stems are exactly those bases that may be combined with an inflectional suffix.)

While Di Sciullo and Williams (1987) come to the conclusion that the inflectional affix is always the head in a Stem-Infl construction, Selkirk (1982) (cf. Section 2.3), and van Eynde (1994) claim that the stem must be the head.

Let us review their arguments and check the potential head property assignments in inflected words. We define inflected words as words that are morphosyntactically fully specified and can thus freely occur, i.e. can serve as input to the syntax, such as `Termin#+e, be+sprach, Vater, Väter`. The concept of inflectedness is thus independent of the presence of an inflectional suffix, the criterion is the presence of full morphosyntactic feature specifications (Figures 41, 42). The lexical type subsuming all inflected words is called *word*, and according to the internal structure of inflected words, it has the two subtypes *stem-free*, and *inflected* (Figure 48). *Stem-free* subsumes those inflected words that are bare morphological stems at the same time, i.e. that do not need an inflectional suffix to form a syntactic atom, such as `be+sprach, Termin, Vater, Väter`. The other subtype, *inflected*, comprises those inflected words that bear an inflectional suffix. In the former, head assignment is clear, i.e. is according to the head principles for complex bases as discussed above. The latter will be examined in the following.

The obligatory element in an inflected word has to be the stem. This claim rests on the fact that bare stems can easily be syntactic atoms in German (cf. Figures 41 and 42). The distributional equivalent of an inflected word is the stem for the same reason.

The stem may also be viewed as the governor of the inflectional suffix, governing the shape of it according to its own inflectional class. We may formulate that e.g. the stem `termin` governs the plural morpheme in that it demands that its shape be `-e` and not e.g. `-er`. At the same time, the inflectional suffix may be viewed as the governor of the stem, constraining the shape of the stem either to be the base stem or an umlauted or ablauted variant. Note that both kinds of conditioning would be purely morpho-lexical (and not phonological).

It probably makes no sense to look for the morphosyntactic locus in a Stem-Infl

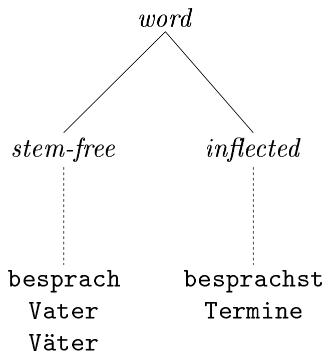


Figure 48: Part of the type hierarchy for *word*

combination as its only point is the realisation of morphosyntactic categories anyway. If anything has to be decided on, the stem is the morphosyntactic locus.

As for feature percolation, the features that have been identified as HEAD features so far percolate partly only from the stem, partly only from the suffix, partly they may be percolating from either. In the example of `Termin#+s` in Figure 49, GENDER definitely percolates from the stem, CASE definitely percolates from the suffix, and UML, LINK, MAJOR, SUFFIXCLASS, NUMBER may be percolating from the stem *or* the suffix. (The GENDER feature in Figure 49 is unambiguously percolating from the stem, as the *masculine/neuter* distinction is not a type-dividing dimension in the hierarchy of inflectional paradigms, cf. Section 5.5.3, and thus is not appropriate for suffixes that go with masculine or neuter nouns in German.) The question of HEAD feature percolation in inflection will be discussed in detail in the following.

It is typical of past tense inflected weak verbs that *two* inflectional suffixes occur in one inflected word. Here, the verbal HEAD feature TENSE percolates from the first inflectional suffix `-t-`, and the agreement features PERSON and NUMBER percolate from the second inflectional suffix, regardless of whether such constructions are analysed as one ternary branching structure (Figure 51), or as a binary branching structure embedded in another binary branching structure (Figure 52).

These cases of percolation of morphosyntactic features from multiple sources are exactly those that led Selkirk (1982) to reject inflectional affixes the status of heads in English and define feature percolation conventions for non-heads instead (pp.74ff). (Di Sciullo and Williams, 1987, p.26) reacted by revising their earlier RHR and formulating the *Relativised Right-hand Head Rule* (RRHR):

> Definition of head$_F$ (read: head with respect to the feature F):
> The head$_F$ of a word is the rightmost element of the word marked for the feature F.

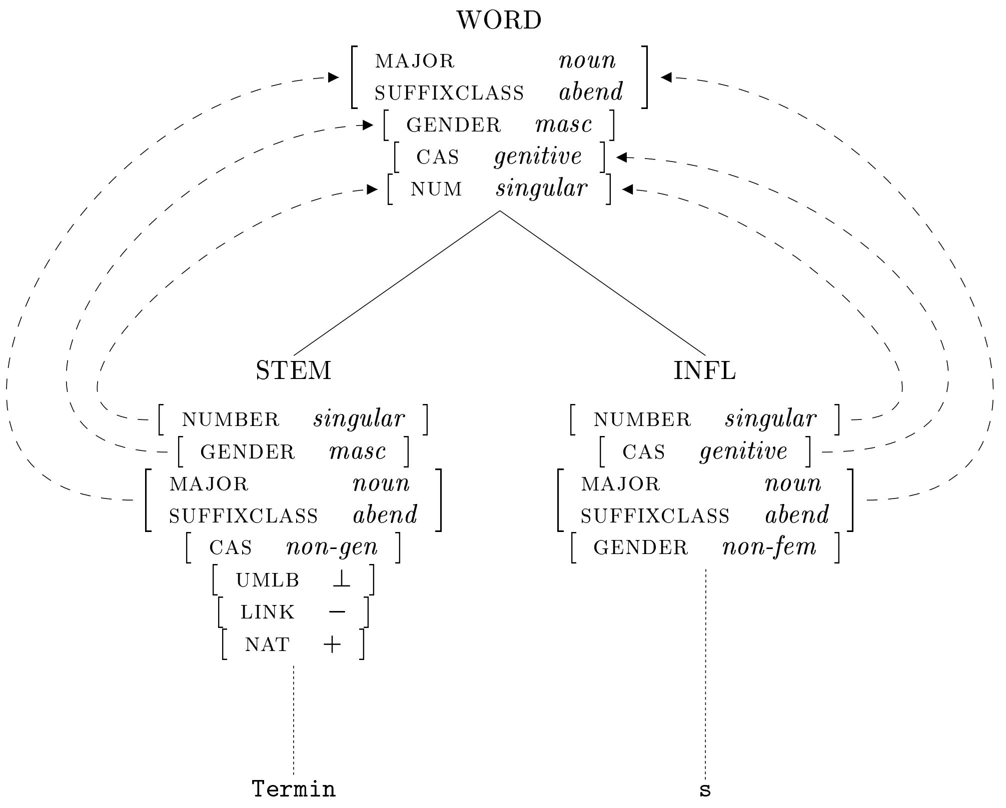


Figure 49: Potential paths of feature percolation in inflection

This Revised RHR is needed for English and German because of the facts of multiple sources of feature percolation in inflection, but (Di Sciullo and Williams, 1987, p.26f.) also cite the example of the Spanish diminutive suffix `-ita` which “can attach to almost any part of speech and [...] the resulting word belongs to the same category as the word to which the diminutive attaches”.

Let us see what arguments Selkirk (1982) and van Eynde (1994) have for regarding the stem as the head of a Stem-Infl construction. (Selkirk, 1982, p.77) simply says that “[i]f the inflectional affix is not the head, then its sister category is”. As no contradiction between the MAJOR feature of the sister category (the stem) and the inflected word as a whole arises, she deems this a sufficient criterion for the time being. She remarks that a more precise characterisation of the notion of inflection is needed to settle this question (p.77).

In (van Eynde, 1994, p.77), the argumentation runs as follows: Italian participles take adjectival inflectional suffixes but as a whole they are verbs when for example used to express perfect tense. Therefore the category (major class) feature must percolate from the stem, where it is always compatible. Consequently, the stem must be the head.

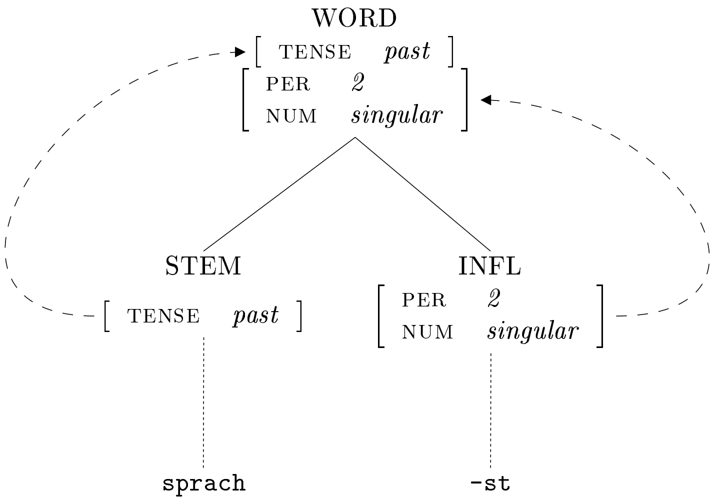


Figure 50: Morphosyntactic feature percolation from two sources, stem and inflectional suffix

However, this argumentation cannot be carried over to German, because participles do not inflect for gender and number when they function as verbs.

Some authors consider participles in German (or English) as a proof that inflectional suffixes can be major class/category-changing and must therefore be heads just like derivational suffixes, but again, we think that a conversion analysis for participles represents a higher generalisation about the behaviour of both present and past participles in German.

In sum, we agree with the view that inflectional suffixes cannot be heads, but we are not convinced, either, that stems must be the heads in inflectional suffixations, because no uniform principle of feature percolation can be formulated for inflection. But instead of claiming multiple headedness (like in the RRHR), we suspect that we are not dealing with heads in inflection at all, but have HEAD feature percolation from different (non-head) elements instead, i.e. contrary to Di Sciullo and Williams (1987), we believe that only elements from which *all* the HEAD features percolate at the same time, should be called heads.

To illustrate this, we would like to discuss two further aspects at this point: Firstly, is a Head Feature Principle the right way of describing feature percolation in inflection, and, having answered this question to the negative, is inflection about feature percolation from a single source at all? Or is it about different kind of compositionality of feature specifications?

We have mentioned that feature percolation is a reflection of the compositionality principle, i.e. a word form has properties that are a function of the properties of its

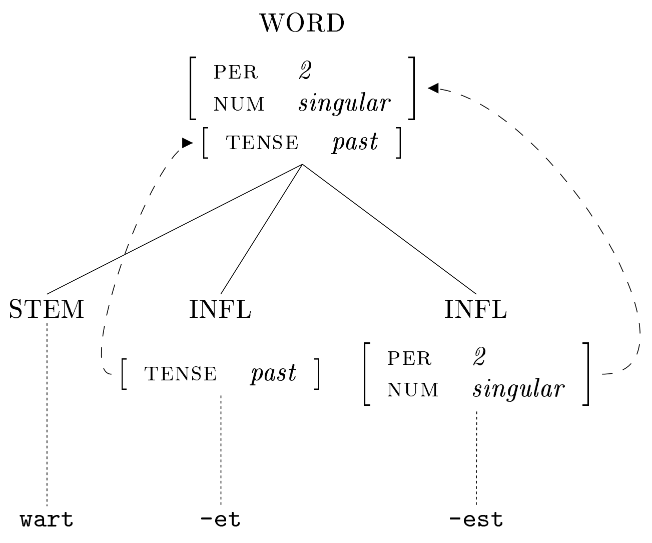


Figure 51: Ternary branching structure for past tense inflection of weak verbs

parts. But how do we decide in the first place which properties (feature specifications) a complex construction bears? The answer is that we check to which properties reference is made when the construction in question is embedded in higher level constructions.

For example, a form like `Arbeit#+s#platz` has the morphological properties of inflectional suffixclass, umlauting behaviour and suitability of linking morphemes, which are identical to those of the root `Platz`. Similarly, the syntactic properties of number, gender, and case that manifest themselves when `Arbeitsplatz` is embedded in an NP, are identical with those of `Platz`. But an inflected form like `Arbeit#+s#platz#+es` is no longer available to the morphology, it can *only* serve as an input to the syntax. Thus, there is neither a reason nor a discovery procedure for *morphological* HEAD feature specifications for words inflected by an inflectional suffix. We believe that it is the lack of a possibility for Stem-Infl constructions to be directly or indirectly embedded in themselves (nor even to be embedded in morphological constructions of any kind) that makes it difficult to talk about morphological headedness in such constructions.

The next question is whether inflection is always about feature percolation at all. Claiming that a word form like `Platz#+es` is specified for/bears/has the feature specification CASE=*genitive*, because the suffix `-es` is specified for it/has it/bears it is counterintuitive when that is the only point of the existence of `-es`, i.e. the only point of the affixation of `-es` is to *provide* this specification for a lexeme like `Platz`. In the least it is a different sense of being specified for certain features. Rather than saying these feature specifications *percolate* from `-es`, we would want to formulate that `-es` *marks* `Platz` for CASE=*genitive*. Lieber (1992) and van Eynde (1994) have developed their theories along these lines. Van Eynde (1994) claims that affixal inflection is best described in terms

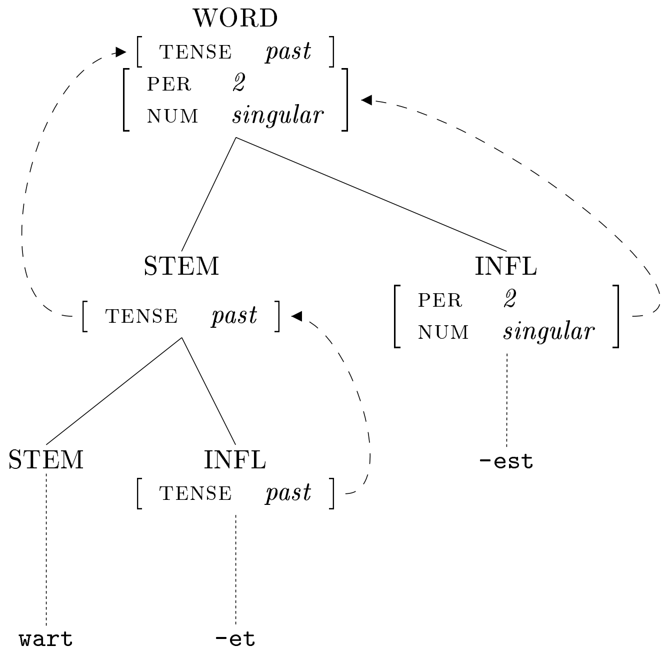


Figure 52: Embedded binary branching structure for past tense inflection of weak verbs

of the *head-marker schema* which was originally introduced in (Pollard and Sag, 1994, p.45f.) for $\bar{\text{S}}$ constructions (Figure 53).

Pollard's and Sag's motivation for markers in syntax lies in the ability of matrix verbs to select for verbal inflectional features of the $\bar{\text{S}}$, which only works if S is the head of $\bar{\text{S}}$. For example, in the sentence *I demand that John leave immediately*, the subjunctive form of *leave* is selected by the matrix verb *demand*. On the other hand, *demand* also selects the complementiser *that* as such, and *that* in turn constrains the S it combines with to be finite.

A complementiser like *that* or *for* is therefore considered a marker, and S the head in $\bar{\text{S}}$ constructions. Markers have head-like properties: first, they select features of the constituents they combine with, formulated as the SPEC principle for functional parts of speech given in 5.4, and second, they share certain features with their mother constituents, formulated as the Marking Principle given in 5.5.

5.4 Specifier Principle *In a headed phrase whose non-head daughter has a* SYNSEM|LOCAL|CATEGORY|HEAD *value of sort 'functional', the* SPEC *value of that value must be token-identical with the phrase's* DAUGHTERS|HEAD-DAUGHTER|SYNSEM *value* (van Eynde, 1994, p.50).

5.5 Marking Principle *In a headed phrase, the* MARKING *value is token-identical with that of the* MARKER-DAUGHTER *if any, and with that of the* HEAD-DAUGHTER *otherwise* (van Eynde, 1994, p.51).

$$\begin{bmatrix} \text{SYNSEM|LOC|CAT} & \begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SUBCAT} & \boxed{2}\\ \text{MARKING} & \boxed{4}\end{bmatrix} \\ \text{DTRS} & {}_{\textit{head-marker-struc}}\begin{bmatrix} \text{MARKER-DTR|SYNSEM|LOC|CAT} & \begin{bmatrix}\text{HEAD} & {}_{\textit{mark}}\begin{bmatrix}\text{SPEC} & \boxed{3}\end{bmatrix}\\ \text{SUBCAT} & \langle\,\rangle \\ \text{MARKING} & \boxed{4}\ \textit{marked}\end{bmatrix} \\ \text{HEAD-DTR|SYNSEM} & \boxed{3}\begin{bmatrix}\text{LOC|CAT} & \begin{bmatrix}\text{HEAD} & \boxed{1}\\ \text{SUBCAT} & \boxed{2}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 53: AVM notation of the Head-marker schema after (Pollard and Sag, 1994, p.46) and (van Eynde, 1994, p.50)

In the head-marker schema (Figure 53), firstly, properties of the HEAD-DTR are constrained by the SPEC specification of the MARKER-DTR (Specifier Principle, cf. tag $\boxed{3}$). Secondly, the phrase shares the MARKING features with its MARKER-DTR (Marking Principle, cf. tag $\boxed{4}$). In addition, the HFP applies as usual (tag $\boxed{1}$). The SUBCAT lists of the HEAD-DTR are shared (tag $\boxed{2}$), although the Subcategorisation Principle applies as usual. This is because the phrase has no COMP-DTR, i.e. it is specified for [DTRS|COMP-DTR<>]. The Marking principle is thus a kind of Head Feature Principle for markers, and "the specifier principle can be seen as the functional counterpart of the subcategorisation principle" (van Eynde, 1994, p.40).

Pollard and Sag (1994) give no other examples of markers, but (van Eynde, 1994, p.54f.) identifies further types of markers in syntax, such as case-marking prepositions,[34] the *to*-infinitive marker, and auxiliaries. According to (van Eynde, 1994, p.78), inflectional suffixes are markers as well, and the stems they are suffixed to are the heads of Stem-Infl structures which are thus classified as head-marker structures.

In the Marking Principle two cases are distinguished, one special case for head-marker structures, and one default case for all other headed structures. It is thus different in character from other principles in HPSG such as the HFP or the subcategorisation principle, which are not formulated in terms of default and exceptional cases. It has the same logical structure as the RRHR introduced by Di Sciullo and Williams (1987), and head and backup percolation introduced by Lieber (1992) (cf. Section 2.3.2) to describe inflection. Thus, van Eynde (1994) is right in regarding the head-marker schema a good

[34] i.e. prepositions in PPs that serve as prepositional objects, and which were formerly supposed to bear a feature named PFORM, cf. Bresnan (1982); Pollard and Sag (1994).

$$\begin{bmatrix}
\textsc{phon} & \langle \text{SPRACH,ST} \rangle \\
\textsc{synsem}|\textsc{loc}|\textsc{cat} & \begin{bmatrix} \textsc{head} & \boxed{1} \\ \textsc{subcat} & \left\langle \boxed{2} \right\rangle \\ \textsc{marking} & \boxed{4} \end{bmatrix} \\
\textsc{dtrs} & {}_{\textit{head-marker-struc}}\begin{bmatrix}
\textsc{head-dtr} & \begin{bmatrix} \textsc{phon} & \langle \text{SPRACH+} \rangle \\ \textsc{synsem}|\textsc{loc}|\textsc{cat} & \begin{bmatrix} \textsc{head} & \boxed{1}\ \textit{vhead} \\ \textsc{subcat} & \left\langle \boxed{2}\text{NP} \right\rangle \\ \textsc{marking} & \textit{unmarked} \end{bmatrix} \end{bmatrix} \\
\textsc{marker-dtr}\ \boxed{3} & \begin{bmatrix} \textsc{phon} & \langle \text{+ST} \rangle \\ \textsc{synsem}|\textsc{loc}|\textsc{cat} & \begin{bmatrix} \textsc{head} & {}_{\textit{mark}}\begin{bmatrix} \textsc{spec}\ \boxed{3}\ \textit{verb} \end{bmatrix} \\ \textsc{marking}\ \boxed{4} & \begin{bmatrix} \textsc{fin} & \textit{finite} \\ \textsc{tense} & \textit{present} \\ \textsc{mood} & \textit{indicative} \\ \textsc{agr} & \begin{bmatrix} \textsc{per} & \textit{2} \\ \textsc{num} & \textit{sing} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}$$

Figure 54: Head-marker analysis of `sprach+st` in the line of van Eynde (1994)

candidate to describe Stem-Infl structures i.e. in regarding Stem-Infl structures as a subtype of head-marker structures.

But the Marking Principle also seems to redundantly include the effect of the HFP for structures that are not head-marker structures, i.e. the default case: For these it does not matter whether a feature is a HEAD feature or a MARKING feature, in either case it will be shared with the head of the structure.

Let us have a closer look at the interaction between the Marking Principle and the HFP in Stem-Infl constructions. The stem is supposed to be the head in such constructions, but the HFP would hardly have any effect since the morphosyntactic features PERSON, TENSE, MOOD, NUMBER, CASE would have to be MARKING features and would be passed by the Marking Principle. The remaining, morphological HEAD features STEMCLASS, SUFFIXCLASS, VROOT, UMLAUT, LINK, and NAT, which we have identified in Sections 5.2.2 and 5.2.1, may still fall under the HFP, but their percolation is totally dispensable at this stage of the morphology, as we have argued above. The only nominal feature that could be a HEAD feature and *must* percolate, since it is needed in the syntax, is GENDER, but it could easily be a MARKING feature as well and be passed by the Marking Principle.

If we now accept the marking analysis for inflection, we are in a situation where we claim the morphosyntactic features PERSON, TENSE, MOOD, NUMBER, CASE, GENDER to be HEAD features in word formation, MARKING features in inflection, and again HEAD features in syntax. Van Eynde (1994) deems these features MARKING features in word formation and in syntax, too, but the latter makes it difficult to define an interface to much of the other work on syntax in HPSG, where they are the typical HEAD features. In other words we face the following dilemma:

1. We want to keep the definition of STEMCLASS, SUFFIXCLASS, VROOT, UML, LINK, NAT as morphological HEAD features and MAJOR, PERSON, TENSE, MOOD, NUMBER, CASE, GENDER as syntactic HEAD features, since that fits well with the analyses in word formation and syntax described so far.
2. We want to have a marker analyses for the provision of morphosyntactic HEAD features by inflectional suffixes in Stem-Infl constructions.

As already indicated, the solution is to reject the idea that Stem-Infl constructions are headed structures. In the Stem-Infl marker constructions, morphosyntactic HEAD features are "collected" from multiple sources, some are marked by the markers, some are passed from the non-marker. The stem in these constructions is the outstanding constituent in terms of syntactic (major class, subcategorisation requirements) and semantic properties, which might lead some to call it the head, but it cannot be the head on account of HFP feature percolation, which we consider as crucial. (Pollard and Sag, 1994, p.397, footnote 6) leave open the question whether there are un-headed structures other than *coord-struc*, thus we feel free to suggest here such an un-headed structure called *marker-struc*, representing Stem-Infl construction where the MARKER-DTR is the inflectional suffix and the NONMARKER-DTR is the stem. The *head-marker-struc* as introduced by Pollard and Sag (1994) for $\bar{\text{S}}$ with complementisers may be a subtype of *marker-struc*.

The following modifications to the Head-Marker Schema yield our new Inflectional Marking Principle shown in Figure 55:

1. The Inflectional Marking Principle states that the marker's MARKING features mark (i.e. specify, insert) values of syntactic HEAD features on the mother constituent, thus all MARKING features will be HEAD features (but not the other way round). The IMP also states that the nonmarker's NONMARKING features are shared with those of the mother constituent. Thus the domains of the HFP and the IMP are properly distinguished.
2. Instead of the Specifier Principle, which is defined for headed structures, we need only employ morphological subcategorisation (Section 5.2.4).
3. As before, we employ only SYN and not SYNSEM and CAT.

A new marker schema for the lexical type *inflected*, comprising the effect of all principles relevant in inflectional suffixation, will be presented in Section 5.5.2.

$$\textit{inflected} \Rightarrow \left[\begin{array}{ll} \text{SYN}|\text{LOC}|\text{HEAD} & {}_{s\text{-}head}\left[\begin{array}{ll} \text{MARKING} & \boxed{1} \\ \text{NONMARKING} & \boxed{2} \end{array}\right] \\ \text{DTRS} & {}_{marker\text{-}struc}\left[\begin{array}{ll} \text{MARKER-DTR}|\text{SYN}|\text{LOC}|\text{HEAD}|\text{MARKING} & \boxed{1} \\ \text{NONMARKER-DTR}|\text{SYN}|\text{LOC}|\text{HEAD}|\text{NONMARKING} & \boxed{2} \end{array}\right] \end{array}\right]$$

Figure 55: The Inflectional Marking Principle (IMP)

$${}_{s\text{-}vhead}\left[\begin{array}{ll} \text{MARKING} & {}_{v\text{-}marking}\left[\begin{array}{ll} \text{TENSE} & \textit{tense} \\ \text{MOOD} & \textit{mood} \\ \text{FIN} & \textit{finiteness} \\ \text{AGR} & {}_{agreement}\left[\begin{array}{ll} \text{PER} & \textit{person} \\ \text{NUM} & \textit{number} \end{array}\right] \end{array}\right] \\ \text{NONMARKING} & {}_{v\text{-}nonmarking}\left[\text{MAJ} \quad \textit{verb}\right] \end{array}\right]$$

Figure 56: Verbal syntactic marking and nonmarking HEAD features

Which are the MARKING features, eventually? As we said above, they are the morphosyntactic HEAD features FIN, TENSE, MOOD, CASE, AGR, i.e. PERSON and NUMBER, and DECL for strong vs. weak adjective declension. GENDER seems to be a NONMARKING feature with nouns but a MARKING feature with adjectives. Also TENSE may be a NONMARKING feature with most strong verbs (cf. Figure 54), but since verbal suffixation is determined by verbal root selection through the VROOT feature, it is sufficient to regard TENSE as a marking feature always provided by the inflectional suffix. Ideally we would need a *default unification* where HEAD|MARKING features always override HEAD|NONMARKING features in case of conflicts, and in order to mirror the RRHR more directly in HPSG feature structure terms; this would also obviate the need to explicitly inventorise the NONMARKING features. But note that in the original head-marker approach by van Eynde (1994) we would have to explicitly sort out the HEAD and MARKING features, too, thus this is not a disadvantage of our non-HFP account of inflectional suffixation. For the time being, GENDER will have a different status in the feature declarations of the different subtypes *n-marking, a-marking* and *n-nonmarking, a-nonmarking*, i.e. the value types of MARKING and NONMARKING, respectively.

$${}_{s\text{-}nhead}\left[\begin{array}{ll} \text{MARKING} & {}_{n\text{-}marking}\left[\begin{array}{ll} \text{CASE} & \textit{case} \\ \text{AGR} & {}_{agreement}\left[\begin{array}{ll} \text{PER} & \textit{person} \\ \text{NUM} & \textit{number} \end{array}\right] \end{array}\right] \\ \text{NONMARKING} & {}_{n\text{-}nonmarking}\left[\begin{array}{ll} \text{MAJ} & \textit{noun} \\ \text{GENDER} & \textit{gender} \end{array}\right] \end{array}\right]$$

Figure 57: Nominal syntactic marking and nonmarking HEAD features

5.2.4. Subcategorisation

A morphology that considers affixes as lexical signs involves two kinds of subcategorisation information, cf. Section 4.4.

1. Morphological subcategorisation: All affixes, and only affixes, are subcategorised for morphological bases in that they select morphological, syntactic, semantic, or sometimes phonological features of these.
2. Syntactic subcategorisation: lexical stems have a syntactic subcategorisation frame that they share with all their inflected forms. This is what is known as the SUBCAT list of lexical entries in HPSG grammars. They may also play a role in morphology: Firstly, one of the elements of a head's subcategorisation list might be satisfied by the modifier in a compound, examples of this have been discussed in Section 5.2.1. Secondly, the SUBCAT list of a morphological base might be altered through derivation or compounding, i.e. the syntactic SUBCAT list of the complex structure is a function of the SUBCAT list of the base. The nature of that function might be specified in the feature structure representing the affix or modifier, or in the morphological construction type that describes the Affix-Base or the Head-Modifier combination (e.g. according to Riehemann's (1993, 1998, 2001) schema-approach). The function might imply only modifications of the syntax, but not the semantics, as is frequently the case in compounding (compare *Treffen* and *Donnerstagstreffen*), but also, for example, in agent noun derivations (*lehren* $\rightarrow$ *Lehrer*).

Within the confines of this thesis, we will deal with morphological subcategorisation only. All types of affixes are appropriate for the feature MORPH|SUBCAT, whose value is a single feature structure description (not a list of feature structure descriptions) and which is typed to *base* (see the collection of feature declarations in Section 5.6 for the complete system). There is no equivalent to the cancelling of elements from a subcategorisation list as in the HPSG subcategorisation principle for syntax, instead, morphological subcategorisation requirements must always be completely satisfied in one Affix-Base combination, thus a *lexComplex* construction is never appropriate for the feature MORPH|SUBCAT.

Consequently, our Morphological Subcategorisation Principle (MSP) looks like the following:

$$\textit{affixed} \Rightarrow \quad \begin{bmatrix} \text{DTRS} {}_{\textit{struc}}\begin{bmatrix} \text{\$BASE-DTR} & \boxed{1} \\ \text{\$AFFIX-DTR|MORPH|SUBCAT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Figure 58: The Morphological Subcategorisation Principle (MSP)

\$BASE-DTR ranges over HEAD-DTR (in case the affix is a prefix), NONHEAD-DTR (in case the affix is a derivational suffix), and MARKING-DTR (in case the affix is an inflectional suffix). The conditions for the respective instantiations will be clarified in Section 5.5.1.

5.3. The hierarchy of morphological types

In this chapter, we present the organisation of lexical objects into an HPSG type hierarchy according to the morphological properties discussed so far. Several partial inheritance hierarchies covering various aspects of morphology that were proposed earlier have been discussed in this thesis. Pollard and Sag (1987) classify English word forms according to their morphosyntactic properties, Riehemann (1998) and Krieger et al. (1993) contain classifications of German *bar*-adjectives, Bleiching (1994) and Cahill and Gazdar (1999) classify nouns according to inflectional classes in German. In Lüngen and Sporleder (1999) a morpheme type hierarchy for German is presented, and Fischer (1993) and Gibbon (1997) present different hierarchies of English compound noun classes. It has not been quite clear, however, how such hierarchies could be integrated into one large, comprehensive hierarchy, or whether they can or should be integrated at all for the purpose of describing the morphology of one language completely. Since that is one of the enterprises of this thesis, the lexicon suggested here is organised into different morphological subhierarchies, the top-levels of which are shown in Figure 59.

The immediate partition of the type *sign* is into the subtypes *complex*, *morpheme*, *lemma*, and *word*. It is different from the partition into *phrasal* and *lexical* put forward in Pollard and Sag (1987) and Pollard and Sag (1994), as we postulate the existence of complex lexical signs, which means that *phrasal* and *lexical* together no longer exhaust the type *sign*. Instead, we propose that *phrasal* is a subtype of *complex* which explicitly denotes a type subsuming *syntactically* complex signs such as NPs and VPs. The types *morpheme, lemma, word,* as well as *lexComplex* are appropriate for our MORPH feature, so these are the ones subsumed under *lexical-sign*. *Lexical-sign* thus denotes lexical objecthood, not to be confused with lexicalisedness. The constructions in our morphology describe lexicalised and non-lexicalised complex words alike, the original LEX feature from HPSG should serve to distinguish complex words according to lexicalisedness. We do not employ this feature in our present morphology, though.

The actual morphology hierarchy (i.e. the one under *lexical-sign*) consists of four main subhierarchies: the *lemma* hierarchy, the *word* hierarchy, the *lexComplex* hierarchy, and the *morpheme* hierarchy.

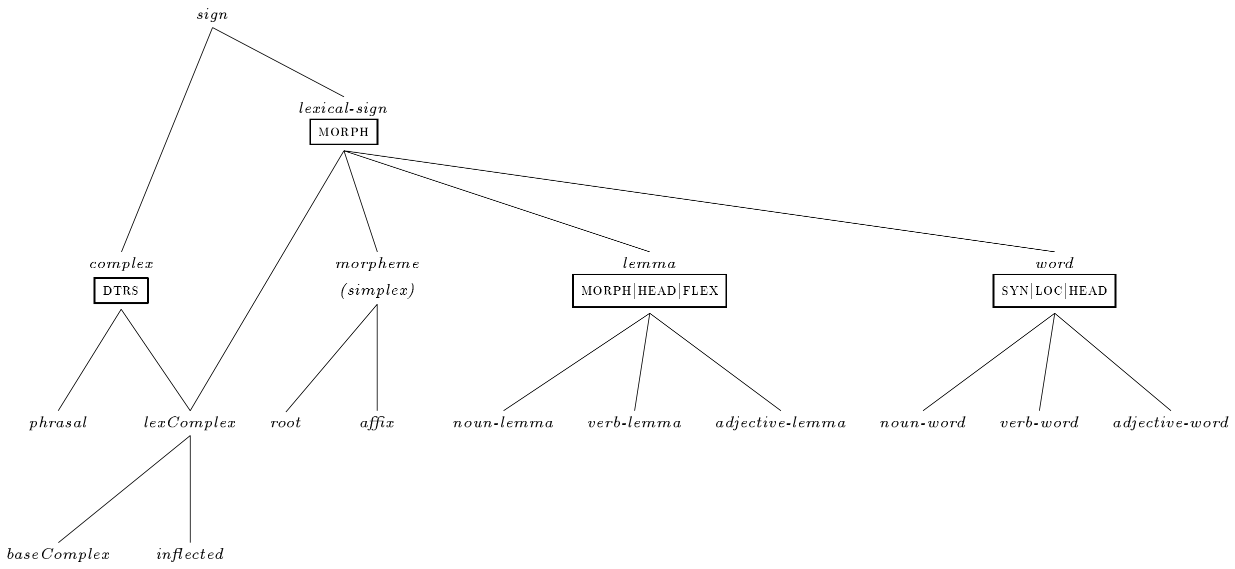


Figure 59: The hierarchy of morphological types

5.3.1. The lemma hierarchy

A *lemma* contains all the morphological information that is common to the word forms belonging to one morphological paradigm. For example, the stems (represented by the orthographic string that is left when the inflectional suffix is stripped off, i.e. preceding the `#+`) of the word forms `schreib#+e, schreib#+st, schreib#+t, schreib#+en, schrieb, schrieb#+e, schrieb#+st, schrieb#+en, geschrieb#+en` are all appropriate for the features STEMCLASS and SUFFIXCLASS that are also found under a more abstract type called *schreiben-lemma*, which is one of the subtypes of *lemma*. Note that we reserve the term *lexeme* for an extended notion of *lemma.* Further syntactic and semantic types of information are associated with e.g. a *lexeme* called *schreiben-lexeme*, e.g. that it is a transitive verb, that the present and past perfect analytical tense constructions are formed with the auxiliary *haben*, that its semantics is a binary relation represented by 'WRITE(WRITER,WRITTEN)' etc. These other types of information are organised similarly in different hierarchies, from which the lexeme node then multiply inherits, that is, *lexeme* implies a partition of lemmata along syntactic and semantic dimensions. The type *lemma*, on which we focus here, however, is appropriate only for morphological types of information, and in turn, only morphological information is used to distinguish between individual homonymic lemmata. There is for example only one noun lemma type called *schloß-lemma* even though two different lexemes will eventually be subsumed by it. On the other hand, there will be two lemmata *bank-lemma-1* and *bank-lemma-2*, as these are morphologically distinguished by their SUFFIXCLASS specifications (*noun-kraft* and *noun-frau*, respectively).

The feature SUFFIXCLASS induces a nominal paradigm class hierarchy on noun lemmata, through partitions and re-partitions of its value type *n-suffixclass* (Figures 60-62).

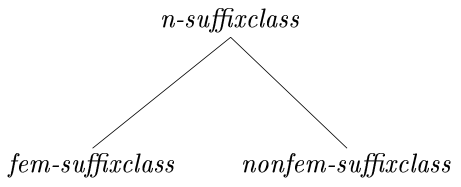


Figure 60: Partition of *n-suffixclass*

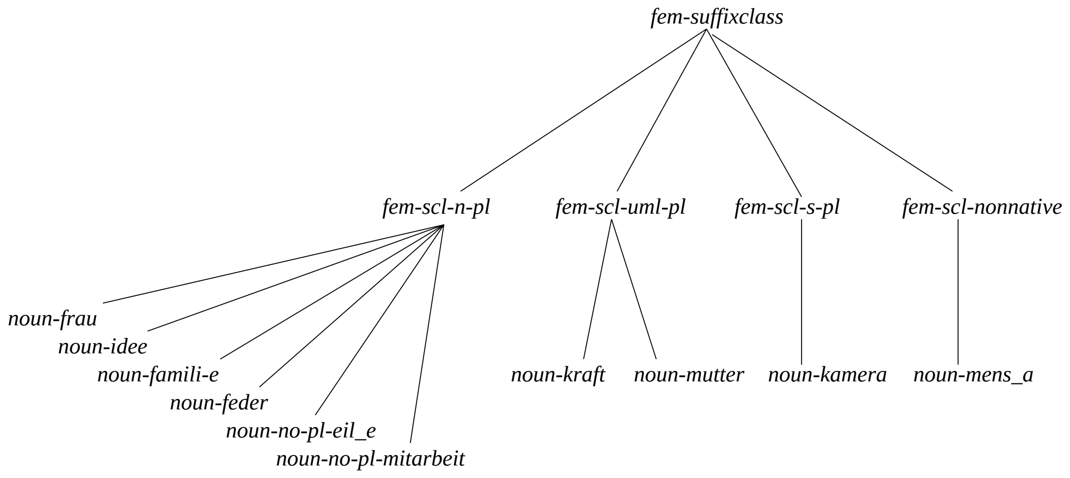


Figure 61: Partition of *fem-suffixclass*

The top-level division of the *n-suffixclass* hierarchy is into feminine and non-feminine suffixclasses. The non-feminine noun classes are basically cross-classified according to genitive and plural formation properties encoded as values of SUFFIXCLASS. Feminine noun classes are differentiated only on account of plural formation properties (for all feminine nouns, the singular formation is identical). The types that go out in bundles under the second-lowest level in Figures 61 and 62, named after prototypical individual lemmata, are supposed to induce the maximal lexical noun lemmata types, from which individual lemmata inherit their attributes. The SUFFIXCLASS feature values of one bundle are differentiated further on possible phonological properties of morphological stems as well as on the (semantic or statistical) non-existence of plural or singular forms for certain lemmata. (Phonological properties will be ultimately inherited from the *base-Complex* and *root* types). Thus, when a two-level morphophonology rule component dealing with umlaut, e-deletion and e-insertion is integrated, most of the class types in one bundle need not be differentiated at all.

Note that though the nominal SUFFIXCLASS feature value types are named after possible inflectional suffixes, at this point we are dealing with morphological properties

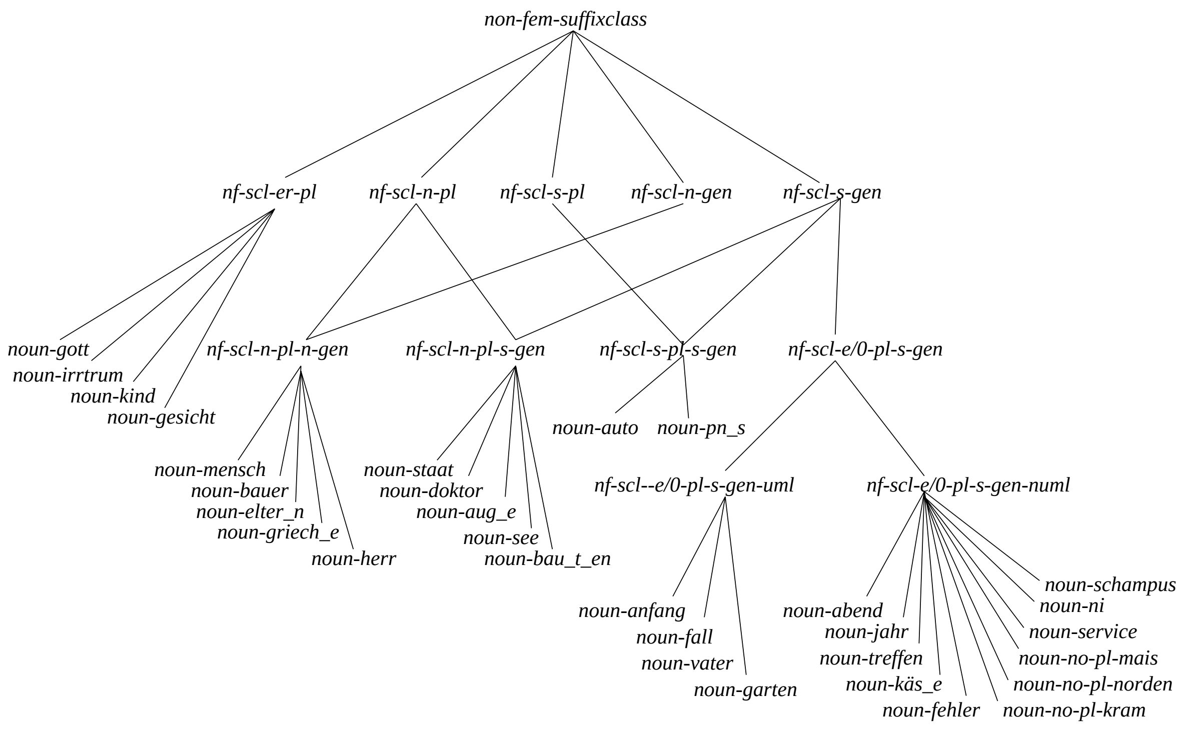


Figure 62: Partition of *non-fem-suffixclass*

of lemmata, which are to be inherited by individual stems. Stems are selected by inflectional suffixes by virtue of morphological subcategorisation. Stems and inflectional suffixes are combined in the construction type *inflected* (see Section 5.5.2).

5.3.2. The word hierarchy

The next main subhierarchy in the hierarchy of morphological types is the *word* hierarchy. It represents the classification of word forms according to the morphosyntactic features, i.e. syntactic HEAD features, such as the famous one proposed for English verbs by (Pollard and Sag, 1987, p.202). The words at the bottom of this hierarchy are the *syntactic atoms* in the sense of Di Sciullo and Williams (1987). Thus, in the partitioning of this hierarchy, it does not play a role whether a word includes a complex or a simple stem, whether it bears an inflectional suffix or whether inflection is indicated by some other phonological process.

The verbal *word* hierarchy for German, for example, is induced through the partitioning of *s-vhead*, which is the value type of the feature SYN|LOC|HEAD, according to the dimensions indicated in Figure 63.

In Section 4.2, we have seen that parallel partitioning of one type according to dif-

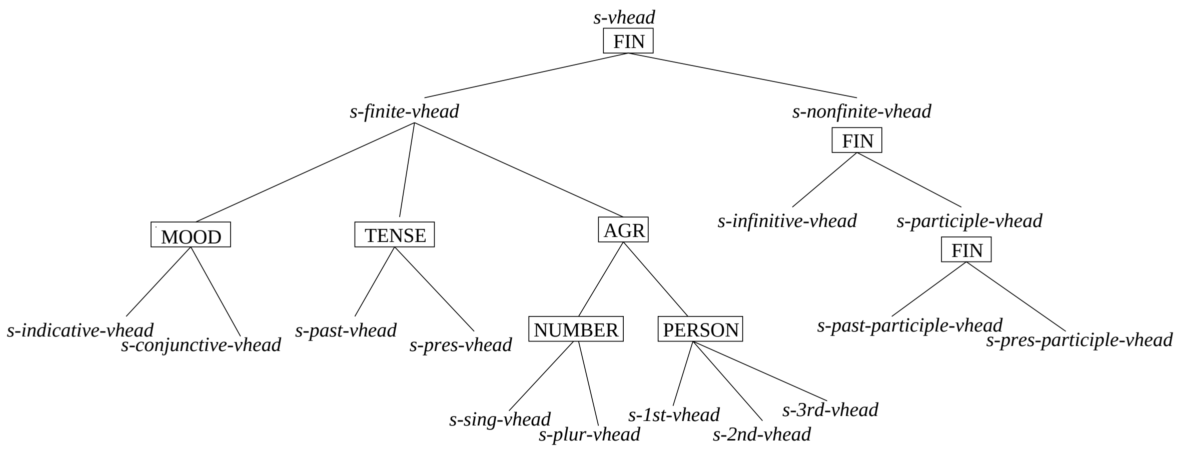


Figure 63: Partition of *s-vhead*, which induces the verbal *word* hierarchy

ferent dimensions leads to a cross-classification of maximal lexical types by multiple inheritance. Thus the dimensions in Figure 63 are those features that occur in the feature declarations of *s-vhead* and its subtype *s-finite-vhead* which are shown in Figure 64.[35]

$$ {}_{\textit{s-vhead}}\begin{bmatrix} \text{MARKING} & {}_{\textit{v-marking}}\begin{bmatrix} \text{TENSE} & \textit{tense} \\ \text{MOOD} & \textit{mood} \\ \text{FIN} & \textit{finiteness} \end{bmatrix} \end{bmatrix} \qquad {}_{\textit{s-finite-vhead}}\begin{bmatrix} \text{MARKING} & {}_{\textit{v-marking}}\begin{bmatrix} \text{TENSE} & \textit{tense} \\ \text{MOOD} & \textit{mood} \\ \text{FIN} & \textit{finite} \\ \text{AGR} & {}_{\textit{agreement}}\begin{bmatrix} \text{PER} & \textit{per} \\ \text{NUM} & \textit{num} \end{bmatrix} \end{bmatrix} \end{bmatrix} $$

Figure 64: Feature declarations for *s-vhead* and *s-finite-vhead*

Multiple inheritance from each partition of this multi-dimensional hierarchy means that each maximal type is a join of types where each conjunct is taken from exactly one partition, and also each partition conversely provides exactly one conjunct.[36] This yields the following set of maximal types under the *s-vhead* hierarchy:

Set of maximal types under the *s-vhead* hierarchy, according to the dimensions FIN, MOOD, TENSE, NUMBER, PERSON:

[35]The top-level branching is a partition along the feature FIN(ITENESS), which is actually a reflection of the appropriateness of the AGR(EEMENT) feature for subtypes of *s-vhead*. That is, lexical objects that are finite require the AGR feature. FIN serves to encode a partition in terms of different values for one feature (FIN), which originally arises on from the presence or absence of another feature (AGR) and is therefore actually redundant. See Lüngen and Sporleder (1999) for a further discussion of type divisions.

[36]This technique is extensively used in Koenig's (1999) approach.

s-indicative-vhead ⊔ *s-present-vhead* ⊔ *s-sing-vhead* ⊔ *s-1st-vhead,*
s-indicative-vhead ⊔ *s-present-vhead* ⊔ *s-sing-vhead* ⊔ *s-2nd-vhead,*
s-indicative-vhead ⊔ *s-present-vhead* ⊔ *s-sing-vhead* ⊔ *s-3rd-vhead,*
s-indicative-vhead ⊔ *s-present-vhead* ⊔ *s-plur-vhead* ⊔ *s-1st-vhead,*
s-indicative-vhead ⊔ *s-present-vhead* ⊔ *s-plur-vhead* ⊔ *s-2nd-vhead,*
s-indicative-vhead ⊔ *s-present-vhead* ⊔ *s-plur-vhead* ⊔ *s-3rd-vhead,*
s-indicative-vhead ⊔ *s-past-vhead* ⊔ *s-sing-vhead* ⊔ *s-1st-vhead,*
s-indicative-vhead ⊔ *s-past-vhead* ⊔ *s-sing-vhead* ⊔ *s-2nd-vhead,*
s-indicative-vhead ⊔ *s-past-vhead* ⊔ *s-sing-vhead* ⊔ *s-3rd-vhead,*
s-indicative-vhead ⊔ *s-past-vhead* ⊔ *s-plur-vhead* ⊔ *s-1st-vhead,*
s-indicative-vhead ⊔ *s-past-vhead* ⊔ *s-plur-vhead* ⊔ *s-2nd-vhead,*
s-indicative-vhead ⊔ *s-past-vhead* ⊔ *s-plur-vhead* ⊔ *s-3rd-vhead,*
s-subjunctive-vhead ⊔ *s-present-vhead* ⊔ *s-sing-vhead* ⊔ *s-1st-vhead,*
s-subjunctive-vhead ⊔ *s-present-vhead* ⊔ *s-sing-vhead* ⊔ *s-2nd-vhead,*
s-subjunctive-vhead ⊔ *s-present-vhead* ⊔ *s-sing-vhead* ⊔ *s-3rd-vhead,*
s-subjunctive-vhead ⊔ *s-present-vhead* ⊔ *s-plur-vhead* ⊔ *s-1st-vhead,*
s-subjunctive-vhead ⊔ *s-present-vhead* ⊔ *s-plur-vhead* ⊔ *s-2nd-vhead,*
s-subjunctive-vhead ⊔ *s-present-vhead* ⊔ *s-plur-vhead* ⊔ *s-3rd-vhead,*
s-subjunctive-vhead ⊔ *s-past-vhead* ⊔ *s-sing-vhead* ⊔ *s-1st-vhead,*
s-subjunctive-vhead ⊔ *s-past-vhead* ⊔ *s-sing-vhead* ⊔ *s-2nd-vhead,*
s-subjunctive-vhead ⊔ *s-past-vhead* ⊔ *s-sing-vhead* ⊔ *s-3rd-vhead,*
s-subjunctive-vhead ⊔ *s-past-vhead* ⊔ *s-plur-vhead* ⊔ *s-1st-vhead,*
s-subjunctive-vhead ⊔ *s-past-vhead* ⊔ *s-plur-vhead* ⊔ *s-2nd-vhead,*
s-subjunctive-vhead ⊔ *s-past-vhead* ⊔ *s-plur-vhead* ⊔ *s-3rd-vhead,*
s-infinitive-vhead,
s-pres-participle-vhead,
s-past-participle-vhead

The set of the PHON and ORTH values of the word forms which belong to these types and share a common stem make up the inflectional *paradigm*, in this case a verbal paradigm. We will discuss the mappings from lemmata to paradigms (i.e. paradigm functions) in Section 5.5.3, after having discussed the features appropriate for the type *inflected*.

5.3.3. The morpheme hierarchy

The *morpheme* is the only subtype of *sign* that is definitely *lexicalised* in that the maximal types subsumed under it have to be 'listed'. Instances of other types, such as *lexComplex*, are not necessarily lexicalised, i.e. the type *lexComplex* subsumes lexicalised as well as non-lexicalised words. The type *morpheme* is the type complementary to *complex* in the partitioning of the type *sign* (Figure 59).

Morpheme is partitioned twice, first, along the boolean-valued dimension MORPH|HEAD|DER|NAT(IVE). Second, along the appropriateness of the feature

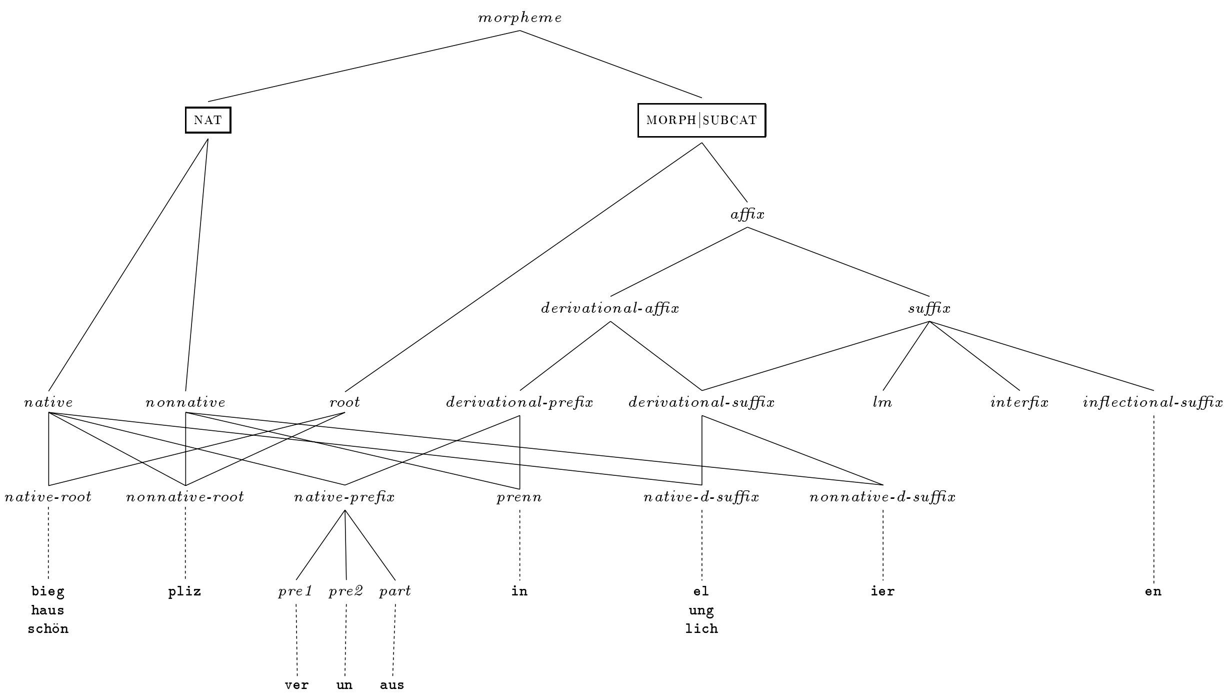


Figure 65: The *morpheme* hierarchy for German

MORPH|SUBCAT into *root* (which have no morphological subcategorisation properties) and *affix. Affix* is partitioned into *d(erivational)-prefix* and *suffix,* which in turn is partitioned into *d-suffix, inflectional-suffix*, plus the two minor types *interfix* and *linking-morpheme (lm).* The partitioning dimension for the first three is a conjunction of the appropriateness of MORPH|SUBCAT and the MPHON|LBOUND and MPHON|RBOUND features. Since this complex dimension leads to three disjunctive subtypes only, there must must be a gap in the set of possible resulting subtypes, and in fact, German is lacking morphemes of a potential type that might be called *inflectional-prefix. D-prefix* is partitioned into *part* (verbal particles),[37] *pre1* (class I prefixes), *pre2* (class II prefixes), and *prenn* (nonnative prefixes).

5.3.4. The lexComplex Hierarchy

The *lexComplex* hierarchy classifies morphological bases according to their constructional make-up, i.e. according the type of the value of the feature DTRS. (The type *lexComplex* is distinguished from the type *morpheme* by the presence of the DTRS attribute, cf. the hierarchy in Figure 59.) *LexComplex* subsumes complex lexical signs. The

[37]In the Verbmobil lexicon context, we considered verbal particles prefixes, but this is of course highly controversial. Cf. Lüdeling (1999) and Müller (2000) for recent comprehensive studies of verbal particles.

top-level distinction is between *baseComplex* and *inflected*. The latter has *marker-struc* as a value of the feature DTRS.

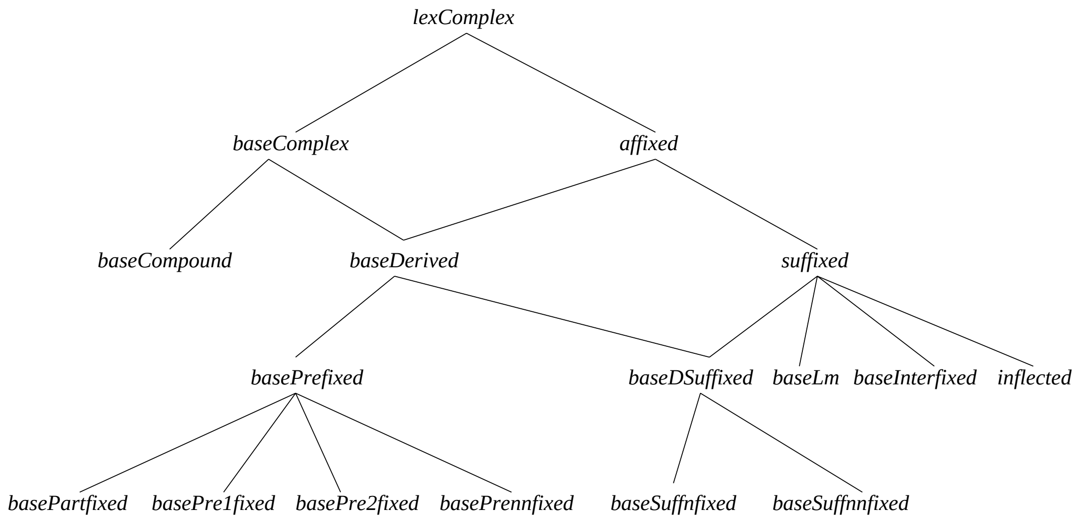


Figure 66: Hierarchy of complex lexical signs

Thus, the type *inflected* is responsible for the combination of stems and inflectional suffixes, where the IMP ensures the marking of syntactic HEAD features on the resulting word, and MSP licenses and constrains only the correct combinations by specifying values of the morphological HEAD features STEMCLASS, SUFFIXCLASS, and VROOT on the stem.

The type *baseComplex* subsumes all types that describe concatenative word formation constructions, i.e. combinations of one base and one affix (*baseDerived*) or combinations of two bases (*baseCompound*). A *base* is thus a type of sign on which a morphological operation can be performed, or, in HPSG terms, that can function as a base daughter in *baseComplex* or the stem daughter in *inflected* constructions. A base may be complex (*baseComplex*), or simplex (*root*). A *base* construction is thus defined recursively, and the recursion is grounded by the possibility for bases to be simplex. How do we distinguish bases from stems then? A stem is supposed to be a base which is at the same time subsumed by the type *lemma*, i.e. a base that additionally bears inflectional class specifications, i.e. which can be selected by inflectional suffixes through morphological subcategorisation. We want to distinguish between stems and bases as there seem to be bases that are not stems, but still play a role in word formation. For example, words like `re+par+ier#+en, Re+par+at+ur, re+par+abel` contain the complex base `re+par-`, which in turn contains the simplex base (i.e. root) `par-` (which also occurs in the word `Kom+par+at+ist+ik`). The bases `par-` and `repar-` do not bear STEMCLASS and SUFFIXCLASS specifications (they do not even seem to be specified for a specific MAJOR syntactic category) and thus cannot be combined with inflectional suffixes. Proponents of realisational morphology might argue that morphological relations exist only between the stems `reparier-` and `Reparatur` and that things like `par-`, `-at-`, `-ier` are not lexical objects. But creative word formation does operate on such objects, if we consider for example the language of advertising or science, where new words seem to be formed from such elements without involving lexical bases that correspond to true lexemes.

The type *baseDerived* is further subdivided into *basePrefixed* and *baseDSuffixed.* Their subtypes are derivations whose DTRS|$AFFIX-DTR value is *part, pre1, pre2, prenn, sufn*, and *sufnn*, respectively. The *baseDerived* hierarchy is thus induced by the *d-affix* hierarchy presented above.

$$
{}_{\textit{basePrefixed}}\left[\begin{array}{ll}
\text{SURF} & \begin{bmatrix}\text{PHON} & \text{2LEVEL}(\boxed{5},\boxed{1})\\ \text{ORTH} & \text{2LEVEL}(\boxed{6},\boxed{2})\end{bmatrix}\\
\text{MORPH} & \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{RBOUND} & \boxed{7}\\ \text{LBOUND} & \boxed{4}\end{bmatrix}\end{bmatrix}\\
\text{DTRS} & {}_{\textit{headed-struc}}\left[\begin{array}{ll}
\text{HEAD-DTR} & \begin{bmatrix}\text{SURF} & \begin{bmatrix}\text{PHON} & \boxed{1}\\ \text{ORTH} & \boxed{2}\end{bmatrix}\\ \text{MORPH} & \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{RBOUND} & \boxed{3}\\ \text{LBOUND} & \boxed{4}\end{bmatrix}\end{bmatrix}\end{bmatrix}\\
\text{NONHEAD-DTR} & \begin{bmatrix}\text{SURF} & \begin{bmatrix}\text{PHON} & \boxed{5}\\ \text{ORTH} & \boxed{6}\end{bmatrix}\\ \text{MORPH} & \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{RBOUND} & \boxed{7}\\ \text{LBOUND} & \boxed{8}\end{bmatrix}\end{bmatrix}\end{bmatrix}
\end{array}\right]
\end{array}\right]
$$

Figure 67: Continuation Schema for *basePrefixed*

The concatenation of morphs has to be taken care of within *lexComplex* and its subtypes, too. For this, we provide three schemata in Figures 67, 68, and 69. The schemata make sure that the RBOUND and LBOUND features of a complex base are shared with those of the constituent that is to appear to the right and to the left, respectively. Thus, in each Continuation Schema, an interface to a concatenating function called 2LEVEL is defined. Only such a schema ensures that a prefix is actually pre-fixed, and a suffix suf-fixed. And from the schema in Figure 68, it follows that heads are right-headed in German derivation and compounding.

The Continuation Schemata remain tentative at this stage, because they do not say anything about how the MPHON information is used in the 2LEVEL function, and because they lack a description of the way the actual RBOUND and LBOUND continuation class requirements are satisfied (in the schema, there is only an account of the way the information not to be satisfied in the schema itself is shared between mother and

$${}_{\textit{baseDSuffixed} \sqcap \textit{baseCompound}}\left[\begin{array}{ll}
\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \textsc{2level}\left(\boxed{1},\boxed{5}\right)\\ \text{ORTH} & \textsc{2level}\left(\boxed{2},\boxed{6}\right)\end{array}\right]\\
\text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{4}\\ \text{LBOUND} & \boxed{7}\end{array}\right]\end{array}\right]\\
\text{DTRS} & {}_{\textit{headed-struc}}\left[\begin{array}{ll}
\text{HEAD-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{1}\\ \text{ORTH} & \boxed{2}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{3}\\ \text{LBOUND} & \boxed{4}\end{array}\right]\end{array}\right]\end{array}\right]\\
\text{NONHEAD-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{5}\\ \text{ORTH} & \boxed{6}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{7}\\ \text{LBOUND} & \boxed{8}\end{array}\right]\end{array}\right]\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 68: Continuation Schema for *baseDSuffixed* ⊓ *baseCompound*

$${}_{\textit{inflected}}\left[\begin{array}{ll}
\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \textsc{2level}\left(\boxed{1},\boxed{5}\right)\\ \text{ORTH} & \textsc{2level}\left(\boxed{2},\boxed{6}\right)\end{array}\right]\\
\text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{4}\\ \text{LBOUND} & \boxed{7}\end{array}\right]\end{array}\right]\\
\text{DTRS} & {}_{\textit{marker-struc}}\left[\begin{array}{ll}
\text{MARKER-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{1}\\ \text{ORTH} & \boxed{2}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{3}\\ \text{LBOUND} & \boxed{4}\end{array}\right]\end{array}\right]\end{array}\right]\\
\text{NONMARKER-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{5}\\ \text{ORTH} & \boxed{6}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{7}\\ \text{LBOUND} & \boxed{8}\end{array}\right]\end{array}\right]\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 69: Continuation Schema for *inflected*

daughters).

5.4. Root allomorphy

The maximal lexical types of the morpheme hierarchy are *morphs* in the sense of morphologically and lexically conditioned allomorphs of morphemes (as opposed to phonologically conditioned allomorphs). This means that differently ablauted roots like *sing-morph*, *sang-morph*, and *sung-morph* are subtypes of *sing-morpheme*, since they are its different morphologically conditioned allomorphs. *Sing-morpheme* represents a class of morphs which is higher up in the type hierarchy. Every sort-resolved feature structure of type *morpheme* is thus a *morph*, signs that are just morphemes but not morphs do not exist.

Forms such as `bleib` as occurring in the word forms `bleib#+st` and `bleib#+e` are classified as one single morph with the underlying phonological representation `blaIb`. The distribution and combination of these morphs in actual morphological constructions (represented in the *lexComplex* hierarchy, cf. Sections 5.5.1 and 5.5.2) is driven by (morphological) features. In Figure 70, the morpheme-morph relation and the type-dividing dimensions for the morpheme *sing-morpheme* are shown. The values of the SURF(ACE) attributes are underspecified in a morpheme. Phonological underspecification techniques for typed feature structures are described in Bird and Klein (1994) and Koenig (1999).

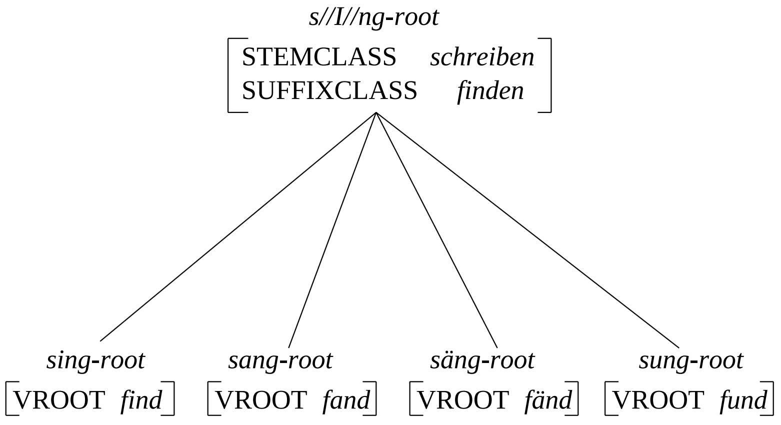


Figure 70: Morpheme-morph relation as type subsumption

Incidentally, the inventory of German morphs in the sense of morphologically and lexically conditioned allomorphs is very close to the minimal units of meaning represented by orthography (thus we would like to call them *orthographic morphs*). In general, German orthography is close to the morphemic level, i.e. cases of orthographic allomorphy are fairly restricted. The main cases where orthography represents allomorphy in German are the said morphological alternations of lexical roots involving umlaut and ablaut, and the different inflectional suffixes representing the same kind of morphosyntactic specifications, i.e. those alternations that are rather difficult to represent in a classical Two-level morphology, since they are controlled by interactions of morphological (i.e. not phonological) properties such as inflectional class and parametrised umlautability, which would have to be encoded as diacritic markers in the lexical form (i.e. the underlying phonological representation) of the morphemes in question. The alternative is to use feature specifications and the operation of unification not only in the description of morphotactics but also of morphophonology.

Other cases of morphologically conditioned allomorphy that is also reflected in orthography involve schwa-elision in stems (`dunkel` vs. `dunkl#+es`), in inflectional suffixes (`Frau#+en` vs. `Feder#+n`), and schwa-epenthesis in inflectional suffixes (`sag#+te` vs. `wart#+ete`).

There are three cases of purely phonologically conditioned orthographic allomorphy. The first is the alternative ss/ß-spelling after short vowels (`ss` before vowels, `ß` elsewhere), where no morphological conditioning is involved.[38] The second is doubling of stem-final single consonants in orthography after short vowels and before unstressed vowels (`Lehrer+in` vs. `Lehrer+inn#+en`). The third is the merger of a stem-final consonant with a suffix beginning with the same consonant e.g. in verbal inflection: `sag#+st` vs. `fax#+t` (the value of SYN|LOC|HEAD of either is typed to $\textit{indicative} \sqcup \textit{present} \sqcup \textit{sing} \sqcup \textit{2nd}$).

Other purely phonologically conditioned allomorphy, such as final devoicing, is not reflected in spelling, i.e. orthography generally represents the archiphonemic, underlying level, over which the *morpheme* hierarchy is constructed. An overview of the morphologically conditioned allomorphies in German is shown in Table 3.

The morpheme-morph/allomorph relation is thus interpreted as type subsumption in our HPSG morphology. The features named in Table 3 are the dimensions that distinguish between the different morph types that are allomorphs of one morpheme.

Note that the definition of the types *lemma* and *stem* is analogous: a *lemma* is a just class of stems, and every sort-resolved *lemma* automatically corresponds to a *stem*, because a stem is the realisation of a lemma. A stem is either identical with a root morph, or has a root morph as its head. A lemma is underspecified in that it corresponds to a root morpheme or has a root morpheme as its head. Note also that we want to avoid to call the word forms that make up a paradigm the *realisations* of a lemma. They are not realisations in the same sense (the sense of type subsumption), because being a member of the paradigm of one lemma involves constituency as well (Section 5.5.3).

[38]In the new German orthography introduced in 1999 (not employed in the Verbmobil transcriptions), the s/ß-based allomorphy has been eliminated (`ss` is now always used after short vowels).

Allomorphy	Partitioning of types in morpheme hierarchy by features	Example of morph types	as in the word forms
Ablaut in verbal roots	STEMCLASS, VROOT	{*sprech∼sprach∼sproch*}	sprechen, sprach, gesprochen
i/e alternation in verbal roots	STEMCLASS, VROOT	{*sprich∼sprech*}	sprichst, sprechen
Umlaut in nominal inflection	UMLD, UMLB, SUFFIXCLASS	{*haus∼häus*}	Haus, Häuser
Umlaut in derivation	UML, UMLB, UMLD	{*wort∼wört*}	Wort, Wörtchen
Stress shift in nonnative nouns with suffix *-or*	SUFFIXCLASS	{*/-o:r/∼/-'o:r/*}	Doktor, Doktoren
Schwa-epenthesis in inflection of weak verbs	SUFFIXCLASS	{*-t∼-et*}	sagt, wartet
Schwa-elision in nominal and verbal inflectional suffixes	SUFFIXCLASS	{*-en∼-n*}	Frauen, Federn; fragen, weigern
Schwa-elision in adjectival and verbal stems	MAJOR	{*dunkel∼dunkl*} {*gammel∼gamml*}	dunkel, dunkles; gammeln, gammle

Table 3: Morphologically conditioned allomorphy in German

5.5. More specific morphological construction types

5.5.1. Derivational affix classification

Before embarking on the examination of types of morphological constructions in German, we want to examine more closely the affix hierarchy for German and the dimensions that distinguish between different types of affixes. As we have seen above, the dimension that distinguishes affixes from other types of morphemes is the appropriateness of the MORPH|SUBCAT attribute, which has a (morphological) base as its value. Prefixes are distinguished from suffixes through their RBOUND and LBOUND values, cf. Table 4.

TYPE	CONSTRAINTS	ISA
prefix	$\left[\text{MORPH}\vert\text{MPHON}\ \left[\begin{array}{ll}\text{SEP} & \textit{boolean}\\ \text{RBOUND} & \left[\begin{array}{ll}\text{BOUND} & \textit{toBind}\\ \text{BND-ARG} & \textit{base-initial}\end{array}\right]\\ \text{LBOUND} & \left[\begin{array}{ll}\text{BOUND} & \textit{opt}\\ \text{BND-ARG} & \textit{stem-final} \sqcap \textit{prefix}\end{array}\right]\end{array}\right]\right]$	*d-affix*
suffix	$\left[\text{MORPH}\vert\text{MPHON}\ \left[\begin{array}{ll}\text{RBOUND} & \left[\begin{array}{ll}\text{BOUND} & \textit{opt}\\ \text{BND-ARG} & \textit{suffix} \sqcap \textit{infl} \sqcap \textit{stem-initial}\end{array}\right]\\ \text{LBOUND} & \left[\begin{array}{ll}\text{BOUND} & \textit{toBind}\\ \text{BND-ARG} & \textit{base-final} \sqcap \textit{prefix}\end{array}\right]\end{array}\right]\right]$	*affix*

Table 4: Feature declarations for *prefix* and *suffix*

In Section 5.1, the two affix-distinguishing dimensions MORPH|MPHON|STR and MORPH|HEAD|DER|NAT were introduced, but to distinguish all different types of affixes shown in Figure 65, we additionally employ the dimensions MORPH|MPHON|ADJ(ACENCY) and MORPH|MPHON|SEP(ARABILITY), after Fleischer and Barz (1992), and Steinbrecher (1995). ADJ is *boolean*-valued and describes the property of being affixable only directly adjacent to a simplex base ([ADJ +]) as opposed to being affixable to any kind of base (complex or simplex, [ADJ −]). SEP is needed to distinguish separable native prefixes (SEP +) from inseparable native prefixes (SEP −).

We can thus characterise the derivational affixes with the help of four boolean features as shown in Table 5.

	NAT	STR	SEP	ADJ	Examples
part	+	+	+	−	*an-, auf,- über-, zu-*
pre2	+	+	−	−	*miß-, un-, erz-, haupt-*
pre1	+	−	−	+	*be-, er-, ent- ver-, zer-*
prenn	−	+	−	+	*in-, hyper-, kon-, re-, sub-*
sufn	+	−	−	−	*-er, -ig, -heit, -keit, -lich, -ung*
sufnn	−	+	−	−	*-ier, -ion, -ität, -iv*

Table 5: Affix classification after Steinbrecher (1995)

(Steinbrecher, 1995, p.11) notes that the feature ADJ (a property considered distinctive by Hoeppner, 1980) is actually redundant in the above scheme, as its value can be predicted by the values of the remaining features. In Table 6, the specific subcategorisation requirements of the suffix -*keit* are shown.

TYPE	CONSTRAINTS	ISA
native-n-suffix	$\left[\begin{array}{ll} \text{SURF} & \left[\begin{array}{ll} \text{PHON} & \texttt{kaIt} \\ \text{ORTH} & \texttt{keit} \end{array}\right] \\ \text{MORPH} & \left[\begin{array}{ll} \text{SUBCAT} & \left[\begin{array}{l} \text{MORPH} \vert \text{DER} \vert \text{NAT:} + \\ \text{SYN} \vert \text{LOC} \vert \text{HEAD:} \textit{ adj} \end{array}\right] \end{array}\right] \end{array}\right]$	*native-n-suffix* ⊔ *noun-frau-lemma* ⊔ *noun-nom-fem-word*

Table 6: Feature declarations for the suffix *keit*

The remaining feature specifications of *keit_sufn* are inherited from the relevant maximal types under the three hierarchies *morpheme*, *lemma*, and *word*, namely *native-suffix*, *noun-frau-lemma* (providing the morphological HEAD features), and *noun-nom-fem-word* (responsible for the syntactic HEAD feature specifications).

Our hierarchy of derivational affixes induces the hierarchy of morphological derivational bases, whose most general supertype is *baseDerived* (cf. Figure 59). In other words, the *baseDerived* hierarchy is subdivided along the dimension of the DTRS attribute, as either the value of DTRS|HEAD-DTR is a subtype of *suffix*, or the value of DTRS|NONHEAD-DTR is a subtype of *prefix*. The different subtypes of *prefix*, for example, yield the subtypes of *basePrefixed* as shown in Table 7. Every kind of derivational affixation must be licensed by one Continuation Schema (Section 5.3.4), the MHFP, (Section 5.2), the MSP (Section 5.2.4), and by the individual lexical entries for affixes.

As we have seen above, the *baseComplex* hierarchy serves to organise the lexicon along morphological construction types, therefore it subsumes the actual, lexicalised bases, and serves as well as the grammar (morphotactics) which describes non-lexicalised, potential words. The following sections are meant to explain in more detail how binary phrase structure word formation rules with annotated equations are expressed as HPSG feature structures denoting lexical construction types. The focus is firstly on more general types of bases (*basePrefixed, baseDsuffixed, baseCompound*) and the type *inflected*. Later sections are devoted to the more specific morphotactic problems such as umlauting in derivation, and linking morphemes in compounding. Some additional morphological features and thus some new subtypes in the *baseComplex* hierarchy are needed to treat these phenomena.

$$
{}_{\textit{basePrefixed}}\left[\begin{array}{ll}
\text{MORPH} & {}_{\textit{base-morphology}}\left[\begin{array}{ll} \text{HEAD} & \boxed{2} \\ \text{MPHON} & \left[\begin{array}{ll} \text{LBOUND} & \boxed{4} \\ \text{RBOUND} & \boxed{5} \end{array}\right] \end{array}\right] \\
\text{SYN} & \text{LOC : HEAD : } \boxed{3} \\
\text{DTRS} & {}_{\textit{headed-struc}}\left[\begin{array}{ll}
\text{NON-HEAD-DTR} & {}_{\textit{prefix}}\left[\begin{array}{ll} \text{MORPH} & {}_{\textit{affix-morphology}}\left[\begin{array}{ll} \text{MPHON} & \left[\text{LBOUND } \boxed{4}\right] \\ \text{SUBCAT} & \boxed{1} \end{array}\right] \end{array}\right] \\
\text{HEAD-DTR} & \boxed{1}\ {}_{\textit{base}}\left[\begin{array}{ll} \text{MORPH} & {}_{\textit{base-morphology}}\left[\begin{array}{ll} \text{MPHON} & \left[\text{RBOUND } \boxed{5}\right] \\ \text{HEAD} & \boxed{2} \end{array}\right] \\ \text{SYN} & \text{LOC : HEAD : } \boxed{3} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Figure 71: The construction type *basePrefixed*

For an example of a lexical construction type cf. the specifications for *basePrefixed* in Figure 71. The tags in the AVM notation are inherited from the respective morphological principles:

1. The prefix is subcategorised for the base, i.e. it constrains the set of bases that can appear in a *basePrefixed* structure (expressed through the tag $\boxed{1}$, inherited from the MSP). These constraints can be phonological (the prefix *im_prenn*, for example, attaches only to roots beginning with a labial consonant), morphological (e.g. constraints on nativeness), syntactic (e.g. constraints on the major syntactic category) or semantic (e.g. constraints on the arity of the valence of the base).
2. The HEAD-DTR shares its morphological HEAD features with the *basePrefixed* structure as a whole (expressed through the tag $\boxed{2}$, inherited from the MHFP).
3. The HEAD-DTR shares its syntactic HEAD features with the *basePrefixed* structure as a whole (expressed through the tag $\boxed{3}$, also inherited from the MHFP).
4. The value of MORPH|MPHON|LBOUND is that of the MORPH|MPHON|LBOUND feature of the base (tag $\boxed{4}$, inherited from a Continuation Schema).
5. The value of MORPH|MPHON|RBOUND is that of the MORPH|MPHON|RBOUND feature of the prefix (tag $\boxed{5}$, also inherited from the Continuation Schema).

TYPE	CONSTRAINTS	ISA
basePre1fixed	$\left[\text{NON-HEAD-DTR} : \textit{prefix1}\right]$	*basePrefixed*
basePre2fixed	$\left[\text{NON-HEAD-DTR} : \textit{prefix2}\right]$	*basePrefixed*
basePartfixed	$\left[\text{NON-HEAD-DTR} : \textit{part}\right]$	*basePrefixed*
basePrennfixed	$\left[\text{NON-HEAD-DTR} : \textit{prenn}\right]$	*basePrefixed*

Table 7: Feature declarations for subtypes of *basePrefixed*

Compare this with the AVM for type *baseDsuffixed* shown in Figure 72 where

1. the suffix is subcategorised for a *base*, i.e. it constrains the set of bases that can appear in the *baseDsuffixed* structure (expressed through the tag $\boxed{1}$, inherited from the MSP).
2. the HEAD-DTR shares its morphological HEAD features with the *baseDsuffixed* structure as a whole (expressed through the tag $\boxed{2}$, inherited from the MHFP).
3. The HEAD-DTR shares its syntactic HEAD features with the *baseDsuffixed* structure as a whole (expressed through the tag $\boxed{3}$, also inherited from the MHFP).
4. the Value of MORPH|MPHON|LBOUND is token-identical with that of the MORPH|MPHON|LBOUND feature of the base (tag $\boxed{5}$, inherited from a Continuation Schema).
5. the Value of MORPH|MPHON|RBOUND is token-identical with that of the MORPH|MPHON|RBOUND feature of the suffix (tag $\boxed{4}$, also inherited from a Continuation Schema).

$$
{}_{\textit{baseDsuffixed}}\begin{bmatrix}
\text{MORPH} & {}_{\textit{base-morphology}}\begin{bmatrix} \text{HEAD} & \boxed{2} \\ \text{MPHON} & \begin{bmatrix} \text{LBOUND} & \boxed{4} \\ \text{RBOUND} & \boxed{5}\end{bmatrix}\end{bmatrix} \\
\text{SYN} & \text{LOC : HEAD : }\boxed{3} \\
\text{DTRS} & {}_{\textit{headed-struc}}\begin{bmatrix} \text{NONHEAD-DTR} & \boxed{1}\ {}_{\textit{base}}\begin{bmatrix}\text{MORPH} & {}_{\textit{base-morphology}}\begin{bmatrix}\text{MPHON} & \left[\text{LBOUND}\ \boxed{4}\right]\end{bmatrix}\end{bmatrix} \\ \text{HEAD-DTR} & {}_{\textit{suffix}}\begin{bmatrix} \text{MORPH} & {}_{\textit{affix-morphology}}\begin{bmatrix}\text{MPHON} & \left[\text{RBOUND}\ \boxed{5}\right] \\ \text{HEAD} & \boxed{2} \\ \text{SUBCAT} & \boxed{1}\end{bmatrix} \\ \text{SYN} & \text{LOC : HEAD : }\boxed{3}\end{bmatrix}\end{bmatrix}
\end{bmatrix}
$$

Figure 72: The construction type *baseDSuffixed*

Thus, apart from the linear precedence distinctions formulated under MORPH|MPHON, the difference between prefixes and suffixes is that prefixes cannot syntactically and morphologically head a morphological base, but suffixes can

$$ {}_{baseDerived}\begin{bmatrix} \text{DTRS} & {}_{headed\text{-}struc}\begin{bmatrix} \text{\$BASE-DTR} & \boxed{1}\ {}_{base}[\,] \\ \text{\$AFFIX-DTR}_{affix} & \begin{bmatrix}\text{MORPH} & {}_{affix\text{-}morphology}\begin{bmatrix}\text{SUBCAT} & \boxed{1}\end{bmatrix}\end{bmatrix}\end{bmatrix}\end{bmatrix} $$

Figure 73: AVM for type *baseDerived*

(this is a characteristic that they share with morphological roots). On the other hand, prefixes share with suffixes the property of being subcategorised for (subtypes of) morphological bases. Note that the NONHEAD-DTR is in both cases not required to be specified for HEAD features. This accounts firstly for prefixes, which are never appropriate for HEAD features (cf. Section 5.2.2), and secondly, for lexical roots, since some lexical roots are not subtypes under the *lemma* hierarchy, as was pointed out in Section 5.3.

In Figure 66 it is shown that there is a supertype *baseDerived* subsuming both *basePrefixed* and *baseDsuffixed.* The associated feature structure, whose AVM is given in Figure 73, states the fact that in a derivation, the affix (AFFIX-DTR) is subcategorised for the base (BASE-DTR, cf. tag $\boxed{1}$).

The following holds for the instantiations of $BASE-DTR and $AFFIX-DTR:

1. The $AFFIX-DTR is the HEAD-DTR of a *baseDerived* if and only if its MORPH|MPHON value is shared with that of the $AFFIX-DTR and its MORPH|MPHON|LBOUND value is shared with that of the $BASE-DTR, which is then the NONHEAD-DTR (this implies that the $AFFIX-DTR is a *suffix*).
2. The $BASE-DTR is the HEAD-DTR of a *baseDerived* if and only if its MORPH|MPHON|RBOUND value is shared with that of the $BASE-DTR and its MORPH|MPHON|LBOUND value is shared with that of the $AFFIX-DTR, which is then the NONHEAD-DTR (this implies that the $AFFIX-DTR is a *prefix*).

Let us now turn to compounds. These are distinguished from derivations by the fact that two bases are involved. The type of the first element is *baseLm*, i.e. a base that may occur as the first constituent of a compound (depending of the obligatoriness of a **L**inking **m**orpheme (*Fugenelement*)). As with derivational construction types, we assume that the right element must always bear HEAD feature specifications, both syntactic and morphological, which are shared with the compound as a whole. Neo-classical compounds (cf. Bauer, 1983, p.213ff) such as `Pathologie` might pose a problem with respect to this claim. In the Verbmobil corpus, 10 neo-classical compound constructions (partly occurring as sub-constituents of even more complex words) could be found, whose right-hand elements were always one of the nonnative roots `-log`, `-skop`, or `-fon/phon`, namely in the word forms `"oko#log+isch, Horo#skop, Tele#fon,`

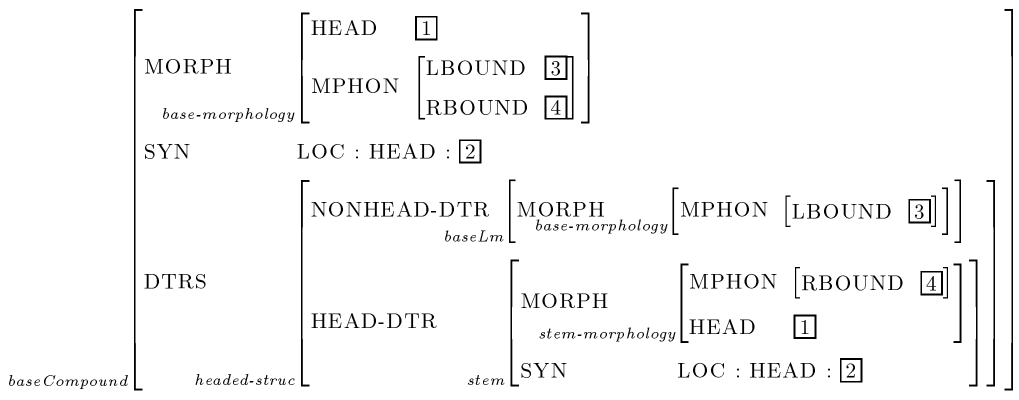


Figure 74: AVM for type *baseCompound*

`Tele#fon+at, Tele#fon+ats, Tele#fon#nummer, Xylo#phon, bio#log+isch#+er, chrono#log+isch#+en, tele#fon+isch`. These roots are definitely specified for HEAD features, as they may be (optionally, or in the case of `-log-`, obligatorily) combined with an inflectional suffix to the right.[39] Until further examinations of corpora with neo-classical compounds, the HEAD-DTR of a compound construction will be typed to *stem*.

The morphological structure sharing that holds for the type *baseCompound*, i.e. for compound stems, can be explained as follows (cf. Figure 74):

1. There are no morphological or syntactic subcategorisation requirements to be satisfied in a *baseCompound* structure. Of course, the NONHEAD-DTR *may* be an element from the SYN|SYN|LOC|SUBCAT-list of the HEAD-DTR, and is very often so, e.g. in the most prominent readings of the lemmata *Doktorandentreffen* and *Terminvereinbarung*. These compounds form a special, syntactically and semantically motivated subtype, but not a morphologically motivated subtype of *baseCompound*.
2. The HEAD-DTR shares its morphological HEAD features with the *baseCompound* structure as a whole (expressed through the tag $\boxed{1}$, inherited from the MHFP).
3. The HEAD-DTR shares its syntactic HEAD features with the *baseCompound* structure as a whole (expressed through the tag $\boxed{2}$, also inherited from the MHFP).
4. The Value of MORPH|MPHON|LBOUND is token-identical with that of the MORPH|MPHON|LBOUND feature of the NONHEAD-DTR (tag $\boxed{5}$, inherited from a Continuation Schema).

[39]Any compound ending in `-log-` is specified as MORPH|HEAD|FLEX|SUFFIXCLASS *griech-e*, so that even the word `Chronologe` is correctly described as a potential word of German.

$$
{}_{\textit{baseComplex}}\begin{bmatrix}
\text{MORPH}_{\textit{base-morphology}} & \begin{bmatrix} \text{HEAD} & \boxed{1} \\ \text{MPHON} & \begin{bmatrix} \text{LBOUND} & \boxed{3} \\ \text{RBOUND} & \boxed{4} \end{bmatrix} \end{bmatrix} \\
\text{SYN} & \text{LOC : HEAD : } \boxed{2} \\
\text{DTRS}_{\textit{headed-struc}} & \begin{bmatrix} \text{NONHEAD-DTR} & {}_{\textit{base}}\begin{bmatrix} \text{MORPH}_{\textit{base-morphology}} \begin{bmatrix} \text{MPHON} \begin{bmatrix} \text{LBOUND} & \boxed{3} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\
\text{HEAD-DTR} & {}_{\textit{base} \sqcap \textit{suffix}}\begin{bmatrix} \text{MORPH}_{\textit{morphology}} & \begin{bmatrix} \text{MPHON} \begin{bmatrix} \text{RBOUND} & \boxed{4} \end{bmatrix} \\ \text{HEAD} \; \boxed{1} \end{bmatrix} \\ \text{SYN} & \text{LOC : HEAD : } \boxed{2} \end{bmatrix} \end{bmatrix}
\end{bmatrix}
$$

Figure 75: AVM for type *baseComplex*

5. The Value of MORPH|MPHON|RBOUND is token-identical with that of the MORPH|MPHON|RBOUND feature of the HEAD-DTR (tag $\boxed{4}$, inherited from a Continuation Schema).

Finally, there is a supertype for all kinds of word formation constructions, subsuming both *baseDerived*, and *baseCompound*. The properties that are shared by these constructions are a.) that they are appropriate for the DTRS attribute, b.) the morphological Head Feature Principle in connection with the Right-Hand Head Rule, which distinguishes them from Stem-Infl constructions (cf. Section 5.2.3). This can be seen in the feature structure given as the AVM in Figure 75.

5.5.2. Inflectional suffixation

As discussed elaborately in Sections 5.2.3 and 5.3, constructions that consist of a stem and an inflectional suffix are licensed by the schema associated with the type *inflected* and are constrained by the Inflectional Marking Principle (IMP), which describes the contributions of the stem and of the inflectional suffix to the syntactic HEAD feature information of a full word form. Stems are selected by the suffixes through morphological subcategorisation of the stems' inflectional class features. Another property that distinguishes inflection from derivation is that a Stem-Infl construction usually shares its syntactic subcategorisation information with that of its stem[40]. The structure sharing described in Figure 76 has the following effects:

1. The inflectional suffix (MARKER-DTR) is morphologically subcategorised for the

[40]Not in the passive and past participle formations; since our theory does not treat syntactic subcategorisation, we do not provide a principle for this.

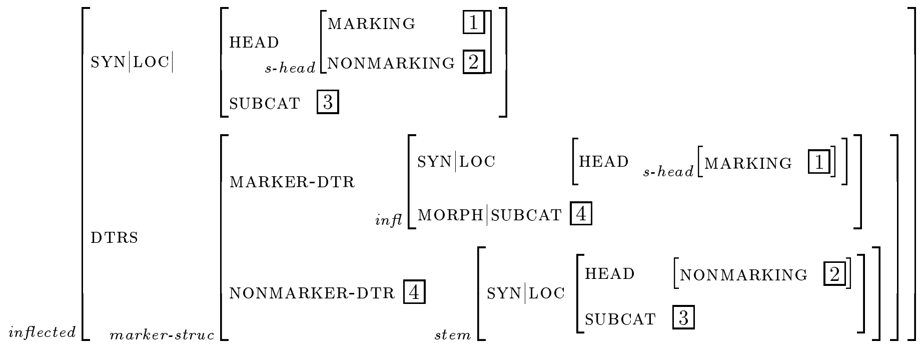


Figure 76: AVM for type *inflected*

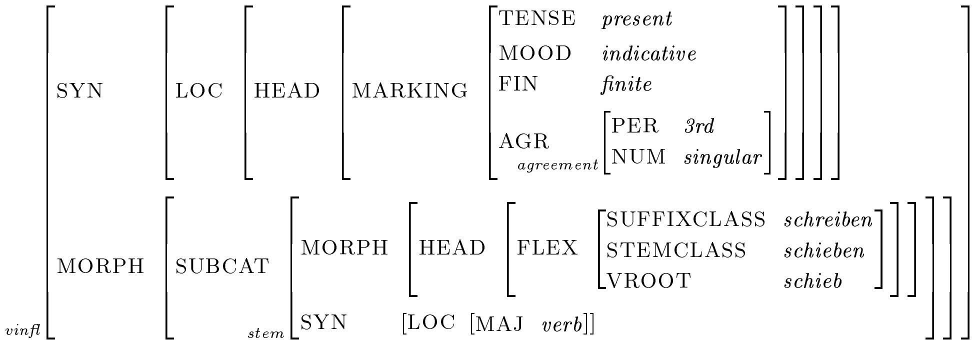


Figure 77: AVM for the suffix *e-vinfl*

stem, cf. the tag $\boxed{4}$, inherited from the MSP. In particular, the constraints on the stem's SUFFIXCLASS, STEMCLASS, and VROOT are formulated here.

2. Those syntactic agreement features that are HEAD|MARKING features for *inflected* are provided by the MARKER-DTR, cf. the tag $\boxed{1}$, inherited from the IMP.
3. Those syntactic agreement features that are HEAD|NONMARKING features for *inflected* are provided by the NONMARKER-DTR, cf. the tag $\boxed{2}$, inherited from the IMP.
4. The value of SYN|LOC|SUBCAT is shared with that of the NONMARKING-DTR, cf. tag $\boxed{3}$, expressing that all inflected forms of one lexeme have the same syntactic valency.

In order to illustrate how inflectional suffixation works, consider the example of the feature structure modelling the inflectional suffix *e-vinfl* in the AVM in Figure 77. It marks verbal word forms as *3rd singular present indicative* and is subcategorised for stems that are specified as [SUFFIXCLASS *schreiben*] and [VROOT *schieb*]. (The other specifications [STEMCLASS *schieben*] and [MAJ *verb*] are actually predictable from the first two, but we include them in our description for explicitness.) Inflectional suffixes are also appropriate for the MORPH|HEAD|DER|NAT feature, as there are nonnative inflectional suffixes such as *-um*, which combine with nonnative stems only, e.g. in the word forms `Dat#+um` and `Praktik#+um`. Nevertheless, the NAT values for inflectional suffixes are inferable from the MORPH|SUBCAT|HEAD|FLEX|SUFFIXCLASS values.

One could argue whether inflectional suffixes have their own SYN|LOC|MAJ specifications or whether they only select these on stems, cf. (van Eynde, 1994, 90ff). In the present description, the latter is the case.

5.5.3. Paradigms

The speech-oriented lemma lexicon in the Verbmobil phase II work package 1.3.1 was originally implemented in Prolog and DATR according to the theory developed in Bleiching (1994), Bleiching et al. (1996), and Bleiching and Gibbon (2000). As their theory is based on paradigmatic morphology, we want to clarify at this point if and how far their theory is equivalent to ours: we have pointed out in Section 5.3 that the backbone of their theory corresponds to our *lemma* hierarchy, but there are two main differences to note:

First, in Bleiching et al. (1996), different ablauted or umlauted roots and inflectional endings do not have the status of lexical signs, they are listed in the lemma entries as strings and serve as parameters in the complex paradigm functions (which also incorporate the stem syncretism relations in inflections). In our theory, however, they are lexical signs that are subtypes of the more abstract *root* morpheme types.

Second, their lemma hierarchy is based on a hierarchy of classes of paradigm functions, whereas our type hierarchy is partitioned along dimensions derived from the appropriateness of value types for the features SUFFIXCLASS and STEMCLASS (cf. Section 4.2). Still, their theory is weakly equivalent to our theory, because the morphological features that we employ arise from the same morphological combinatorics that lead to the paradigm function classes.

In Bleiching et al. (1996), stem syncretism classes and suffix syncretism classes for German are presented. Verbs are cross-classified under these, nouns belong only to suffix classes. In our theory, the stem classes and suffixes classes appear as morphological features of lemmata (stems), justified by the fact that inflectional suffixes are subcategorised for them (Section 5.5.2). These inflectional class features induce the lemma hierarchy over all stems in our theory, and there would be a strong equivalence between Bleiching's paradigm class hierarchy and our hierarchy of lemma types, were it not for the default inheritance employed in Bleiching's hierarchy, which allows her to stipulate fewer intermediate types/classes. But default inheritance might be regarded as a mere abbreviation device in the case of inflectional classes.

As umlauted and ablauted verbal roots are lexical signs in our theory, we need an additional feature that partitions a root morpheme into root morphs. This is the VROOT feature (Section 5.4), which percolates to complex stems by virtue of the MHFP, and verbal inflectional suffixes are subcategorised for VROOT values of stems (see Section 5.5.2). This subcategorisation information corresponds to the possible positions of the inflectional suffix string parameters in a paradigm function.

Thus, the STEMCLASS/VROOT and the SUFFIXCLASS specifications of the verbal stem and the subcategorisation specifications of the inflectional suffixes account for the occurrence of a particular stem or inflectional suffix at a particular position in a morphological paradigm.

The actual paradigm function, which is indispensable in paradigmatic morphology, has no direct equivalent in our theory. The forms that occur in a paradigm are simply licensed by the feature structure description associated with the type *word.* As a *word* may be a bare *stem* or *inflected*, the relation between a word form and its lemma distinguishes two cases:

1. Immediate type subsumption: The word form is identical with a bare stem (*vermied*), which is subsumed by the corresponding lemma node.
2. Type subsumption of a constituent: The word form contains a stem as the value of NONMARKING-DTR, whose type is subsumed by the corresponding lemma node.

Thus, a stem and its inflected forms are systematically related in our theory, though not through one particularly distinguished paradigm function. It is nevertheless possible to explicitly define classes of feature structures in the description language that denote paradigms.

5.5.4. Umlaut

Umlaut in modern German is a morphophonological alternation triggered by non-trivial *morphological* context conditions. Certain roots, but also some derivational suffixes (`tum`∼`tüm`), undergo umlaut (change a possible [+back] feature of their final major vowel to [−back]) when certain suffixes are attached). That is, only morphemes with a [−back] major vowel may umlaut, but not all of them do (cf. the word `Bagger`). Within inflection, umlaut appears in certain nominal roots in the plural, and in the subjunctive forms of strong verbs. In our morphology, the inflectional umlaut is implicitly encoded in the SUFFIXCLASS and VROOT features, but it would be easy to devise alternative verbal and nominal inflectional class hierarchies that do not take into account umlaut as a partitioning dimension. These hierarchies would then have to be supplemented by an umlaut hierarchy, and roots and suffixes would have to be cross-classified along the respective inflectional class and the umlaut hierarchy. As we do not count adjective gradation as inflection, umlaut does not play a role in adjectival inflection.

$$\begin{bmatrix} \textit{händ-n-root} & \\ \text{SURF} & \begin{bmatrix} \text{PHON} & \texttt{hEnd} \\ \text{ORTH} & \texttt{händ} \end{bmatrix} \\ \text{MORPH} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{UML} & \begin{bmatrix} \text{UMLD} & + \\ \text{UMLB} & \textit{chen-sufn} \sqcap \textit{el-sufn} \sqcap \ldots \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 78: Umlaut specifications appropriate for the nominal root `händ`

$$\begin{bmatrix} \textit{hand-n-root} & \\ \text{SURF} & \begin{bmatrix} \text{PHON} & \texttt{hand} \\ \text{ORTH} & \texttt{hand} \end{bmatrix} \\ \text{MORPH} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{UML} & \begin{bmatrix} \text{UMLD} & - \\ \text{UMLB} & \textit{chen-sufn} \sqcap \textit{el-sufn} \sqcap \ldots \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 79: Umlaut specifications appropriate for the nominal root `hand`

$$\begin{bmatrix} \textit{termin-n-root} & \\ \text{SURF} & \begin{bmatrix} \text{PHON} & \texttt{tErmi:n} \\ \text{ORTH} & \texttt{termin} \end{bmatrix} \\ \text{MORPH} & \begin{bmatrix} \text{HEAD} & \begin{bmatrix} \text{UML} & \begin{bmatrix} \text{UMLD} & - \\ \text{UMLB} & \perp \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Figure 80: Umlaut specifications appropriate for the nominal root `termin`

The very tricky umlaut conditions in derivation arise from the fact that it is **not** the case that there are a set of umlaut-triggering suffixes and a set of umlautable roots, and umlaut simply occurs every time two of these are combined, cf. the studies by Zwicky (1967), Wiese (1987), and Reinhard (1991). A counterexample to this hypothesis is the morpheme *hand*. The two suffixes *-chen* and *-lich* are in general umlaut-triggering, but in *hand*, only *-chen* causes umlaut, cf. `Händchen, handlich, *händlich`.

In Trost (1993), it is suggested that all umlautable bases must be specified for the set of suffixes that cause umlaut in them. We follow this general idea but deviate from his proposal in the following respects. In Trost (1993), feature structures in which attribute names such as CHEN-UMLAUT, ER-UMLAUT occur are employed. These attributes take boolean values and are appropriate for lexical bases. We consider the appearance of suffixes as attributes names as linguistically undesirable and prefer them to appear as feature structures associated with a type that represent lexical entries, just like in other contexts in our morphology. Thus, we introduce two features, UMLAUT|UMLAUTED (short UML|UMLD) and UMLAUT|UMLAUTABLE (UML|UMLB), which are appropriate for the type *base-final*, a subtype of *morpheme* which exactly comprises the class of roots and derivational suffixes (see the feature declarations in Section 5.6). As our morphology is morph-based, and umlaut is one dimension along which morphemes are partitioned into morphs (Section 5.4), there are [UMLD +] and [UMLD −] morph subtypes for each

$$\begin{bmatrix} \textit{chen-sufn} & \\ \text{SURF} & \begin{bmatrix}\text{PHON} & \texttt{x@n}\\ \text{ORTH} & \texttt{chen}\end{bmatrix} \\ \text{MORPH}|\text{SUBCAT}|\text{MORPH}|\text{HEAD} & \begin{bmatrix} \text{UML} & \begin{bmatrix}\text{UMLD} & + \\ \text{UMLB} & \textit{chen-sufn}\end{bmatrix} \\ \vee\ \text{UML} & \begin{bmatrix}\text{UMLB} & \perp\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 81: Umlauting specifications appropriate for the suffix `-chen`

umlautable morpheme, which can be seen in the specifications for the morphs *hand-n-root* and *händ-n-root* given in Figures 79 and 78. At the same time, umlautable morphemes (and thereby their umlauted as well as their non-umlauted allomorphs) have a disjunction of those suffix types that require their umlauted alternant as the value of UMLB. Morphemes that have no umlauted allomorphs are simply specified as [UMLB $\perp$], see Figure 80.

Every potentially umlauting suffix is then subcategorised for bases that are umlauted and have the suffix itself as a possible value of their UMLB specification, or alternatively, are not umlautable at all, see the example of the suffix *-chen* in Figure 81.

The general umlauting subcategorisation specifications for umlauting suffix such as *chen-sufn* can be seen in Figure 82. The mutual subcategorisation requirements of the umlauting suffix and an umlauted base are mirrored in the feature structure's cyclicity brought about by the tag $\boxed{1}$. By contrast, every non-umlauting suffix looks for a non-umlauted morph, or a morpheme that is not umlautable at all (Figure 83).

$$\boxed{1}\begin{bmatrix} \textit{umlauting-suffix} & \\ \text{MORPH}|\text{SUBCAT}|\text{MORPH}|\text{HEAD} & \begin{bmatrix} \text{UML} & \begin{bmatrix}\text{UMLD} & + \\ \text{UMLB} & \boxed{1}\end{bmatrix} \\ \vee\ \text{UML} & \begin{bmatrix}\text{UMLB} & \perp\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 82: Umlauting specifications appropriate for umlauting suffixes

$$\begin{bmatrix} \textit{nonumlauting-suffix} & \\ \text{MORPH}|\text{SUBCAT}|\text{MORPH}|\text{HEAD} & \begin{bmatrix} \text{UML} & \begin{bmatrix}\text{UMLD} & - \end{bmatrix} \\ \vee\ \text{UML} & \begin{bmatrix}\text{UMLB} & \perp\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 83: Umlauting specifications appropriate for non-umlauting suffixes

Thus the correct combinations are licensed, and the wrong combinations are ruled out by the MSP. The same treatment may also be applied in a morpheme-based morphology. The umlaut vowel of a root must then be represented as a morphophoneme, and the feature UMLD will be instantiated to + or − through the unification with the subcategorisation information of a suffix. The value must then be available as an MPHON feature to the two-level rule component, too.

Note that UML is a HEAD feature which percolates from a *base-final* morph to any *baseComplex* construction where it functions as the head daughter, thus accounting for the fact that e.g. the umlaut properties of the complex base `Morgenzug` are those of the simplex base `Zug`.

5.6. Feature declarations and principles for the types of lexical signs

We close the description of our theory of German morphology by listing the feature declarations, type constraints, principles, and continuation schemata for all types of lexical signs that have been presented.

5.6.1. Top level types

TYPE	CONSTRAINTS	ISA
sign	$\begin{bmatrix} \textsc{surf} & \begin{bmatrix} \textsc{phon} & \textit{phon-string} \\ \textsc{orth} & \textit{orth-string} \end{bmatrix} \\ \textsc{syn(tax)} & \textit{syntax} \\ \textsc{sem(antics)} & \textit{semantics} \end{bmatrix}$	$\top$
lexical-sign	$\begin{bmatrix} \textsc{morph} & \begin{bmatrix} \textsc{mphon} & \begin{bmatrix} \textsc{rbound} & \textit{boundness} \\ \textsc{lbound} & \textit{boundness} \end{bmatrix} \end{bmatrix} \end{bmatrix}$	*sign*
complex-sign	$\begin{bmatrix} \textsc{dtrs} & \textit{struc} \end{bmatrix}$	*sign*
simplex-sign	(absence of DTRS)	*sign*
phrasal-sign	(absence of MORPH)	*complex-sign*
word	$\begin{bmatrix} \textsc{syn} \mid \textsc{local} \mid \textsc{head} & \textit{s-head} \end{bmatrix}$	*lexical-sign*

5.6.2. Morpheme types

TYPE	CONSTRAINTS	ISA
morpheme	$\begin{bmatrix}\text{MORPH} & \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{ADJ(ACENCY)} & \textit{boolean}\\ \text{STR(ESS)} & \textit{boolean}\end{bmatrix}\\ \text{HEAD} & \begin{bmatrix}\text{DER}\vert\text{NAT(IVE)} & \textit{boolean}\end{bmatrix}\end{bmatrix}\end{bmatrix}$	*simplex-sign*
root	(absence of MORPH$\vert$SUBCAT)	*morpheme*
native-root	$\begin{bmatrix}\text{MORPH}\vert\text{HEAD}\vert\text{DER}\vert\text{NAT} & +\end{bmatrix}$	*root*
nonnative-root	$\begin{bmatrix}\text{MORPH}\vert\text{DER}\vert\text{NAT} & -\end{bmatrix}$	*root*
nominal-root	$\begin{bmatrix}\text{SYN}\vert\text{LOC}\vert\text{HEAD}\vert\text{NONMARKING}\vert\text{MAJ} & \textit{noun}\end{bmatrix}$	*root*
verbal-root	$\begin{bmatrix}\text{SYN}\vert\text{LOC}\vert\text{HEAD}\vert\text{NONMARKING}\vert\text{MAJ} & \textit{verb}\end{bmatrix}$	*root*
adjectival-root	$\begin{bmatrix}\text{SYN}\vert\text{LOC}\vert\text{HEAD}\vert\text{NONMARKING}\vert\text{MAJ} & \textit{adjective}\end{bmatrix}$	*root*
affix	$\begin{bmatrix}\text{MORPH}\vert\text{SUBCAT} & \textit{base}\end{bmatrix}$	*morpheme*
d-affix	$/\begin{bmatrix}\text{MORPH}\vert\text{SUBCAT} & \textit{stem}\end{bmatrix}$	*affix*
prefix	$\begin{bmatrix}\text{MORPH}\vert\text{MPHON} & \begin{bmatrix}\text{SEP} & \textit{boolean}\\ \text{RBOUND} & \begin{bmatrix}\text{BOUND} & \textit{toBind}\\ \text{BND-ARG} & \textit{base-initial}\end{bmatrix}\\ \text{LBOUND} & \begin{bmatrix}\text{BOUND} & \textit{opt}\\ \text{BND-ARG} & \textit{stem-final}\\ & \sqcap\ \textit{prefix}\end{bmatrix}\end{bmatrix}\end{bmatrix}$	*d-affix*
suffix	$\begin{bmatrix}\text{MORPH}\vert\text{MPHON} & \begin{bmatrix}\text{RBOUND} & \begin{bmatrix}\text{BOUND} & \textit{opt}\\ \text{BND-ARG} & \textit{suffix}\\ & \sqcap\ \textit{infl}\\ & \sqcap\ \textit{stem-initial}\end{bmatrix}\\ \text{LBOUND} & \begin{bmatrix}\text{BOUND} & \textit{toBind}\\ \text{BND-ARG} & \textit{base-final}\\ & \sqcap\ \textit{prefix}\end{bmatrix}\end{bmatrix}\end{bmatrix}$	*affix*
d-suffix		*d-affix* $\sqcup$ *suffix*

5.6.3. Affix subtypes

base-final	$\left[\text{MORPH}\vert\text{HEAD}\vert\text{UML}\ \begin{bmatrix}\text{UMLD} & \textit{boolean}\\ \text{UMLB} & \textit{suffix}\end{bmatrix}\right]$	*morpheme*
prefix1 (pre1)	$\left[\text{MORPH}\ \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{ADJ} & +\\ \text{SEP} & -\\ \text{STR} & -\end{bmatrix}\\ \text{HEAD} & \left[\text{DER}\vert\text{NAT}\ +\right]\\ \text{SUBCAT} & \textit{rkernel}\end{bmatrix}\right]$	*prefix*
prefix2 (pre2)	$\left[\text{MORPH}\ \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{ADJ} & -\\ \text{SEP} & -\\ \text{STR} & +\end{bmatrix}\\ \text{HEAD} & \left[\text{DER}\vert\text{NAT}\ +\right]\\ \text{SUBCAT} & \textit{rkernel}\\ & \sqcap\ \textit{basePre1fixed}\end{bmatrix}\right]$	*prefix*
nonnative-prefix (prenn)	$\left[\text{MORPH}\ \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{ADJ} & +\\ \text{SEP} & -\\ \text{STR} & -\end{bmatrix}\\ \text{HEAD} & \left[\text{DER}\vert\text{NAT}\ -\right]\\ \text{SUBCAT} & \textit{rkernel}\end{bmatrix}\right]$	*prefix*
particle (part)	$\left[\text{MORPH}\ \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{ADJ} & -\\ \text{SEP} & +\\ \text{STR} & +\end{bmatrix}\\ \text{HEAD} & \left[\text{DER}\vert\text{NAT}\ +\right]\\ \text{SUBCAT} & \textit{rkernel}\\ & \sqcap\ \textit{basePre2fixed}\\ & \sqcap\ \textit{basePre1fixed}\\ & \sqcap\ \textit{basePrennfixed}\end{bmatrix}\right]$	*prefix*
native-suffix (sufn)	$\left[\text{MORPH}\ \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{ADJ} & -\\ \text{SEP} & -\\ \text{STR} & -\end{bmatrix}\\ \text{HEAD} & \left[\text{DER}\vert\text{NAT}\ +\right]\\ \text{SUBCAT} & \textit{lkernel}\end{bmatrix}\right]$	*suffix*
nonnative-suffix (sufnn)	$\left[\text{MORPH}\ \begin{bmatrix}\text{MPHON} & \begin{bmatrix}\text{ADJ} & -\\ \text{SEP} & -\\ \text{STR} & +\end{bmatrix}\\ \text{HEAD} & \left[\text{DER}\vert\text{NAT}\ -\right]\\ \text{SUBCAT} & \textit{lkernel}\end{bmatrix}\right]$	*suffix*
umlauting-suffix	$\boxed{1}\left[\text{MORPH}\vert\text{SUBCAT}\vert\text{MORPH}\vert\text{HEAD}\ \begin{bmatrix}\text{UML} & \begin{bmatrix}\text{UMLD} & +\\ \text{UMLB} & \boxed{1}\end{bmatrix}\\ \vee\ \text{UML} & \left[\text{UMLB}\ \bot\right]\end{bmatrix}\right]$	*suffix*
nonumlauting-suffix	$\boxed{1}\left[\text{MORPH}\vert\text{SUBCAT}\vert\text{MORPH}\vert\text{HEAD}\ \begin{bmatrix}\text{UML} & \begin{bmatrix}\text{UMLD} & -\\ \text{UMLB} & \bot\end{bmatrix}\\ \vee\ \text{UML} & \end{bmatrix}\right]$	*suffix*
infl(ection)	$\left[\text{MORPH}\vert\text{SUBCAT}\ \ \textit{stem}\right]$	*suffix*
interfix	$\boxed{1}\left[\text{MORPH}\vert\text{SUBCAT}\vert\text{MORPH}\vert\text{HEAD}\vert\text{DER}\vert\text{INTERF}\ \ \boxed{1}\right]$	*suffix*
linking-morpheme (lm)	$\boxed{1}\left[\text{MORPH}\vert\text{SUBCAT}\vert\text{MORPH}\vert\text{HEAD}\vert\text{LINK}\vert\text{LINKBL}\ \ \boxed{1}\right]$	*suffix*

5.6.4. Lemma types

TYPE	CONSTRAINTS	ISA
lemma	$\left[\text{MORPH}\vert\text{HEAD}\vert\text{FLEX}\ \ \textit{flex}\right]$	*lexical-sign*
noun-lemma	$\left[\text{MORPH}\vert\text{HEAD}\vert\text{FLEX}\ \left[\text{SUFFIXCLASS}\ \ \textit{n-suffixclass}\right]\right]$	*lemma*
verb-lemma	$\left[\text{MORPH}\vert\text{HEAD}\vert\text{FLEX}\ \begin{bmatrix}\text{SUFFIXCLASS} & \textit{v-suffixclass}\\ \text{STEMCLASS} & \textit{v-stemclass}\end{bmatrix}\right]$	*lemma*
adj-lemma	$\left[\text{MORPH}\vert\text{HEAD}\vert\text{FLEX}\ \ \textit{adjectival}\right]$	*lemma*

5.6.5. LexComplex types

TYPE	CONSTRAINTS	ISA
lexComplex	$\begin{bmatrix}\text{DTRS} & \textit{marker-struc} \sqcap \textit{headed-struc}\end{bmatrix}$	*complex* ⊔ *lexical-sign*
baseComplex	$\begin{bmatrix}\text{DTRS} & \textit{headed-struc}\end{bmatrix}$	*lexComplex*
baseDerived	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{\$AFFIX-DTR} & \textit{d-affix}\\ \text{\$BASE-DTR} & \textit{base}\end{bmatrix}\end{bmatrix}$	*baseComplex*
baseInterfixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{\$AFFIX-DTR} & \textit{interfix}\end{bmatrix}\end{bmatrix}$	*suffixed*
baseLm	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{\$AFFIX-DTR} & \textit{lm}\end{bmatrix}\end{bmatrix}$	*suffixed*
baseCompound	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{HEAD-DTR} & \textit{base}\\ \text{NONHEAD-DTR} & \textit{baseLm}\end{bmatrix}\end{bmatrix}$	*baseComplex*
affixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{\$BASE-DTR} & \textit{base}\\ \text{\$AFFIX-DTR} & \textit{affix}\end{bmatrix}\end{bmatrix}$	*lexComplex*
suffixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{\$AFFIX-DTR} & \textit{suffix}\end{bmatrix}\end{bmatrix}$	*affixed*
baseDsuffixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{HEAD-DTR} & \textit{d-suffix}\end{bmatrix}\end{bmatrix}$	*baseDerived* ⊔ *suffixed*
baseSuffnfixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{HEAD-DTR} & \textit{native-suffix}\end{bmatrix}\end{bmatrix}$	*baseDsuffixed*
baseSuffnnfixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{HEAD-DTR} & \textit{nonnative-suffix}\end{bmatrix}\end{bmatrix}$	*baseDsuffixed*
basePrefixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{HEAD-DTR} & \textit{base}\\ \text{NONHEAD-DTR} & \textit{prefix}\end{bmatrix}\end{bmatrix}$	*baseDerived*
basePre1fixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{NON-HEAD-DTR} & \textit{prefix1}\end{bmatrix}\end{bmatrix}$	*basePrefixed*
basePre2fixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{NON-HEAD-DTR} & \textit{prefix2}\end{bmatrix}\end{bmatrix}$	*basePrefixed*
basePartfixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{NON-HEAD-DTR} & \textit{part}\end{bmatrix}\end{bmatrix}$	*basePrefixed*
basePrennfixed	$\begin{bmatrix}\text{DTRS} & \begin{bmatrix}\text{NON-HEAD-DTR} & \textit{prenn}\end{bmatrix}\end{bmatrix}$	*basePrefixed*
inflected	$\begin{bmatrix}\text{DTRS} & \textit{marker-struc}\end{bmatrix}$	*suffixed* ⊔ *word*

5.6.6. Types described by join or meet of other types

TYPE	CONSTRAINTS	ISA
base	*root* ⊓ *baseComplex*	*lexical-sign*
stem	()	*lemma* ⊔ *base*
stem-simple	()	*lemma* ⊔ *root*
stem-free	()	*stem* ⊔ *word*
base-initial	*prefix* ⊓ *root*	*morpheme*
base-final	*suffix* ⊓ *interfix* ⊓ *root*	*morpheme*
stem-initial	*prefix* ⊓ *stem-simple*	*morpheme*
stem-final	*stem-simple* ⊓ *d-suffix*	*morpheme*
rkernel	*root* ⊓ *baseDsuffixed*	*base*
lkernel	*root* ⊓ *basePrefixed*	*base*

5.6.7. Principles and Continuation Schemata

$$\textit{baseComplex} \Rightarrow \begin{bmatrix}\text{SYN}\mid\text{LOC}\mid\text{HEAD} & \boxed{1}\\ \text{MORPH}\mid\text{HEAD} & \boxed{2}\\ \text{DTRS}_{\textit{headed-structure}} & \begin{bmatrix}\text{HEAD-DTR}_{\textit{base}} & \begin{bmatrix}\text{SYN}\mid\text{LOC}\mid\text{HEAD} & \boxed{1}\\ \text{MORPH}\mid\text{HEAD} & \boxed{2}\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Figure 84: The Morphological Head Feature Principle (MHFP)

$$\textit{affixed} \Rightarrow \begin{bmatrix} \text{DTRS} \; {}_{\textit{struc}}\begin{bmatrix} \text{\$BASE-DTR} & \boxed{1} \\ \text{\$AFFIX-DTR}|\text{MORPH}|\text{SUBCAT} & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Figure 85: The Morphological Subcategorisation Principle (MSP)

$$\textit{inflected} \Rightarrow \begin{bmatrix} \text{SYN}|\text{LOC}|\text{HEAD} \; {}_{\textit{s-head}}\begin{bmatrix} \text{MARKING} & \boxed{1} \\ \text{NONMARKING} & \boxed{2} \end{bmatrix} \\ \text{DTRS} \; {}_{\textit{marker-struc}}\begin{bmatrix} \text{MARKER-DTR}|\text{SYN}|\text{LOC}|\text{HEAD}|\text{MARKING} & \boxed{1} \\ \text{NONMARKER-DTR}|\text{SYN}|\text{LOC}|\text{HEAD}|\text{NONMARKING} & \boxed{2} \end{bmatrix} \end{bmatrix}$$

Figure 86: The Inflectional Marking Principle (IMP)

$$ {}_{\textit{basePrefixed}}\begin{bmatrix} \text{SURF} & \begin{bmatrix} \text{PHON} & \text{2LEVEL}\left(\boxed{5},\boxed{1}\right) \\ \text{ORTH} & \text{2LEVEL}\left(\boxed{6},\boxed{2}\right) \end{bmatrix} \\ \text{MORPH} & \begin{bmatrix} \text{MPHON} & \begin{bmatrix} \text{RBOUND} & \boxed{7} \\ \text{LBOUND} & \boxed{4} \end{bmatrix} \end{bmatrix} \\ \text{DTRS} & {}_{\textit{headed-struc}}\begin{bmatrix} \text{HEAD-DTR} & \begin{bmatrix} \text{SURF} & \begin{bmatrix} \text{PHON} & \boxed{1} \\ \text{ORTH} & \boxed{2} \end{bmatrix} \\ \text{MORPH} & \begin{bmatrix} \text{MPHON} & \begin{bmatrix} \text{RBOUND} & \boxed{3} \\ \text{LBOUND} & \boxed{4} \end{bmatrix} \end{bmatrix} \end{bmatrix} \\ \text{NONHEAD-DTR} & \begin{bmatrix} \text{SURF} & \begin{bmatrix} \text{PHON} & \boxed{5} \\ \text{ORTH} & \boxed{6} \end{bmatrix} \\ \text{MORPH} & \begin{bmatrix} \text{MPHON} & \begin{bmatrix} \text{RBOUND} & \boxed{7} \\ \text{LBOUND} & \boxed{8} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix} $$

Figure 87: Continuation Schema for *basePrefixed*

$${}_{\textit{baseDSuffixed} \sqcap \textit{baseCompound}}\left[\begin{array}{ll}
\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \text{2LEVEL}\left(\boxed{1},\boxed{5}\right)\\ \text{ORTH} & \text{2LEVEL}\left(\boxed{2},\boxed{6}\right)\end{array}\right]\\
\text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{4}\\ \text{LBOUND} & \boxed{7}\end{array}\right]\end{array}\right]\\
\text{DTRS} & {}_{\textit{headed-struc}}\left[\begin{array}{ll}
\text{HEAD-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{1}\\ \text{ORTH} & \boxed{2}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{3}\\ \text{LBOUND} & \boxed{4}\end{array}\right]\end{array}\right]\end{array}\right]\\
\text{NONHEAD-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{5}\\ \text{ORTH} & \boxed{6}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{7}\\ \text{LBOUND} & \boxed{8}\end{array}\right]\end{array}\right]\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 88: Continuation Schema for *baseDSuffixed* ⊓ *baseCompound*

$${}_{\textit{inflected}}\left[\begin{array}{ll}
\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \text{2LEVEL}\left(\boxed{1},\boxed{5}\right)\\ \text{ORTH} & \text{2LEVEL}\left(\boxed{2},\boxed{6}\right)\end{array}\right]\\
\text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{4}\\ \text{LBOUND} & \boxed{7}\end{array}\right]\end{array}\right]\\
\text{DTRS} & {}_{\textit{marker-struc}}\left[\begin{array}{ll}
\text{MARKER-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{1}\\ \text{ORTH} & \boxed{2}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{3}\\ \text{LBOUND} & \boxed{4}\end{array}\right]\end{array}\right]\end{array}\right]\\
\text{NONMARKER-DTR} & \left[\begin{array}{ll}\text{SURF} & \left[\begin{array}{ll}\text{PHON} & \boxed{5}\\ \text{ORTH} & \boxed{6}\end{array}\right]\\ \text{MORPH} & \left[\begin{array}{ll}\text{MPHON} & \left[\begin{array}{ll}\text{RBOUND} & \boxed{7}\\ \text{LBOUND} & \boxed{8}\end{array}\right]\end{array}\right]\end{array}\right]
\end{array}\right]
\end{array}\right]$$

Figure 89: Continuation Schema for *inflected*

6. The Mclass system

In the preceding chapters we have described what could be part of the requirement specifications for a morphological processing component: a comprehensive and consistent theory of the morphology of German, covering lexical construction types for inflection, derivation, and compounding. In order to operationally evaluate at least parts of the theory, it was employed in the design of a spoken language lexicon acquisition component in the Verbmobil project.

6.1. The lexical acquisition task

In the Verbmobil project, the aim of morphological corpus processing was to automatise the acquisition of the word lemma background lexicon, from which the speech-oriented Verbmobil lexical database was subsequently generated. The background lexicon had to be acquired from the orthographic transcriptions of the Verbmobil dialogue corpora, which look like the following:[41]

```
g114axx0_010_JAK_021040: oh , das w"ar' ja sch"on<Z> , -/das w"ar'
 ja/- <A> also <P> da w"urde sich der August ja ideal f"ur eignen .
 so<Z> . <P> <#Klopfen> so langsam ausklingen lassen des Sommers ,
 <A> also <:<#Klopfen> das:> w"ar' romantisch . <hm> 'n Termin <P>
 w"urde mir ganz gut passen <Schmatzen> <"ah> <P> Montag , der
 #achte , <A> bis Freitag , der #zw"olfte , da h"att' ich dann
 irgendwann Zeit . <#Klopfen>
```

In this narrow transcription, numerous features of spontaneously spoken language are transcribed as well as extralinguistic acoustic events (see Burger (1997) for an account of the so-called 'transliteration' format). The preprocessor `trlfilter` (Gibbon and Steinbrecher, 1995) filters out all tags and comments and normalises the orthography, and the tokeniser `trl2wl` provides a file containing a list of orthographic word forms according to the spelling conventions defined in Gibbon (1995).

This is the interface to our lexical acquisition system, the task of which thus consists of constructing a *lexical lemma entry* out of a word form and link the entry the right place in the hierarchy of *lemma* and *stem* types. We have dubbed this morphological corpus analysis system for the Verbmobil lexical acquisition task MCLASS because one of its central tasks is automatic *m*orphological *class*ification. On the level of the feature structure descriptions of our theory of morphology, the input to MCLASS corresponds to the value of the SURF|ORTH attribute of a lexical object of the type *word*, and the output corresponds to one or more feature structures representing a lemma entry as in Figure

[41]Verbmobil CDROM 02, Dialog `g114a01.trl`.

$$\begin{bmatrix} \textit{noun-lemma} & \\ \text{SURF} & \begin{bmatrix}\text{PHON} & \texttt{tE6.m''i:n\#f6.+?'''aIn+ba:.r+UN} \\ \text{ORTH} & \texttt{Termin\#ver+ein+bar+ung}\end{bmatrix} \\ \text{MORPH} \mid & \text{HEAD} \mid \text{FLEX} \mid \text{SUFFIXCLASS} \quad \textit{frau} \\ \text{SYN} \mid \text{LOC} \mid & \text{HEAD} \begin{bmatrix}\text{MAJ} & \textit{noun}\\ \text{GEN} & \textit{feminine}\end{bmatrix}\end{bmatrix}$$

Figure 90: Noun entry in the lemma lexicon

90. A lemma entry may correspond to more than one node in the hierarchy of lexical stems if the rightmost morpheme in the stem is a root that has ablauted or umlauted variants. Then a small hierarchy of stems must be constructed, consisting of a node representing an underspecified stem (=the lemma), and several sub-nodes representing stems fully specified for the features SURF|PHON, SURF|ORTH, MORPH|HEAD|VROOT, MORPH|HEAD|DER|UML|UMLD, as described in Section 5.4. An example of such a lexical entry comprising a sub-hierarchy of stems is given in Figure 91.

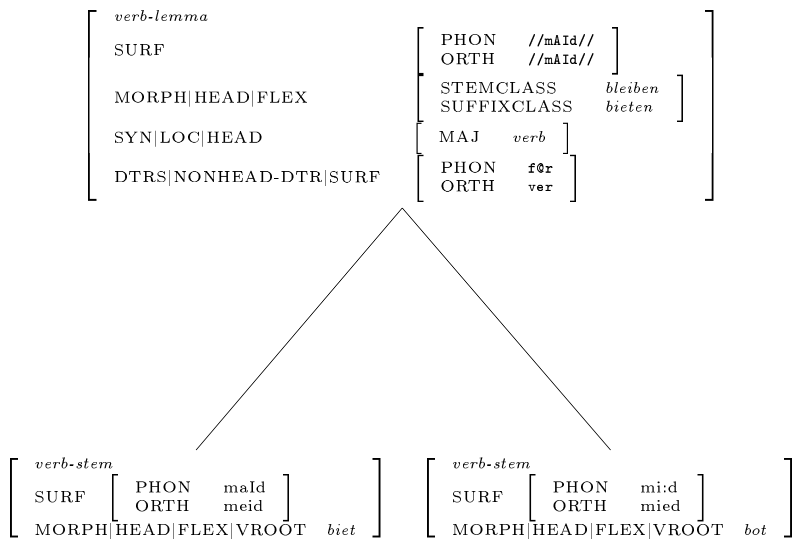


Figure 91: Verb lemma entry for *vermeiden*

Figure 92 shows the data flow between MCLASS and MCLASS-external modules.

Once the information about the words from the Verbmobil dialogue corpora has been acquired and been inserted into the lemma lexicon as described above, the vocabulary is extended and completed through automatic paradigm extension. The model and implementation of the Bielefeld paradigm generator, which generates a lexical database for all projected word forms, is documented in Bleiching et al. (1996).

Subdatabases of this lexical databases have served as the dictionaries of word recognition and prosodic analysis systems within Verbmobil, as it contains consistent phono-

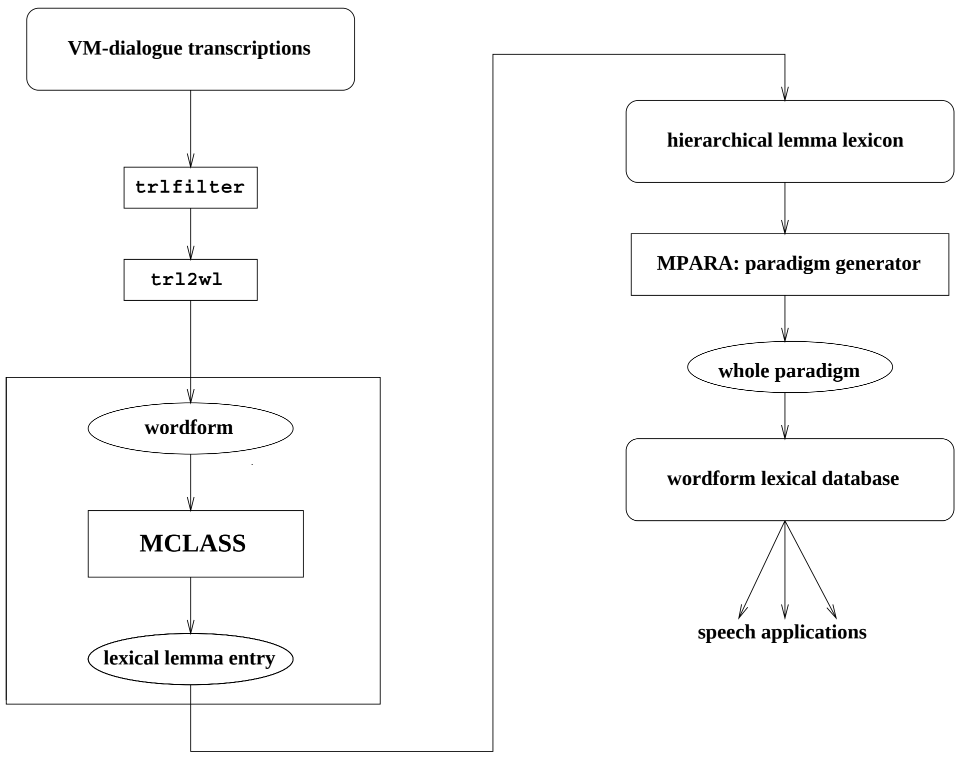

VM-dialogue transcriptions
trlfilter
trl2wl
wordform
MCLASS
lexical lemma entry
hierarchical lemma lexicon
MPARA: paradigm generator
whole paradigm
wordform lexical database
speech applications

Figure 92: Lexical acquisition in Verbmobil: Mclass and external modules

logical surface representations and prosodic information such as syllable structure and lexical stress patterns of words (which are largely determined by morphological structure, as shown in e.g. Pampel (1991), Wothke (1993), and Bleiching (1994)).

In addition, the lexical infrastructure provides lingware and and processing techniques for speech recognition with meaning-bearing subunits of words, and parts of it been actually been employed in the development of such systems by Berton et al. (1996), Lüngen et al. (1996), Althoff (1997), Strom and Heine (1999), and Pampel (1999).

6.1.1. Components of Mclass

The lexical classification requires morphological parsing of the input word form. That is, segmenting the input string into its constituent stems and morphs and assigning it a feature structure description according to our morphology, which, via the DTRS attributes, includes descriptions of the attribute values of all its constituents. From the information obtained, a fully-fledged lexical lemma entry may then be constructed by consulting the stem hierarchy once more. Therefore, the central component of Mclass is its morphological parser.

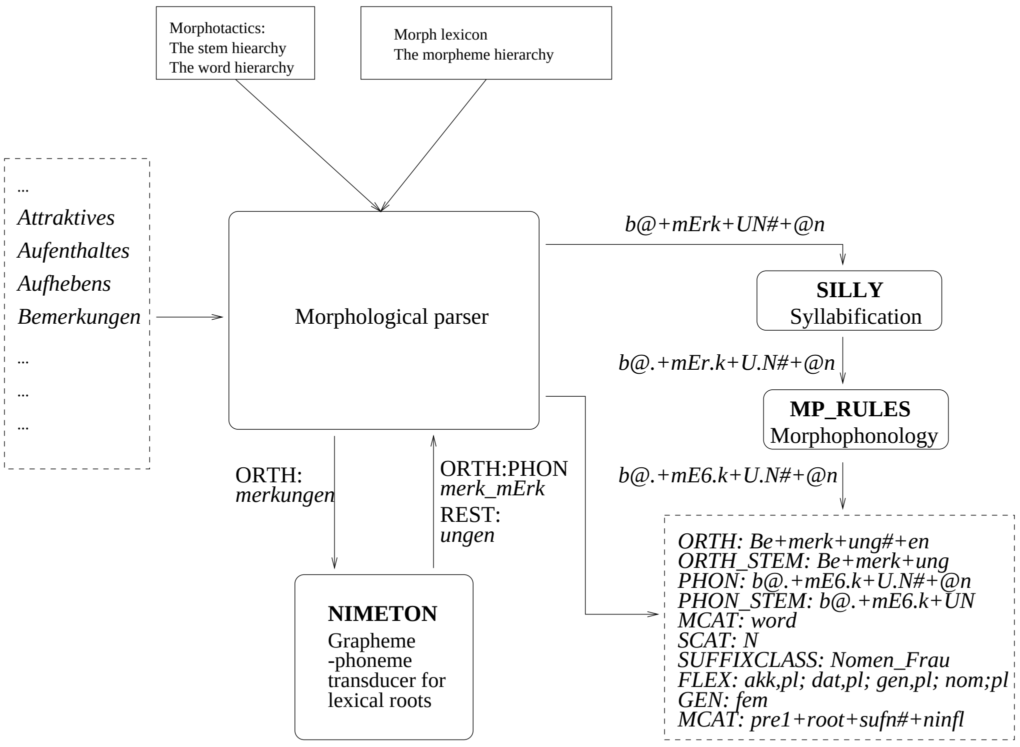


Figure 93: Lexical acquisition in Verbmobil: Internal modules of MClass

A parsing system traditionally uses a grammar and a lexicon as knowledge bases. In the case of morphological parsing, this would correspond to a word grammar/morphotactics and a morpheme/morph lexicon. To be more precise with respect to our morphological theory, the morphotactics consists of all possible morphological construction types and their descriptions, that is, types under the morphological *base* and *stem* hierarchies (describing derivatives and compounds) and types under the word hierarchy (describing inflections). Figure 93 shows the internal knowledge bases and processing modules of MClass. In addition to grammar, lexicon, and parser, there are other components which relate not directly to an evaluation of our morphological theory, but nevertheless serve functionalities in the Verbmobil lexical acquisition task: mp-rules, and silly (Matthiesen, 1998) and a (so far unrealised) stress assignment component are employed to compute the phonology values complete with prosodic information. For improving the robustness of the system, the component Nimeton is invoked (Matthiesen, 1996), when a part of the input string does not match any lexical string. Nimeton checks whether the respective substring starts with a potential root morph, and if so, its phonology is computed. Nimeton uses a finite state transducer representing a grapho-phonological root network for German. The MClass parser continues the analysis under the assumption that it has just retrieved a root morph entry from the lexicon (underspecified for certain features). This procedure is justified because all other morph classes are closed and exhaustively inventorised in our morph lexicon.

6.2. Data structures

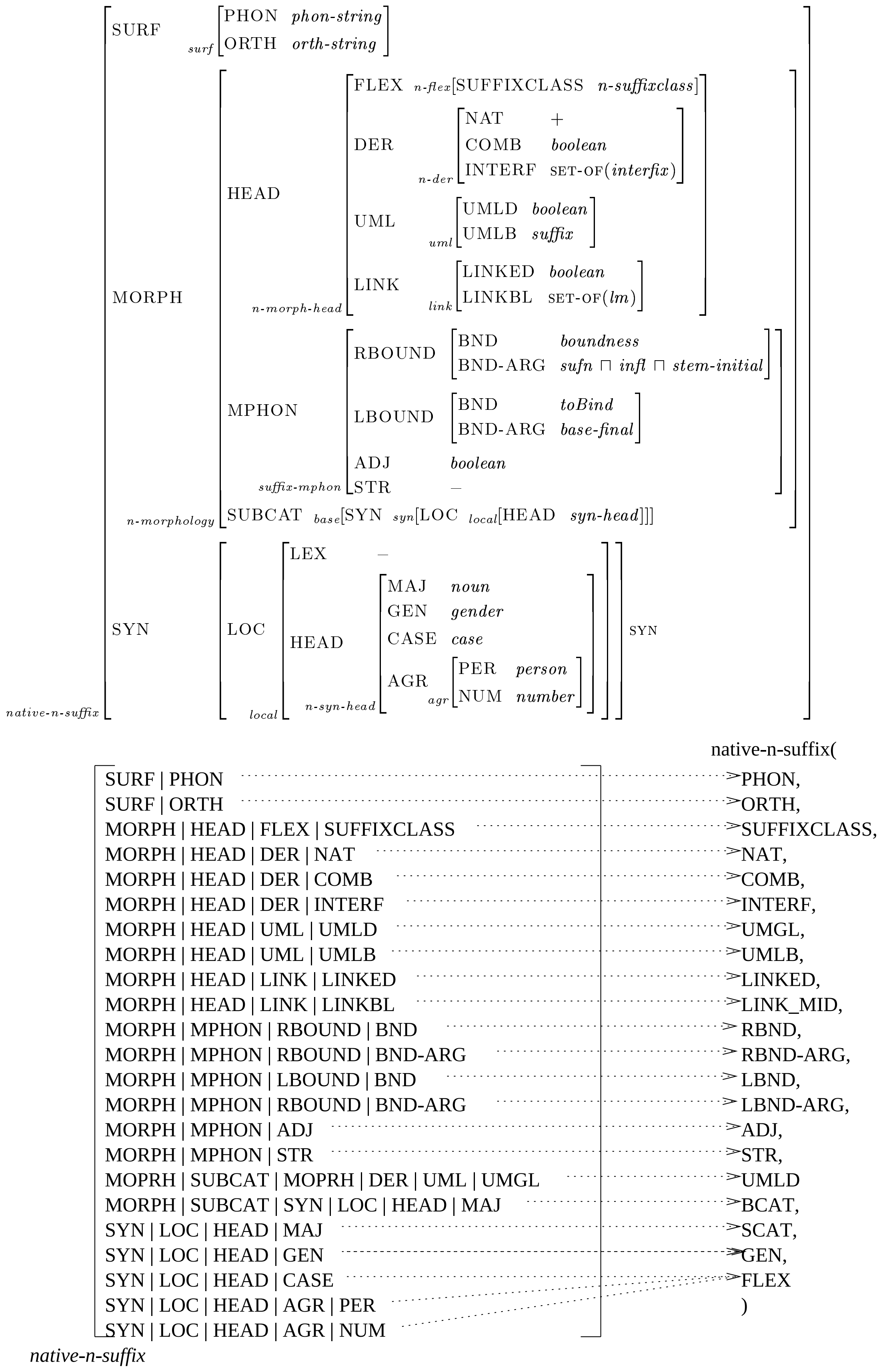
6.2.1. DCG terms and typed feature structures

We chose Prolog for implementing the MClass parser, because it is a logical programming language widely used for grammar representation and natural language parsing. The Prolog interpreter is a theorem prover for Horn Clause Logic and employs *term unification* as one built-in operation, which is about the closest thing to feature structure unification offered by a general programming language. The formalism of *Definite Clause Grammar* (DCG, Pereira and Warren, 1980) has been developed basically for easy implementation of NLP grammars in Prolog, but in principle it is an implementation-independent grammar formalism like PATR-II (Shieber, 1986), although not a fully-fledged grammar framework and linguistic theory like HPSG. In the following, we show how our abstract data structures such as feature structure descriptions and type hierarchies and their interrelations in our morphological theory are rendered in a DCG.

With some modifications, our feature structure descriptions are represented as flat *term structures* in a DCG, i.e. as predicate-argument structures such as $p(a1, a2, ..., a_n)$ which in comparison with feature structure descriptions have the following properties.

1. Features do not appear as attribute-value pairs. Instead, a value is either an atom or a variable in certain argument position, and the feature is represented by that position.
2. Term structures do not appear as the value of a variable i.e. no feature paths are represented in flat terms. (This is the property of being flat, which we have imposed on the terms ourselves. It is not an original property of Prolog, where nested ('compound') terms can be represented without problems).
3. The number (i.e. arity of a predicate) and order (i.e. argument positions) of features in a term is fixed. Each time reference is made to a term, all its arguments have to be represented, if only by anonymous variables.
4. The type symbol of a typed feature structure appears as the functor of a term.

Figure 94 illustrates the principle how our feature structure descriptions were mapped onto flat DCG terms, by the example of the AVM depicted earlier in Figure 16: In a first step, all substructuring is converted by spelling out each path (similar to DATR or PATR notation). Then each path is assigned a position in a term, filled by a variable (if the value of the path was a type that has some subtypes) or an atom (if the value was one of the lowest possible types).



SURF surf PHON phon-string
ORTH orth-string
MORPH n-morphology HEAD n-morph-head FLEX n-flex[SUFFIXCLASS n-suffixclass]
DER n-der NAT +
COMB boolean
INTERF SET-OF(interfix)
UML uml UMLD boolean
UMLB suffix
LINK link LINKED boolean
LINKBL SET-OF(lm)
MPHON suffix-mphon RBOUND BND boundness
BND-ARG sufn ⊓ infl ⊓ stem-initial
LBOUND BND toBind
BND-ARG base-final
ADJ boolean
STR –
SUBCAT base[SYN syn[LOC local[HEAD syn-head]]]
SYN LOC local LEX –
HEAD n-syn-head MAJ noun
GEN gender
CASE case
AGR agr PER person
NUM number
SYN
native-n-suffix
native-n-suffix(
SURF | PHON → PHON,
SURF | ORTH → ORTH,
MORPH | HEAD | FLEX | SUFFIXCLASS → SUFFIXCLASS,
MORPH | HEAD | DER | NAT → NAT,
MORPH | HEAD | DER | COMB → COMB,
MORPH | HEAD | DER | INTERF → INTERF,
MORPH | HEAD | UML | UMLD → UMGL,
MORPH | HEAD | UML | UMLB → UMLB,
MORPH | HEAD | LINK | LINKED → LINKED,
MORPH | HEAD | LINK | LINKBL → LINK_MID,
MORPH | MPHON | RBOUND | BND → RBND,
MORPH | MPHON | RBOUND | BND-ARG → RBND-ARG,
MORPH | MPHON | LBOUND | BND → LBND,
MORPH | MPHON | RBOUND | BND-ARG → LBND-ARG,
MORPH | MPHON | ADJ → ADJ,
MORPH | MPHON | STR → STR,
MOPRH | SUBCAT | MOPRH | DER | UML | UMGL → UMLD
MORPH | SUBCAT | SYN | LOC | HEAD | MAJ → BCAT,
SYN | LOC | HEAD | MAJ → SCAT,
SYN | LOC | HEAD | GEN → GEN,
SYN | LOC | HEAD | CASE → FLEX
SYN | LOC | HEAD | AGR | PER
SYN | LOC | HEAD | AGR | NUM
)
native-n-suffix

Figure 94: Mapping feature structure descriptions on DCG flat terms

In the terms that we actually used, the argument order is different, and there are more variables than there are attribute names in the original AVM, as *all* attributes *ever* occurring in *any* feature structure description of our HPSG morphology are represented in *each* DCG term. Each DCG term used in our theory thus has the same arity, namely 38 argument positions, cf. the terms in the example DCG rule in Figure 96. This is because we want to exploit the above mentioned term unification, or list unification, respectively, in our Prolog implementation. Two Prolog terms T_1, T_2 unify only if their functors and their arity are identical. Then, the i-th argument of T_1 unifies with the i-th argument of T_2. Prolog list unification works accordingly, only lists do not have a functor.

By having feature structure descriptions represented as flat Prolog terms in the fashion described above, the term/list unification operation of the Prolog interpreter can be employed to fulfil the essential functions of feature structure unification. The gain is improved efficiency: "Generally, the time complexity of term unification is linear on size of the terms, whilst for DAGs it is $\mathcal{O}(n^2)$ in the worse due to the need to search the DAGs for corresponding feature labels and recursively traverse the structures." (Schöter, 1993, p.6).

6.2.2. DCG rules and feature structure descriptions for complex signs

In principle, for all feature structure descriptions that represent maximal lexical types in our theory, there is a corresponding term in the DCG, appearing on the RHS or the LHS of a DCG rule.[42] But in addition, the binary DCG rules themselves represent the feature structure descriptions for *complex* signs, i.e. those that have a DTRS attribute, i.e. all maximal types under the *lexComplex* hierarchy. The term on the LHS of a DCG rule represents the actual complex sign, and the terms on the RHS of the DCG rule represent its daughters. In our theory, feature structures that represent complex signs are re-entrant. That is, feature structure unification occurs in descriptions of these, inherited from the MHFP, the IMP, the MSP, or a Continuation Schema, appearing as tags in the corresponding AVM notations.

In our DCG rules, the re-entrancies are rendered as unificational annotations, resulting in a PATR-II-like notation. Note that in principle, the sources of such annotated equations may only be one of the three morphological principles, the MHFP, the IMP, the MSP, or a Continuation Schema.[43] Compare the AVM for the morphological construction type *basePartfixed* given in Figure 95 with the DCG rule derived from it given

[42]In our discussion, we avoid the terms *head* and *body*, which are generally used for the parts of a Prolog rule.

[43]There may be a fifth set of annotational statements, where restrictions on feature values of daughters (RHS categories) are encoded. But these annotations are redundant, as the respective features

$$
{}_{\mathit{basePrefixed}}\begin{bmatrix}
\text{MORPH} & {}_{\mathit{base\text{-}morphology}}\begin{bmatrix} \text{HEAD} & \boxed{2} \\ \text{MPHON} & \begin{bmatrix} \text{LBOUND} & \boxed{4} \\ \text{RBOUND} & \boxed{5} \end{bmatrix} \end{bmatrix} \\
\text{SYN} & \text{LOC : HEAD : } \boxed{3} \\
\text{DTRS} & {}_{\mathit{headed\text{-}struc}}\begin{bmatrix}
\text{NON-HEAD-DTR} & {}_{\mathit{vpart}}\begin{bmatrix} \text{MORPH} & {}_{\mathit{affix\text{-}morphology}}\begin{bmatrix} \text{MPHON} & \begin{bmatrix} \text{LBOUND} & \boxed{4} \end{bmatrix} \\ \text{SUBCAT} & \boxed{1} \end{bmatrix} \end{bmatrix} \\
\text{HEAD-DTR} & \boxed{1}\ {}_{\mathit{base}}\begin{bmatrix} \text{MORPH} & {}_{\mathit{base\text{-}morphology}}\begin{bmatrix} \text{MPHON} & \begin{bmatrix} \text{RBOUND} & \boxed{5} \end{bmatrix} \\ \text{HEAD} & \boxed{2} \end{bmatrix} \\ \text{SYN} & \text{LOC : HEAD : } \boxed{3} \end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

Figure 95: The construction type *basePartfixed*

in Figure 96:[44] Since in the DCG flat terms, reference cannot be made to whole substructures, there are groups of unificational annotations, each group corresponding to one pair of tags in the AVM.

In DCG rules, like in PS rules and unlike in HPSG feature structures, the order of the categories on the RHS represents the linear precedence of constituents. The order in our DCG rules is derived from the values of RBOUND and LBOUND and the unifications according to the Continuation Schema for the respective type of complex lexical sign (Section 5.3.4). All our DCG rules representing our feature structure descriptions of morphological complex signs are documented in Appendix B. They were, however, never intended for direct parsing with the Prolog interpreter. Instead, we have implemented the *BUP parser* (Matsumoto et al., 1983; Matsumoto and Kiyono, 1985), which requires a different rule format that is not so convenient for a grammar writer, though. The BUP rule format is automatically compiled from our DCG, this is the reason why our DCG rules are slightly insufficient for direct parsing with the Prolog interpreter. Apart from the fact that they contain left-recursive rules, two components are missing:

1. Each rule would need to have the orthographic and phonological string append relations annotated:
   `append(X1_ORTH, X2_ORTH, X0_ORTH),`
   `append(X1_PHON, X2_PHON, X0_PHON).`
   Alternatively, variables representing open lists could have been introduced; this is what was done in the BUP implementation (see Section 6.4).

are already type-checked in the morph lexicon, as explained in Section 6.2.3. These type-checking annotations in DCG rules annotations stem from a former stage of the implementation and have been retained as a kind of safety check.

[44]A morphological base (cf. Section 1.6) is called a "`stem`" in the DCG.

```
% Rule3a            stemPrefixed       -->  part  rkernel
% Example           anbrenn            -->  an    brenn
% Example           "uberinterpretier  --> "uber  interpretier

% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,
X0_UMLB,X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,
X0_INTERF_ARG,X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,
X0_FLEX,X0_PARTSTEM,X0_PSTEMS,X0_OSTEMS):-

     part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,
X1_INTERF_ARG,X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS),

     rkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,
X2_INTERF_ARG,X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,
X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Category instantiation
  X0_MCAT=stemPrefixed,

% MSP Morphological Subcategorisation Principle
% X1_BCAT=X2_SCAT,

% Type checking of X1 instantiations
% not(X1_LBND=='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Type checking of X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% CS Continuation Schema
  X1_LBND=X0_LBND,
  X2_RBND=X0_RBND,

% MHFP Morphological Head Feature Principle
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.
```

Figure 96: DCG-rule for *basePartfixed*

2. The morph lexical entries would have to be available in DCG format as part of the DCG, too. However, they are stored in our morph database format described in Section 6.2.3.

We have stressed that the DCG contains rules that represent the base types of our HPSG morphology. But where are the feature structure descriptions associated with types higher up in the *baseComplex*, *stem* and *word* hierarchies, and how is the inheritance of features, value type appropriateness specifications, and morphological principles implemented? The answer is, indirectly. For the implementation, we have introduced DCG rules for the lowest types of our *lexComplex*, *stem* and *word* hierarchies, such as *basePre1fixed*. That means that horizontally, the DCG covers the whole morphological theory developed in this thesis, i.e. all morphological construction types for German, i.e. the lowest types of the hierarchy in Figure 66. Some vertical information, however, i.e. abstract types such as *stem* or *baseDerived*, are not represented, and the inheritance of features introduced at these types in our theory is not implemented. Instead, each possible feature is spelt out as a variable name in each DCG term, as described above. Since the higher construction types are not represented in the DCG version, the morphological principles had to be introduced separately and multiply in each DCG rule by the DCG writer. This could be done in a systematic way, though, as our theory provides the consistent guidelines. The appropriateness of features and the typing and type checking of feature values is guaranteed by the method the *morph* lexicon was acquired.

6.2.3. Simplex signs and the morph database

The morph lexicon is first stored as a UNIX database in the form of a large ASCII table with newline as record delimiter and space as the field delimiter, where each field in a record represents one attribute appropriate for the types of *morph* signs according to our theory. There is one field for each attribute that may occur, plus an identifier, an example and a source field, these make up 41 fields. The second field contains the type symbol of a morph, and only those fields that are associated with attributes appropriate for the morph type specified in the type symbol are filled with appropriate values, whereas the remaining fields contain the "empty" symbol '~'. This way, the flattened morph lexical entries for the DCG are encoded. The bottom-up part of the parser ensures that features not appropriate for a construction, or values not appropriate for a feature are never instantiated during the parsing process.

There is an independent lexicographic tool for the interactive acquisition of morph database records called IAMW (*Interactive Acquisition of Morphological Knowledge*), see Lüngen et al. (1998) and Ehlebracht (1999). IAMW is implemented in Prolog, too, and uses the type/feature appropriateness specifications of our theory of morphology in the form of implicational statements (i.e. like the feature co-occurrence restrictions of GSPG) implemented as Prolog rules. Field value acquisition for a database record is

```
schutz_root_n root S'Uts schutz ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ \
str+ ~ linkImp ~ umgl- umlb+ ~ ~ ~ ~ ~ ~ ~ interf- ~ m ~ ~ ~ ~ Nomen_Jahr \
akk,sg;nom,sg S'Uts schutz ~ ~ Schutzbehauptung cd01

morf(root,[115,99,104,117,116,122|X],X,[83,39,39,85,116,115|Y],Y,
[schutz_root_n,root,_,'lbndOpt',['d','c'],'rbndOpt',['d','c','f'],_,n,'+',
_,'+',_,'linkImp',[_],'-','+',_,_,_,_,_,_,_,'-',_,m,_,_,_,_,'Nomen_Jahr',
'akk,sg;nom,sg',_,_,_],['S''Uts'],[schutz],['SUts'],[schutz]).
```

Figure 97: Morph database entry and generated BUP lexical morph entry

viewed as a classification task and is guided by the traversal of a decision tree. Starting from the root of the tree, at each node it is first attempted to infer the next feature value from the knowledge base, and if this is impossible, the lexicographer is requested to choose a feature value from the set of appropriate values, which is presented to him/her. The value obtained then determines which arc starting from the present node must be traversed to reach the next node, i.e. the next field value acquisition. When a final leaf of the tree is reached, a morph is fully classified, and all the feature appropriateness specifications for the corresponding morph type as well as the type restrictions on the feature values have been observed.

The morph lexicon presently contains value specifications of 5700 morphs, including 405 different combinatorial readings of derivational affixes, which are fully inventorised, and the about 3600 lexical roots from the Verbmobil corpora. IAMW is thus mainly used to classify new root morphs, as this is the the only open class of morphs.

The entries for inflectional suffixes, though, i.e. specifications for their orthographic and phonological form, morphosyntactic feature values, and morphological subcategorisation specifications for the STEMCLASS, SUFFIXCLASS, and VROOT features, were acquired independently and fully automatically from the existing paradigm class hierarchy knowledge base described in Section 5.5.3.

From this morph database, the lexical entries used by the morphological parser are subsequently generated by converting a database record into a Prolog term, where the the predicate is `morf/38`, and the arguments are derived from the field values. These Prolog terms match the BUP format used in the parser, i.e. are not directly compatible with the DCG rules described so far.

6.3. The left corner parsing algorithm

The Prolog inference machine may be directly employed for parsing using a regular DCG. However, Prolog's standard procedure calling and backtracking strategy result in a simple top-down parser with depth-first search, which is known to have several drawbacks: It may take a large number of rule expansions before it can be established that an input substring matched in the lexicon does not belong to the expected terminal category and thus that one of the rules that led to it was chosen wrongly. Moreover, in different processing states the same expansions may be tried over and over again, because no bookkeeping about intermediate results is provided. When the input string is processed from left to right (which it typically is), the DCG must not contain left-recursive rules, as then the parsing process might not terminate.

Likewise, bottom-up parsers using the Prolog interpreter and its backtracking strategy can be devised, (cf. e.g. Naumann and Langer, 1994, p.63ff), which permit processing of regular left-recursive rules in the DCG. Still, the basic shift-reduce parser may require massive backtracking because wrong partial analyses may be built up and rejected only at a relatively late stage of the analysis. The possibility to store intermediate results is not given, either, and numerous analyses may have to be made multiply.

As an alternative, we have implemented the so-called *left-corner algorithm* (described e.g. in Naumann and Langer, 1994, p.83ff), which provides a combination of the two basic parsing approaches. The left corner of a context-free phrase structure rule is the first category symbol (be it a terminal or non-terminal category) of its RHS. Defining the left corner relation on the category set of a grammar, a left corner of a category C is a category which is the first symbol on the RHS of a rule whose LHS is C. The basic idea is that a rule is invoked for top-down parsing only when its left corner category parse tree has already been built bottom up, in that case the categories following the left-corner in the rule become the new top-down *goal* categories. In the beginning, the goal is initialised with the start symbol of the grammar. There are three basic operations (cf. Arnold, 2001):

Scan: Try to analyse an initial portion of the current input string as the current goal by retrieving the category of the first word of the input string from the lexicon. Then check, which rules have this category as their left corner.

Predict: The categories following the left corner on the RHS of one of the found rules become the next current goals for Scan one after the other, and the remainder of the string the current input string. When all categories of the RHS have been successfully scanned, you can look up (i.e. scan or succeed with) the LHS of the current rule.

Succeed: When the LHS category found in Predict matches the current goal, this goal succeeds (is removed from the list of goal categories). (Alternatively, when all possible Scans for the goal have been tried out without success, the current goal fails).

The algorithm has the following attractions:

- Though its worst case behaviour is exponential just like that of ordinary bottom-up algorithms, it is in practice remarkably more efficient through the combination of data-driven and expectation-driven parts.
- For the same reason, it has some psychological plausibility (cf. Arnold, 2001).
- It can still be implemented straightforwardly in Prolog, using the Prolog back-tracking strategy to ensure the pursuit of all alternatives.
- The performance can be improved through the introduction of a reachability look-ahead (the *link relation*, also called the *oracle*). The link relation is the reflexive and transitive closure of the left corner relation on the category set and pre-computed from the grammar. It is then used as an additional knowledge base for the parser (cf. Naumann and Langer, 1994, p.93) to be consulted at the beginning of each Predict: Before a next goal is analysed (becomes the new current goal), it is checked whether the target LHS for this next goal is actually 'linked' to the current goal, i.e. whether it is a left corner of the left corner of the ... of the current goal.
- The performance can be even more improved through maintaining a well-formed substring table (WFST) during the parsing process, which makes the worst case behaviour equal to that of a chart parser, namely cubic. A lookup in the WFST is performed at the beginning of each Predict, and when an analysis of the current goal at the current position in the input string is found in it, Predict is already completed (and information found earlier re-used). Otherwise, the processing continues with the original Predict. The WFST is filled, naturally, after each Succeed (or fail), with the the goal, the analysis of the goal, and the current position in the input string.

6.4. Implementation: The BUP parsing system

All of the above features are included in Matsumoto's Prolog left-corner parser called BUP (Matsumoto et al., 1983; Matsumoto and Kiyono, 1985). (BUP stands somewhat misleadingly for *Bottom-Up Parser*.) We have adapted and enhanced it for the processing of our feature-based morphotactic DCG. The basic idea is to convert the DCG rules into a different Prolog format where the calling part of a rule actually is its left corner. That way, the step of explicitly consulting the rule knowledge base to look up a rule via its left corner is omitted. The actual parser then consists only of the the central `goal`-predicate.

In the following smallish example, we illustrate how the BUP clauses are obtained from the DCG after (Matsumoto et al., 1983, p.148). (We have adapted the example to morphology and added the link lookup and the treatment of feature unificational statements.)

Suppose that we have the following DCG morphotactics:

```
word(FS0):-     stem(FS1), suffix(FS2),           % rule1
                {rule1 unificational equations}

stem(FS0):-     morf(bild,FS0).                   % rule2

suffix(FS0):-   morf(ung,FS0).                    % rule3
```

They correspond to the following BUP clauses:

```
%% (1) BUP rule
stem(G,X,Z,FSZ):-
                link(word,G),
                goal(suffix,X,Y,FSY),
                word(G,Y,Z,FSY,FSZ).

%% (2) Dictionary entries
morf(stem,[98,105,108,100|X],X,FS).
morf(suffix,[117,110,103|X],X,FS).

%% (3) Termination clauses
word(word,X,X,FS,FS).
stem(stem,X,X,FS,FS).
suffix(suffix,X,X,FS,FS).
```

Consequently, the `goal` predicate looks like the following:

```
goal(G,X,Z,FSZ):-
        morf(C,X,Y,FSX),
        P =.. [C,G,Y,Z,FSX,FSZ],
        call(P).
```

Every DCG rule is converted into a clause such as in (1), henceforth called a BUP rule. (For unary rules, the goal part is simply omitted). Note that the rule is to be called by the predicate that represents the left corner of the original DCG rule (`stem`). First, it is checked whether there is a link relation between the target LHS of the original DCG rule (`word`), and the current goal `G`. Then the goal predicate is called with the category of the second category of the RHS of the original DCG rule (`suffix`) as the goal for a new Predict. The final clause invokes a new BUP rule with the LHS category of the original DCG rule (`word`). (Alternatively, a fact such as under (3) may be found and lead to `Succeed`.)

For every morph in the lexicon, a `morf` clause such as the ones under (2) is included. The BUP `morf` clauses are generated directly from the morph database described in Section 6.2.3. Note that e.g. the orthographic string `bild` appears as the open list of ASCII characters in decimal code, `[98,105,108,100|X]`. This is an approach differing from Matsumoto's system (devised for syntactic parsing), where the lexical entries are represented by open lists of atoms which represent whole words, e.g. `[elapsed|X]`. An input string for the original BUP system is represented as a lists of words, where the words are already separated by commas, which makes sense because in English running text the words usually occur delimited from each other. However, the input to a morphological parser are word forms, and these are usually not already segmented into morph atoms in running texts, thus we cannot start out with a list of morphs (which would be the equivalent to a list of words in syntactic parsing). The PC-KIMMO-2 system by Antworth (1994), for example, therefore operates in two phases. The first one is the segmentation phase, where a Two-level morphology is employed, and the second one the actual parsing phase, where a feature-based word grammar is employed. The latter is as we know more powerful and can describe hierarchical structures and feature constraints on these. But word grammars include the linear precedence morphotactic information contained in the continuation class network of a Two-level morphology, thus the feature grammar should actually replace (not supplement) the continuation class network, as exemplified in Ritchie et al. (1992), and also applied in our implementation:

When in our BUP parser the string "`bildung`" is looked up in the dictionary, it matches `[98,105,108,100|X]`, i.e. the orthography given in the entry for `bild`. The analysis then continues with the tail list, i.e. the string "`ung`". Thus the dictionary lookup of the parser constitutes the morphological segmentation in our system, using the feature grammar which is the one and only morphotactics in the parser. And as we have defined the base level of our morph hierarchy to correspond to orthographic morphs (Section 5.4), we need not apply two-level rules at this point. The algorithm can however be extended such that two-level rules are invoked exactly at this lexicon lookup and segmentation stage, should it be necessary, e.g. for a different language.

With the termination clauses under (3), the condition for `Succeed` is implemented, i.e. that the tail input string is empty and the freshly analysed category is identical to the current goal. (Absence of a matching termination clause causes a `fail`).

The central BUP `goal` predicate invokes a new `Scan` phase. The category of a newly found lexicon entry is the argument represented by the variable `C` in the `morf` call. To call a new BUP rule by it, the 'univ' operator `=..` has to be applied (Prolog does not allow for variables to range over functors).

The (meta-)variables `FS0`, `FS1`, and `FS2` in the DCG example represent the arguments representing the sets of morphological features; and `FS`, `FSX`, `FSY`, and `FSZ`, in the BUP code represent those features, collected in a list (we chose to store all features derived from a DCG term in one list for easier reference to the whole set of features). The unificational equations annotated to a DCG rule are expressed by identical variable names in the BUP implementation (this cannot be seen in the example above, but cf. the DCG and BUP codes in the appendices to this thesis).

In the following, we explain briefly the enhancements we made to the original BUP code, which are not shown in the example above.

Structure Building For structure building, we have introduced additional arguments in the terms representing the morphological categories. The structures built are:

1. morphologically segmented orthography strings (`An#stell+ung#+en`)
2. morphologically segmented phonology strings (e.g. `?'an#St'E.l+U.N#+@n`, with morphological border characters)
3. labelled bracketing structures with morphological categories and orthographic morph strings (e.g. `[word, [stemSuffixed, [stemPrefixed, [part, an],[root, stell]],[sufn, ung]] ,[n_infl, en]]`)
4. labelled bracketing structures with morphological categories and phonological morph strings (e.g. `[word, [stemSuffixed, [stemPrefixed, [part, ?an],[root, StEl]], [sufn, UN]], [n_infl, @n]]`)

i.e. altogether four types of structure. For every type of structure, two argument positions are introduced in each term in each rule. They represent the substructure for the category of the string prefix of the current input string which could be retrieved from the lexicon, and the substructure found, or to be found, for the category of the respective tail string. Thus, they are used in parallel with the `X,Y`, and `Z` variables in the BUP rules, which represent the current strings and tail strings. New string-operative structure building predicates (`constrStruct` and `constrSeg`) are then called within the goal predicate.

Split of the `goal` predicate As a consequence, the `goal` predicate had to be split into two clauses, one where right-bracketing structures are built (to be built before `goal` is called), and one where left-bracketing structures are built (these can only be built after `goal` has succeeded).

6.4.1. Compiling BUP from DCG

To compile this morphological parser following Matsumoto's BUP system from our morphological DCG described in Section 6.2, theoretically also Prolog could have been used.

We do not, however, have a debugging system as presented in Matsumoto and Kiyono (1985) available for the BUP parser, i.e. we had to use the Sicstus Prolog built-in debugging tools. During a trace session, it is quite important for the programmer to be able to interpret the feature variables by their “speaking” names such as `X1_CASE` or `X2_STEMCLASS`, as it is far too tiresome to figure out every feature by verifying its position in the very large feature list. It is however not at all straightforward to compile the BUP code out of the DCG using Prolog without all variables being changed into internal variables (which are identified only by numbers). Hence, we have implemented the compiler in Perl, which makes it easier to treat Prolog variable names as strings.

The BUP rules are generated directly from the DCG rules. The main steps of processing one DCG rule are given in the following:

- Store the parts of the DCG rule in the arrays `@lhs`, `@rhs_left`, and `@rhs_right`. (The latter is omitted if the DCG rule is unary).
- Read the annotational equations into a two-dimensional array.
- Substitute and instantiate the variables in `@lhs`, `@rhs_left`, and `@rhs_right` according to the annotational equations, such variables unified by ‘`=`’ receive identical names, and the remaining become the anonymous variable ‘`_`’.
- Print out `@lhs`, `@rhs_left`, and `@rhs_right` such that they form a BUP rule (i.e. `@rhs_left` first etc.).

Furthermore, for each functor (morphological category) occurring in the DCG, a BUP termination clause is generated. The goal predicate is inserted independently at the end of the BUP code file (i.e. it need not be derived from the DCG). The BUP `morf` clauses are generated from the morph database and stored in a different `.pl` file. The link relation knowledge base is generated by a different compiler written in Prolog, following (Naumann and Langer, 1994, p.280)

In the appendices to this thesis, the following source files and generated file are provided as a reference:

1. The morphological DCG (Appendix B)
2. Perl source code of the BUP parser compiler (Appendix C)
3. The BUP parser (generated code, cf. Appendix D)
4. Morph database sample (Appendix E)

6.5. Pre- and Post-processing components

Apart from the parser that forms the core of MCLASS, it contains a pre-processor (written in Prolog) in which the input is pre-processed and certain facts are asserted that may be consulted during parsing, or by the post-processor (s.b.). The pre-processor mainly treats the parameters that are passed in the call of the top-level predicate `mclass/9`:

1. *Nimeton* (`on|off`): Analysis of unknown substrings in the input by the component NIMETON (see Section 6.1.1).
2. *Number-of-hypotheses* (integer): The desired maximum number of alternative analyses per word.
3. *Output-format* (`lemma|matrix|eval`):
   `Lemma` formats the an output analysis like a Prolog lemma entry for the Verbmobil lemma lexicon (Section 6.1).
   `Matrix` shows an analysis as (flat) attribute-value pairs.
   `Eval` formats an analysis as one database line per input word, containing those types of lexical information that were relevant in the Verbmobil project, see Section 6.6 and Appendix F.
4. *Case-sensitive* (`on|off`:) When analysing a word form with an initial capital letter, the variable `SCAT` is instantiated to `'n'` (noun), which then functions as a top-down constraint for all analyses of this word form.[45]
5. *Analyse-subwords* (`on|off`): In word forms containing one or more hyphens (like `Vier-Tages-Reise`), analyses for the subparts are given in addition to the analyses of the whole word.
6. *Include-POS* (`on|off:`) The input format must be a list with two columns, the first one containing word forms in VM-orthography, the second one Part-of-speech (POS) tags for the word. The POS tag is then used by MCLASS as a top-down constraint on the Variable `SCAT` in the analysis.

The two top-level predicates `mclass/9` and `start/9` process a whole wordlist or one single word form, respectively, the latter e.g. for testing or debugging purposes.

The Post-processor, also written in Prolog, deals with

1. the syllabification and narrow transcription of the (archi-)phonemic representation found for the input string by passing it to the components SILLY and MP-RULES (Matthiesen, 1998, and see Section 6.1.1).
2. formatting the analyses according to the parameters asserted during pre-processing.

[45]According to the Verbmobil spelling conventions, only nouns are transcribed with initial capitals.

6.6. Evaluation

Below, we present the results of two test runs with wordlists from the Verbmobil corpora. For the test runs, the `eval` output mode was selected, and an evaluation script then matched the results against the Bielefeld Lexicon Database, which contains numerous types of lexical information for all Verbmobil word forms and lemmata. Until the year 2000, the database has undergone several phases of manual and automatic consistency checks and correction suites, and furthermore, feedback from its application in the Verbmobil speech recognition and speech synthesis systems has continually been included in its updates. Therefore the lexical information in the database has a very high consistency and correctness (cf. Gibbon and Lüngen, 2000).

The MCLASS evaluation script evaluates those types of lexical information that are crucial for the Verbmobil context:

1. `OrthSeg:` Orthography with morphological boundaries, e.g. `Dienst#reis#+e`
2. `PhonSeg:` Phonology with morphological and syllable boundaries, e.g. `di:nst#raI.z#+@`
3. `Phon:` Plain phonology, e.g. `di:nstraIz@`
4. `Lemma:` Citation form, e.g. `Dienstreise`
5. `Suffixclass:` Name of the suffixclass for verbs and nouns, e.g. `Nomen_Famili-e`
6. `Stemclass:` Name of the stemclass for verbs e.g. `SCHIEBEN`

Information types such as the major syntactic category were omitted simply because they were not included in the evaluation script.

In our two test runs, we have varied the input word form list, otherwise MCLASS was run with the following settings: Nimeton=`on`, Number-of-hypotheses=`5`, Output-format=`eval`, Case-sensitive=`on`, Analyse-subwords=`off`, Include-POS=`off`. The tests have been run using the complete morph lexicon and the complete morphotactics. The input word lists are characterised as follows:

VMCD15: 1379 words, comprising all the word forms types found on the Verbmobil CDROM 15. CDROM 15 was published in late 1996 and is one of the first to contain dialogues in the extended scenario of appointment scheduling and travel planning. Its material had been included in the acquisition of the MCLASS morph database.

VMCD59: 584 words, comprising all the word forms types found on the Verbmobil CDROM 59. CDROM 59 was published in September 2000, and was one of the last Verbmobil CDROMs to be delivered. Its material had *not* been explicitly included in the acquisition of the MCLASS morph database.

Evaluation results for VMCD15

```
========================================================
OrthSeg Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd15.eval.hyp5.nim2.out.rfm
========================================================

total               1379 100 %
correct             1270 92.0957  %
no_match/no_result  109 7.90428 %


========================================================
PhonSeg Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd15.eval.hyp5.nim2.out.rfm
========================================================

total               1379 100 %
correct             1167 84.6265  %
no_match/no_result  212 15.3735 %


========================================================
Phon Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd15.eval.hyp5.nim2.out.rfm
========================================================

total               1379 100 %
correct             1236 89.6302  %
no_match/no_result  143 10.3698 %


========================================================
Lemma Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd15.eval.hyp5.nim2.out.rfm
========================================================

total               1379 100 %
correct             1278 92.6759  %
no_match/no_result  101 7.32415 %


========================================================
Suffixclass Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd15.eval.hyp5.nim2.out.rfm
========================================================

total               1379 100 %
correct             1245 90.2828  %
no_match/no_result  134 9.71719 %


========================================================
Stemclass Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd15.eval.hyp5.nim2.out.rfm
========================================================

total               1379 100 %
correct             1346 97.607  %
no_match/no_result  33 2.39304 %
```

Evaluation results for VMCD59

```
========================================================
OrthSeg Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd59.eval.hyp5.nim2.out.rfm
========================================================

total              583 100 %
correct            548 93.9966  %
no_match/no_result  35 6.00343 %


========================================================
PhonSeg Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd59.eval.hyp5.nim2.out.rfm
========================================================

total              583 100 %
correct            507 86.964  %
no_match/no_result  76 13.036 %


========================================================
Phon Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd59.eval.hyp5.nim2.out.rfm
========================================================

total              583 100 %
correct            523 89.7084  %
no_match/no_result  60 10.2916 %


========================================================
Lemma Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd59.eval.hyp5.nim2.out.rfm
========================================================

total              583 100 %
correct            529 90.7376  %
no_match/no_result  54 9.26244 %


========================================================
Suffixclass Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd59.eval.hyp5.nim2.out.rfm
========================================================

total              583 100 %
correct            535 91.7667  %
no_match/no_result  48 8.23328 %


========================================================
Stemclass Evaluation of mclass output file
MCLASS2_OUTPUT_FILES/cd59.eval.hyp5.nim2.out.rfm
========================================================

total              583 100 %
correct            566 97.084  %
no_match/no_result  17 2.91595 %
```

The comparison between VMCD59 and VMCD15 shows that the domain of appointment scheduling and travel planning seems to be well-covered by the morph database, as the VMCD59 results are not significantly worse than the VMCD15 results, although the former was a CDROM previously "unseen" by MCLASS. The good results for orthographic segmentation indicate that the right rules (morphological construction types) are found and applied. The stemclass and suffixclass results in particular confirm that the MHFP of our theory is well-motivated and crucial for the classification task. In sum, MCLASS works satisfactorily as a grapheme-to-phoneme converter, morphological segmentation tool, lemmatiser, and a morphological classifier for vocabulary from the domain of appointment scheduling and travel planning.

Note also that we have evaluated the boundary symbol and phoneme transcriptions on the word level only, i.e. we have not applied a string distance measure for an evaluation in terms of phoneme accuracy (cf. Gibbon et al., 1997, p.381ff). That is, if a word contained only one phoneme or boundary insertion, omission, or substitution, this was counted as a `no_match` for the entire word. Thus we would expect phoneme accuracy rates to be higher than the transcription rates by word given above.

An error analysis revealed that many of the missing or wrong analyses are due to the following shortcomings:

- The sequence `@.r` has been changed to `6.r` in the lexical database, for example, MP-RULES yields `?E6.+?I.n@.r#+@`, which does not match with `?E6.+?I.n6.r#+@` in the lexical database.
- MCLASS transcribes an orthographic `und` in numbers regularly as `?Und` i.e. including a glottal stop, whereas in the lexicon database this has been consistently changed to a transcription without glottal stop, e.g. `fYnf#Unt#tsvan.+tsIC`. This leads to several `Phon` and `PhonSeg` mitsmatches. (They are some more of such (systematic) inconsistencies between transcriptions in the morph database and the lexical database, which affect the phonological transcription results.)
- We have no morphotactic or morphophonological rule for `t`-insertion before the suffix `-lich` after an unstressed syllable ending in `-n`, as in the word forms `morgen+t+lich, eigen+t+lich, öffen+t+lich`. Therefore such word forms are regularly rejected by MCLASS.
- For the infinitive suffix `-en` of nominalised infinitives, MCLASS puts out the inflectional boundary `#+`, whereas in the lexicon database, it is segmented by the derivational boundary `+` e.g. `Flieg#+en` vs. `Flieg+en`. This leads to several mismatches of `OrthSeg` and `PhonSeg` values.
- Nominalised verbs and adjectives have no suffixclass specification in the lexicon database,[46] whereas MCLASS marks them (correctly, cf. Bleiching and Gib-

[46](because they were originally generated from the respective verb and adjective lemma entries by the Bielefeld paradigm generator).

bon, 2000) as `Nomen_Treffen` and `Nomen_NA`, respectively. This leads to some `Suffixclass` mismatches.

- The implementation of the look-up of the correct stem for the citation form of an input word form using the suffixclass information contains a bug which concerns some rare verbal and nominal suffixclasses. This affected the lemmatisation rate somewhat.

7. Summary and Perspectives

7.1. Summary

In this thesis, we have presented a sign-based theory of the morphology of German. Our morphological objects are lexical signs, comprising signs of the subtypes *morpheme, lemma, lexComplex* and *word.* All signs under *morpheme* are lexicalised, but *lemma, lexComplex*, and *word* subsume both lexicalised and non-lexicalised signs.

Our first focus was on headedness and the nature of a possible morphological Head Feature Principle. Under the assumption that headedness must be determined on the grounds of uniform feature percolation, those HEAD features that play a role in morphology were inventorised. We have distinguished between syntactic HEAD features, which are the morphosyntactic features familiar from several syntactic HPSG theories, and morphological HEAD features, which are relevant in morphotactics, namely in morphological subcategorisation. But both syntactic and morphological HEAD features play a role in morphology in that they are subject to the Morphological Head Feature Principle (MHFP). The MHFP is effective in derivational and compounding structures (subsumed under the type *baseComplex* in our theory), which are right-headed in German. In inflection, though, there is no percolation of morphological HEAD features, and the percolation of syntactic HEAD features is more complex than the sharing of features by a mother constituent and a single HEAD daughter. Moreover, the examination of previously presented principles of headedness in morphology revealed that for inflection, they generally divide HEAD feature percolation into default and exceptional cases, i.e. do not formulate a uniform percolation principle. As an alternative, we have put forward the Inflectional Marking Principle (IMP) for the way the morphosyntactic features are combined in inflectional constructions, which are subsumed under the type *inflected.*

Derivational as well as inflectional affixes select bases on account of their morphological or morphosyntactic properties such as major class (i.e. part of speech), nativeness and inflectional class. This could be summarised in terms of a Morphological Subcategorisation Principle (MSP), which consequently applies to constructions of the type *affixed.*

We have presented three Continuation Schemata for the linearisation of morphs in a word form. In these, it is established that prefixes are actually pre-fixed, and suffixes suffixed, and that compounds and derivatives are right-headed. Otherwise, the schemata remained somewhat preliminary as we could not specify our idea of the interface to a two-level rule component further.

The remaining part of our theory deals with more specific morphotactic problems, such as the role of ablaut in inflection, and the role of umlaut in derivation and inflection. We have furthermore provided an account of how paradigms are not independent lexical objects, but still arise from our organisation of the type hierarchies.

Although we have in fact implemented solutions for the selection of linking morphemes in compounding, of interfixes in derivation, and the role of ablaut and conversion in derivation, we have not presented the HPSG description for them in this thesis.

In general, our approach to morphology relies much on the notion that lexical types are induced by feature values. For example, we would assume that the possible values types of SYN|LOC|HEAD in Pollard and Sag (1994) (e.g. *noun, verb, adjective*) partition *lexical-sign* automatically partitioned into objects like *nominal-sign*, *verbal-sign*, *adjectival-sign*, and so forth. Correspondingly, in our morphology, the different affix types that appear as values of the HEAD-DTR and NONHEAD-DTR features of complex lexical signs induce construction types like *basePrefixed* or *baseSuffixed.* Thus the hierarchy of lexical signs is often a duplication of the hierarchies for values types of certain features of lexical signs. We found it however desirable to make explicit the *lexComplex* hierarchy for our implementation. The *lemma* hierarchy, induced by the MORPH|HEAD|STEMCLASS and MORPH|HEAD|SUFFIXCLASS corresponds to a paradigm class hierarchy, and the *word* hierarchy, induced by the feature SYN|LOCAL|HEAD, is in analogy to the lexical hierarchy presented in (Pollard and Sag, 1987, p.202).

7.2. Perspectives for Mclass

Our implementation strategy for the HPSG morphology can be summarised as follows:

1. The feature structure descriptions of the maximal types at the bottom of the *lexComplex* hierarchy are manually encoded as DCG rules.
2. The morph entries at the bottom of the *morpheme* hierarchy are stored as morph database entries, acquired and type-checked using the interactive classification tool IAMW.
3. DCG and morph database are converted into a BUP parser by means of a Perl program

The inheritance of feature-value specifications and principles from types higher up in the *lexComplex, word*, and *lemma* hierarchies was not implemented, but those in the *morpheme* hierarchy were implemented in the form of the feature co-occurrence restrictions used by IAMW. This means that unfortunately those parts of the theory that lack an implementation could not be operationally evaluated. One could argue that a publicly available tool for constraint-based grammar development, such as Stefan Müller's Babel (`http://www.dfki.de/~stefan/Babel/e_babel.html`), Anne Copestake's LKB (`http://www-csli.stanford.edu/~aac/lkb.html`), or Gerald Penn's ALE (`http://www.cs.toronto.edu/~gpenn/ale.html`) should have been used for implementing the theory. Unfortunately, these were not taken into consideration at the beginning of the project, and besides do not seem to be straightforwardly adaptable to spoken language, i.e. employing pairs of orthographic and phonological representation. But at least our inheritance hierarchies might be implemented and tested for consistency in one of these formalisms, and DCG or BUP code could still be generated from these representations. Moreover, additional constraints such as semantic features could then be integrated more easily.

Our implementation of MCLASS and the software evaluation presented were also quite constricted by the requirements of the lexical acquisition application in the Verbmobil project. Some additional features for MCLASS seem to be desirable. For one thing, one would like to see fully-fledged attribute-value matrices as its output, which is in principle feasible but has not been fully implemented. It would also be an interesting task to implement a morphological generator based on the theory presented.

Moreover, the performance of MCLASS in terms of speed needs improvement. The present implementation could manage the regular Verbmobil corpus extension batches (usually 500-2000 new word forms) in reasonable time, but is simply not suitable for large vocabularies. This is mainly due to two factors: First, the pre- and post-processor cause the performance to be very slow, as they were coded in a more ad-hoc fashion and not guided by theoretical considerations as was the parser. Thus, they should be reimplemented. Second, the interaction of MCLASS and the component NIMETON delays an analysis considerably since in different parsing states, the same string is tested for being a grapho-phonological root. Therefore, a bookkeeping device for storing intermediate results from NIMETON should be implemented. We believe that the method used for the dictionary look-up and storage of so-called inflectional analyses results in the original BUP parser (Matsumoto and Kiyono, 1985) is perfectly applicable to the interface between MCLASS and NIMETON.

Finally, for more efficiency, it would be also desirable to automatically convert our morphology into a finite-state transducer, at least for purposes such as morphological segmentation, grapheme-phoneme conversion, morphosyntactic analysis, and lemmatisation. This is of course not a priori possible for a context-free grammar, but one direction of computational linguistics research deals with finite-state approximations of context-free grammars or even feature grammars, cf. Pereira and Wright (1991), Rood (1996), and Nederhof (2000). We expect that it is feasible to approximate our morphology to an FST, because proper center-embedding rules are fairly restricted, namely to compounding construction types and to those derivational construction types where one daughter is of the type *lkernel* or *rkernel*. Furthermore, the maximum number of actual embeddings in the morphological structure of any German word found in a corpus will be quite limited. In sum, we argue that a lexicographer should ideally have a tool like the HPSG-formalism at their disposal to be able to describe the recursive constituent structures of morphology, while for certain kinds of morphological processing the performance could be improved by an automatic finite-state approximation of the morphological grammar.

Two test runs of Mclass with word lists from the Verbmobil corpus yielded satisfactory results for the types of lexical information that were evaluated. We also pointed out several improvements that could immediately be made to reduce the error rates. But many more kinds of evaluations and differently parametrised test runs could have been carried out. It would be most interesting to run Mclass on corpora from domains other than the appointment scheduling and travel planning dialogues. Such word lists were however not at our disposal in the required Verbmobil orthography, but we expect the results to be worse, because the morph database will not cover all the material. Thus, the morph database should be completed on account of such test runs.

Practice has shown that most of the words from appointment scheduling and travel planning dialogues that are misanalysed by Mclass are all kinds of names as well as words from other languages that were nevertheless used in German speech. As for names, it has been demonstrated in various studies that a morphological or pseudo-morphological description of the structure of names is beneficial for the grapheme-to-phoneme conversion of names, cf. e.g. Belhoula (1993) on German names and Gustafson (1995) on Swedish names (the structures of which are similar to those of German names). We have created an experimental version of Mclass that is able to transcribe German place, street and family names with surprisingly good results by making some simple changes to the morphotactics and the grapho-phonotactic network used by Nimeton, based on an inspection of faulty name transcriptions in the output of Mclass. It would thus be a possible next step to create a morphology that can deal with both names and non-names at the same time.

8. References

References

Althoff, F. (1997). MEWES: Ein Modul für den Einsatz morphologischen Wissens bei der Erkennung gesprochener Sprache. Master's thesis, Universität Bielefeld.

Althoff, F., G. Drexel, H. Lüngen, M. Pampel, and C. Schillo (1996). The treatment of compounds in a morphological component for speech recognition. In D. Gibbon (Ed.), *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, Berlin. Mouton de Gruyter.

Anderson, S. (1992). *A-morphous morphology.* Cambridge, U.K.: Cambridge University Press.

Anderson, S. R. (1982). Where's morphology. *Linguistic Inquiry 13*, 571–613.

Antworth, E. (1994). Morphological parsing with a unification-based word grammar. In *Paper presented at the North Texas Natural Language Processing Workshop*, Arlington, Texas.

Arnold, D. (2001). Bottom up parsing: Left corner algorithm. http://www.essex.ac.uk/course/LG511/7-LeftC/index.pdf.

Aronoff, M. (1986). *Word Formation in Generative Grammar.* Cambridge, Mass.: MIT Press.

Aronoff, M. (1994). *Morphology by itself.* Cambridge, Mass.: MIT Press.

Bauer, L. (1983). *English Word Formation.* Cambridge, U.K.: Cambridge University Press.

Belhoula, K. (1993). Rule-based grapheme-to-phoneme conversion of names. In *Proceedings of EUROSPEECH*, pp. 881–884.

Berton, A., P. Fetter, and P. Regel-Brietzmann (1996). Compound words in large-vocabulary German speech recognition systems. In *Proceedings of the ICSLP-96*, pp. 1165–1168.

Bird, S. and E. Klein (1994). Phonological analysis in typed feature systems. *Computational Linguistics 20*(3), 455–491.

Bleiching, D. (1991). Default-Hierarchien in der deutschen Wortbetonung. ASL-TR-19-91/UBI. Universität Bielefeld.

Bleiching, D. (1992). Prosodisches Wissen im Lexikon. In G. Görz (Ed.), *Proceedings of KONVENS 92*, pp. 59–68. Springer.

Bleiching, D. (1994). Integration von Morphophonologie und Prosodie in ein hierarchisches Lexikon. In *Proceedings of KONVENS 94*, Vienna, pp. 32–41.

Bleiching, D., G. Drexel, and D. Gibbon (1996). Ein Synkretismusmodell für die deutsche Morphologie. In D. Gibbon (Ed.), *Natural Language Processing and Speech Technology. Results of the 3rd KONVENS Conference*, Berlin, pp. 237–248. Mouton de Gruyter.

Bleiching, D. and D. Gibbon (2000). Morphological classes, attributes and values in the Bielefeld lexicon. Verbmobil Technisches Dokument 77. Universität Bielefeld.

Bloomfield, L. (1933). *Language*. New York: Holt.

Brachman, R. and J. Schmolze (1985). An overview of the KL-ONE knowledge representation system. In *Cognitive Science*, Volume 9, pp. 171–216.

Brehmer, K. E. (1985). *German Verbal Prefixes and Modern Generative Theories of Word Structure*. Ph. D. thesis, University of Princeton.

Bresnan, J. (Ed.) (1982). *The Mental Representation of Grammatical Relations*, Cambridge, Mass. MIT Press.

Briscoe, E. and A. Copestake (1999). Lexical rules in constraint-based grammar. *Computational Linguistics 25*(4), 487–526.

Burger, S. (1997). Transliteration spontansprachlicher Daten. Verbmobil Technisches Dokument 56. Universität München.

Cahill, L. and G. Gazdar (1997). The inflectional phonology of German adjectives, determiners and pronouns. *Linguistics 35*(2), 211–245.

Cahill, L. and G. Gazdar (1999). German noun inflection. *Journal of Linguistics 35*, 1–42.

Calcagno, M. (1995). Interpreting lexical rules. In *Proceedings of the Conference on Formal Grammar*, Barcelona. Universidad Politecnica de Catalunya.

Calder, J. (1989). Paradigmatic morphology. In *Proceedings of the 4th Conference of the European Chapter of the Associatio for Computational Linguistics (ACL)*, Manchester, pp. 58–65.

Carden, G. (1983). The non-finite-state-ness of the word formation component. *Linguistic Inquiry 14*, 537–541.

Carpenter, B. (1992). *The Logic of Typed Feature Structures.* Cambridge,U.K.: Cambridge University Press.

Chomsky, N. (1957). *Syntactic Structures.* The Hague: Mouton.

Chomsky, N. (1964). *Current Issues in Linguistic Theory*, Volume 38 of *Janua Linguarum. Series Minor.* The Hague: Mouton. 4th Printing 1969.

Chomsky, N. (1965a). *Aspects of the theory of syntax.* Cambridge, Mass.: MIT.

Chomsky, N. (1965b). On certain formal properties of grammars. In R. D. Luce, R. R. Bush, and E. Galanter (Eds.), *Readings in mathematical psychology*, pp. 323–418. New York: John Wiley and Sons. First published in 1959.

Chomsky, N. (1970). Remarks on nominalization. In R. Jacobs and P. Rosenbaum (Eds.), *Readings in English Transformational Grammar.* Waltham, MA: Blaisdell.

Chomsky, N. (1981). *Lectures on Government and Binding.* Dordrecht: Foris.

Chomsky, N. and M. Halle (1968). *The Sound Pattern of English.* New York: Harper and Row.

Copestake, A. (1999). The (new) LKB system. CSLI Stanford University.

Copestake, A. and T. Briscoe (1992). Lexical operations in a unification based framework. In J. Pustejovsky and S. Bergler (Eds.), *Lexical Semantics and Knowledge Representation*, Berlin, pp. 101–119. Springer.

Copestake, A. and T. Briscoe (1996). Controlling the application of lexical rules. In *Proceedings of the ACL-SIGLEX Workshop on Breadth and Depth of Semantic Lexicons*, Santa Cruz, CA, pp. 7–19. ACL.

Corbett, G. and N. Fraser (1993). Network morphology: A DATR account of russian nominal inflection. *Journal of Linguistics 29*, 113–42.

Şehitoğlu, O. T. and B. H. Cem (1996). Morphological productivity in the lexicon. In *Proceedings of ACL-SIGLEX 1996*, pp. 105–114.

Di Sciullo, A. M. and E. Williams (1987). *On the Definition of Word.* Number 14 in Linguistic Inquiry Monographs. Cambridge, Mass.: MIT Press.

Döpke, J. and J. Walmsley (2000). The proper treatment of derivation in Head-driven Phrase Structure Grammar. In E. e. a. Reitz (Ed.), *Proceedings of the Anglistentag 1999*, Trier. Wissenschaftlicher Verlag.

Ehlebracht, K. (1999). Interaktive Akquisition morphologischer Daten. Internal manual. Universität Bielefeld.

Eisele, A. and J. Dörre (1988). Unification of disjunctive feature structure descriptions. In *Proceedings of the ACL 1988*, pp. 286–294.

Erjavec, T. (1993). Formalising realizational morphology in typed feature structures. In G. Bouma and G. van Noord (Eds.), *Papers from the Fourth CLIN Meeting*, Groningen, pp. 47–58.

Evans, R. and G. Gazdar (1996). DATR: A language for lexical knowledge representation. *Computational Linguistics 22*, 167–216.

Féry, C. (1986). Metrische Phonologie und Wortakzent im Deutschen. In D. Wunderlich (Ed.), *Studium Linguistik*, pp. 16–43. Meisenheim: Hain.

Fischer, K. (1993). Kompositionelle Semantik im Lexikon am Beispiel der englischen Nominalkomposita. Master's thesis, Universität Bielefeld.

Fleischer, W. and I. Barz (1992). *Wortbildung der deutschen Gegenwartssprache.* Tübingen: Niemeyer. 2nd edition 1995.

Flickinger, D. (1987). *Lexical Rules in the Hierarchical Lexicon.* Ph. D. thesis, Stanford University.

Gazdar, G., E. Klein, G. Pullum, and I. Sag (1985). *Generalized Phrase Structure Grammar.* Cambridge, Mass.: Harvard University Press.

Geutner, P. (1995). Using morphology towards better large-vocabulary speech recognition systems. In *Proceedings of the ICASSP-95*, pp. 445–448.

Gibbon, D. (1991). ILEX: A linguistic approach to computational lexica. *Computatio Linguae. Zeitschrift für Dialektologie und Linguistik Beiheft 73*, 32–53.

Gibbon, D. (1995). Verbmobil lexicon: Conventions for spelling and pronunciation. Verbmobil Technisches Dokument 31. Universität Bielefeld.

Gibbon, D. (1997). Compositionality in the inheritance lexicon: English nouns. Synthesis of a talk held at DGfS 1995 annual Meeting. Universität Bielefeld.

Gibbon, D. (2000). Computational lexicography. In F. van Eynde and D. Gibbon (Eds.), *Lexicon Development for speech and Language Processing*, Dordrecht, The Netherlands, pp. 1–42. Kluwer Academic Publishers.

Gibbon, D. and U. Ehrlich (1995). Spezifikation für ein Verbmobil-Lexikondatenbankkonzept. Verbmobil Memo 69. Universität Bielefeld, Daimler Benz AG.

Gibbon, D. and H. Lüngen (2000). Speech lexica and consistent multilingual vocabularies. In W. Wahlster (Ed.), *Verbmobil: Foundations of Speech-to-Speech Translation*, Berlin, pp. 296–307. Springer.

Gibbon, D., R. Moore, and R. Winski (Eds.) (1997). *Handbook of Standards and Resources for Spoken Language Systems.* Berlin: Mouton de Gruyter.

Gibbon, D. and S. Reinhard (1991). Prosodic inheritance and morphological generalisations. In *Proceedings of the 5th Annual Meeting of the European Association for Computational Linguistics*, Berlin.

Gibbon, D. and D. Steinbrecher (1995). Verbmobil-Standardfilter für Transliterationen Version 2.2. Verbmobil Technisches Dokument 38. Universität Bielefeld.

Ginzburg, J. and I. A. Sag (1999). English interrogative constructions. Stanford: CSLI Publications.

Gustafson, J. (1995). Using Two-level morphology to transcribe Swedish names. In *Proceedings of EUROSPEECH*, Madrid.

Halle, M. (1973). Prolegomena to a theory of word formation. *Linguistic Inquiry 4*, 13–16.

Halliday, M. (1985). *An Introduction to Functional Grammar.* London: Arnold.

Harris, Z. (1939). Yokuts structure and Newman's grammar. *International Journal of American Linguistics 10*, 169–80.

Hockett, C. (1954). Two models of grammatical description. *Word 10*, 210–231.

Hoeppner, W. (1980). *Derivative Wortbildung der deutschen Gegenwartssprache und ihre algorithmische Analyse.* Tübingen: Narr.

Hopcroft, J. E. and J. D. Ullman (1979). *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley series in computer science. Reading, Mass.: Addison-Wesley.

Hudson, R. A. (1987). Zwicky on heads. *Linguistics 23*, 109–132.

Jackendoff, R. (1977). *X-Bar Syntax. A Study of Phrase Structure.* Number 2 in Linguistic Inquiry Monographs. Cambridge, Mass.: MIT Press.

Jackendoff, R. (1990). *Semantic Structures.* Cambridge, Mass.: MIT Press.

Jespersen, O. (1933). *Essentials of English Grammar.* London: Allen and Unwin.

Johnson, C. D. (1972). *Formal aspects of phonological description.* The Hague: Mouton.

Karttunen, L. (1983). Kimmo: A general morphological processor. *Texas Linguistic Forum 22*, 165–186.

Karttunen, L. (1984). Features and values. In *Proceedings of COLING 84*, pp. 28–33.

Karttunen, L. and K. R. Beesley (1992). *Two-level rule compiler*, Volume ISTL-92-2 of *Technical Report*. Palo Alto, CA.: Xerox Palo Alto Research Center and CSLI.

Kasper, R. (1987). A unification method for disjunctive descriptions. In *Proceedings of the ACL 1987*, pp. 235–242.

Kasper, R. and W. Rounds (1986). A logical semantics for feature structures. In *Proceedings of ACL 1986*, pp. 257–266.

Kathol, A. (1999). Agreement and the syntax-morphology interface in HPSG. In R. Levine and G. Green (Eds.), *Studies in Current Phrase Structure Grammar*, pp. 223–274. Cambridge, U.K.: Cambridge University Press.

Kay, M. (1979). Functional unification grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, University of California, Berkeley, CA., pp. 142–158.

Keller, B. (1993). *Feature Logics, Infinitary Descriptions and Grammar.* CSLI Lecture Notes. Stanford, CA.: CSLI.

King, P. J. (1989). *A logical formalism for Head-driven Phrase Structure Grammar.* Ph. D. thesis, Manchester University, Manchester, U.K.

Kiparsky, P. (1982a). From cyclic phonology to lexical phonology. In H. van der Hulst and N. Smith (Eds.), *The Structure of Phonological Representation I.* Dordrecht: Foris.

Kiparsky, P. (1982b). Lexical morphology and phonology. In *Linguistics in the morning calm.* Seoul: Hanshin.

Kiparsky, P. (1985). Some consequences of lexical phonology. In *Phonology Yearbook 19*, Volume 2, pp. 83–136.

Kiss, T. (1995). *Merkmale und Repräsentationen. Eine Einführung in die deklarative Grammatikanalyse.* Opladen: Westdeutscher Verlag.

Koenig, J.-P. (1999). *Lexical Relations.* Stanford Monographs. Stanford University, C.A.: CSLI Publications.

Koenig, J.-P. and D. Jurafsky (1994). Type underspecification and on-line type construction in the lexicon. In *Proceedings of the 13th West Coast Conference on Formal Linguistics*, CSLI, Stanford CA, pp. 270–285.

Koskenniemi, K. (1983a). Two-level-model for morphological analysis. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI*, Karlsruhe, pp. 683–685.

Koskenniemi, K. (1983b). *Two-Level Morphology. A General Computational Model for Word-Form Recognition and Production.* Ph. D. thesis, University of Helsinki.

Koskenniemi, K. (1985). Compilation of automata from morphological two-level rules. In *Papers from the 5th Scandinavian Conference of Computational Linguistics*, University of Helsinki, pp. 143–149.

Koskenniemi, K. and K. Church (1988). Two-level Morphology and Finnish. In *Proceedings of COLING-88*, Budapest, pp. 335–339.

Krieger, H.-U. (1993). Derivation without lexical rules. DFKI Research Report RR-93-27. DFKI Saarbrücken.

Krieger, H.-U., H. Pirker, and J. Nerbonne (1993). Feature-based allomorphy. In *Proceedings of 31st Annual Meeting of the Association for Computational Linguistics*, pp. 140–147.

Langer, S. (1998). Zur Semantik von Nominalkomposita. In B. Schröder, W. Lenders, W. Hess, and T. Portele (Eds.), *Computer Studies in Language and Speech. Proceedings of the 4th Conference on Natural Language Processing - KONVENS-98*, Frankfurt, pp. 83–97.

Lüdeling, A. (1999). *On Particle Verbs and Similar Constructions.* Ph. D. thesis, Universität Stuttgart.

Lehmann, C. (1990). Morphologische Kategorien. Internal Report. Universität Bielefeld.

Lieber, R. (1992). *Deconstructing Morphology. Word Formation in Syntactic Theory.* Chicago: University of Chicago Press.

Lüngen, H., K. Ehlebracht, D. Gibbon, and A. P. Quirino Simões (1998). Bielefelder Lexikon und Morphologie in Verbmobil Phase II. Verbmobil Report 233. Universität Bielefeld.

Lüngen, H., M. Pampel, G. Drexel, D. Gibbon, F. Althoff, and C. Schillo (1996). Morphology and speech technology. In *Proceedings of the 2nd ACL-SIGPHON Conference*, University of California, Santa Cruz. Association for Computational Linguistics.

Lüngen, H. and C. Sporleder (1999). Automatic induction of lexical inheritance hierarchies. In J. Gippert (Ed.), *Multilinguale Corpora: Kodierung, Strukturierung, Analyse*, Prague, pp. 42–52. Enigma Corporation.

Matsumoto, Y. and M. Kiyono (1985). Facilities of the BUP parsing system. In V. Dahl and P. Sain-Dizier (Eds.), *Natural Language Understanding and Logic Programming*, North Holland, pp. 97–106. Elsevier Science Publishers B.V.

Matsumoto, Y., H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa (1983). BUP: A Bottom-up parser embedded in Prolog. *New Generation Computing 1*(2), 145–158.

Matthews, P. (1974). *Morphology. An Introduction to the Theory of Word-Structure.* Cambridge: Cambridge University Press.

Matthiesen, M. (1996). Nimeton - Ein Graphem-Phonem-Übersetzer für das Deutsche. Verbmobil Memo 109. Universität Bielefeld.

Matthiesen, M. (1998). SILLY - Silbifizierung mittels morphologischer Informationen. Universität Bielefeld. Verbmobil Memo 137.

Matthiesen, M. (1999). Morphologie im Textmining. Master's thesis, Universität Bielefeld.

Mel'cuk, I. A. (1988). *Dependency Syntax: Theory and Practice.* New York: State University of New York Press.

Mengel, A. (1999). A phonetic morpheme lexicon for German. In *Proceedings of the ICPhS*, San Francisco CA.

Müller, S. (1999). *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche.* Number 394 in Linguistische Arbeiten. Tübingen: Max Niemeyer Verlag.

Müller, S. (2000). HPSG Analysis of German. In W. Wahlster (Ed.), *Verbmobil: Foundations of Speech-to-Speech Translation*, Berlin, pp. 238–253. Springer.

Mohanan, K. P. (1982). *Lexical Phonology.* Ph. D. thesis, University of Texas.

Müller, S. (2000). *Complex Predicates: Verbal Complexes, Resultative Constructions, and Particle Verbs in German.* Habilitationsschrift, Universität des Saarlandes, Saarbrücken.

Naumann, S. and H. Langer (1994). *Parsing. Eine Einführung in die maschinelle Analyse natürlicher Sprache.* Leitfäden und Monographien der Informatik. Stuttgart: Teubner.

Nederhof, M.-J. (2000). Practical experiments with regular approximation of context-free languages. *Computational Linguistics 26*(1), 17–44.

Netter, K. (1994). Towards a theory of functional heads: German nominal phrases. In J. Nerbonne, K. Netter, and C. Pollard (Eds.), *German in Head-Driven Phrase Structure Grammar*, Number 46 in CSLI Lecture Notes, Stanford, CA, pp. 297–340.

Pampel, M. (1991). Die Repräsentation lexikalischen phonologischen Wissens am Beispiel der Wortbetonung. Master's thesis, Universität Bielefeld.

Pampel, M. (1999). *Morphologische Wortmodellierung und automatische Spracherkennung.* Ph. D. thesis, Universität Bielefeld.

Pereira, F. and D. Warren (1980). Definite Clause Grammars for natural language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence 13*, 231–278.

Pereira, F. and R. Wright (1991). Finite-state approximation of phrase-structure grammars. In *Proceedings of the 29th ACL Conference*, pp. 246–255.

Pollard, C. and I. Sag (1987). *Information-Based Syntax and Semantics.* Menlo Park, CA: CSLI International.

Pollard, C. and I. Sag (1994). *Head-Driven Phrase Structure Grammar.* Chicago: University of Chicago Press.

Reinhard, S. (1991). Adäquatheitsprobleme automatenbasierter Morphologiemodelle am Beispiel der deutschen Umlautung. Master's thesis, Universität Trier.

Riehemann, S. (1993). Word formation in lexical type hierarchies. A case study of *bar*-adjectives in German. Master's thesis, Universät Tübingen.

Riehemann, S. Z. (1998). Type-based derivational morphology. *Journal of Comparative Germanic Linguistics 2*, 49–77.

Riehemann, S. Z. (2001). *A constructional approach to idioms and word formation.* Ph. D. thesis, Stanford University, Palo Alto, CA.

Ritchie, G. D., G. J. Russel, A. W. Black, and S. G. Pulman (1992). *Computational Morphology.* London: MIT Press.

Robins, R. (1959). In defence of WP. In *Diversions of Bloomsbury: Selected Writings on Linguistics*, North Holland Linguistic Series, Chapter 4, pp. 116–144. Amsterdam: North Holland Publishing Company.

Rood, C. (1996). Efficient finite-state approximation of context-free grammars. *Proceedings of the ECAI Workshop on Extended Finite State Models of Language 22*, 58–64. Budapest 1996.

Sag, I. A. (1997). English relative clause constructions. *Journal of Linguistics 33*, 431–484.

Sag, I. A. and T. Wasow (1999). *Syntactic Theory. A Formal Introduction.* Stanford, CA: CSLI.

Schöter, A. (1993). Compiling feature structures into terms: An empirical study in Prolog. Research Paper EUCCS-RP-1993-1.

Selkirk, E. O. (1982). *The Syntax of Words.* Cambridge, Mass.: MIT Press.

Shieber, S. M. (1986). *An introduction to unification-based approaches to grammar*, Volume 4 of *CSLI Lecture Notes.* Stanford University, CA: CSLI.

Siegel, D. (1979). *Topics in English Morphology.* Outstanding Dissertations in Linguistics. New York: Garland Publ.

Spencer, A. (1991). *Morphological Theory.* Blackwell Textbooks in Linguistics. Oxford: Blackwell.

Sporleder, C. (1999). Learning lexical generalisations. An operational evaluation of current machine learning methods. Master's thesis, Universität Bielefeld.

Sproat, R. (1992). *Morphology and Computation.* Cambridge, Mass.: MIT Press.

Steinbrecher, D. (1995). MSEG: Morphologische Segmentierung deutscher Wortformen. Verbmobil Memo 99. Unversität Bielefeld.

Stowell, T. (1990). Subjects, specifiers, and X-bar theory. In *Alternative Conceptions of Phrase Structure*, pp. 232–262. University of Chicago Press.

Strom, V. and H. Heine (1999). Utilizing prosody for unconstrained morpheme recognition. In *Proceedings of EUROSPEECH 1999*, Budapest.

Stump, G. (1991). A paradigm-based theory of morphosemantic mismatches. *Language 67*, 675–725.

Toman, J. (1987). *Wortsyntax. Eine Diskussion ausgewählter Probleme deutscher Wortbildung.* Tübingen: Niemeyer.

Travis, L. (1990). Parameters of phrase structure. In M. Baltin and A. Kroch (Eds.), *Alternative Conceptions of Phrase Structure.* Chicago: University of Chicago Press.

Trost, H. (1993). Coping with derivation in a morphological component. In *Proceedings of the 6th EACL Conference*, Utrecht, pp. 268–376.

Uszkoreit, H. (1986). *Categorial Unification Grammars.* Number 86-66 in CSLI Reports. Palo Alto, CA.: CSLI, Stanford University.

Uszkoreit, H., R. Backofen, S. Busemann, A. Diagne, E. Hinkelman, W. Kasper, B. Kiefer, H.-U. Krieger, K. Netter, G. Neumann, S. Oepen, and S. Spackmann (1994). DISCO - An HPSG-based NLP system and its application for appointment scheduling. In *Proceedings of COLING-94*, pp. 436–440.

Uszkoreit, H., D. Flickinger, W. Kasper, and I. A. Sag (2000). Deep linguistic analysis with HPSG. In W. Wahlster (Ed.), *Verbmobil: Foundations of Speech-to-Speech Translation*, Berlin, pp. 216–237. Springer.

van Eynde, F. (1994). *Auxiliaries and Verbal Affixes. A Monostratal Cross-Linguistics Analysis.* Ph. D. thesis, Katholieke Universiteit Leuven.

Wahlster, W. (2000a). Mobile speech-to-speech translation of spontaneous dialogs: An overview of the final Verbmobil system. In W. Wahlster (Ed.), *Verbmobil: Foundations of Speech-to-Speech Translation*, pp. 3–21. Berlin: Springer.

Wahlster, W. (Ed.) (2000b). *Verbmobil: Foundations of Speech-to-Speech Translation.* Berlin: Springer.

Wiese, R. (1987). Phonologie und Morphologie des Umlauts im Deutschen. *Zeitschrift für Sprachwissenschaft 6*(2), 227–248.

Williams, E. (1981). On the notions 'lexically related' and 'head of a word'. *Linguistic Inquiry 12*, 245–274.

Witt, A. and S. Müller (2002). Grundlagen für den Computereinsatz in der Linguistik: Attribute, Werte, Unifikation. In H. Müller (Ed.), *Arbeitsbuch Linguistik*, UTB. Paderborn: Schöningh.

Wothke, K. (1993). Morphologically based automatic phonetic transcription. *IBM Systems Journal 32*(3), 31–38.

Zwicky, A. M. (1967). Umlaut and noun plurals in German. *Studia Grammatica 6*, 35–45.

Zwicky, A. M. (1985). Heads. *Linguistics 21*, 1–29.

A. Table of abbreviations

ADJ	Adjective
AP	Adjective phrase
AVM	Attribute-value matrix
CUG	Categorial Unification Grammar
DAG	Directed acyclic graph
DP	Determiner phrase
Det	Determiner
FUG	Functional Unification Grammar
GB	Government and Binding
GPSG	Generalised Phrase Structure Grammar
HFP	Head Feature Principle
HPSG	Head-driven Phrase Structure Grammar
IA	Item-and-arrangement
IAMW	Interaktive Akquisition morphologischer Daten
ILEX	Integrated Lexicon with Exceptions
IMP	Inflectional Marking Principle
IP	Item-and-process
LFG	Lexical Functional Grammar
LHS	Left-hand side
LPM	Lexical Phonology and Morphology
MAWR	Morphological affix-word rule
MHFP	Morphological Head Feature Principle
MSP	Morphological Subcategorisaton Principle
N	Noun
NP	Noun phrase
P	Preposition
PP	Prepositional phrase
PS-grammar	Phrase structure grammar
PSG	Phrase structure grammar
RHR	Right-hand head rule
RHS	Right-hand side
RRHR	Relativised right-hand head rule
SAMPA	Speech assessment methodologies phonetic alphabet
SPE	*The Sound Pattern of English* (Chomsky and Halle, 1968)
TBDM	Type-based derivational morphology (Riehemann 1998)
V	Verb
VP	Verb phrase
WFST	Well-formed substring table
WP	Word-and-paradigm

B. DCG morphotactics

```
% Rule1a              stemPrefixed  -->  part stemPre2fixed
% Example                           -->


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPre2fixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-------------------------------------------------------------------

% Rule1b              stemPrefixed  -->  part  stemPre1fixed
% Example             vorbehalt     -->  vor   behalt


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPre1fixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,
  not(X2_LBND==lbndFree),
% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%%----------------------------------------------------------------------

% Rule1c              stemPrefixed  -->  part stemPre1Zufixed
% Example             anzubehalt    -->  an   zubehalt


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPre1Zufixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,
  X0_FLEX=infinitive_flex,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,
% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
% X2_FLEX=X0_FLEX,                      % wird oben initialisiert.
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%%--------------------------------------------------------------------

% Rule1d             stemPrefixed    -->  pre2  stemPre1Zufixed
% Example            mi"szuversteh   -->  mi"s  zuversteh


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

pre2(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPre1Zufixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,
  X0_FLEX=infinitive_flex,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
% X2_FLEX=X0_FLEX,                          % wird oben initialisiert.
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%%----------------------------------------------------------------------


% Rule2             stemPre1Zufixed -->  infin[zu]  stemPre1fixed
% Example           zuversteh   -->      zu         versteh


% Linear Precedence and Feature Vectors:

stemPre1Zufixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

infin(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPre1fixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPre1Zufixed,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
  X1_MID=zu_infin,
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='-',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%----------------------------------------------------------------------

% Rule3a            stemPrefixed       -->  part  rkernel
% Example           anbrenn            -->  an    brenn
% Example           "uberinterpretier  --> "uber  interpretier


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

rkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%----------------------------------------------------------------------

% Rule3b             stemPrefixed    -->  part  zuRkernel
% Example            anzubrenn       -->  an    zubrenn


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

zuRkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,
  X0_FLEX=infinitive_flex,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
```

```
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
% X2_FLEX=X0_FLEX,                      % wird oben initialisiert.
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%----------------------------------------------------------------------

% Rule3c             stemPrefixed    -->  pre2  zuRkernel
% Example            mi"szutrau      -->  mi"s  zutrau
% Comment            zweifelhaft


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

pre2(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

zuRkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,
  X0_FLEX=infinitive_flex,
% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
% X2_FLEX=X0_FLEX,                       % wird oben initialisiert.
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%-----------------------------------------------------------------------

% Rule3e             zuRkernel  -->  infin[zu]  rkernel
% Example            zuversteh  -->  zu         nehm


% Linear Precedence and Feature Vectors:

zuRkernel(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

infin(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

rkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=zuRkernel,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
  X1_MID=zu_infin,
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='-',
% X1_ADJ='-',
% X1_BCAT=v,

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%----------------------------------------------------------------------------------------------


% Rule3f               stemPrefixed  -->  pre2  stemPre1fixed
% Example              mi"sversteh   -->  mi"s  versteh


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

pre2(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPre1fixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),


% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
% X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%----------------------------------------------------------------------------------------------------

% Rule3g             stemPrefixed --> stemPre2fixed


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemPre2fixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),


% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


%----------------------------------------------------------------------------------------------

% Rule3h               stemPrefixed --> stemPre1fixed

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemPre1fixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------------------------------

% Rule4             stemPre2fixed       -->  pre2  rkernel
% Example           mi"strau            -->  mi"s  trau
% Example           mi"sinformier       -->  mi"s  informier


% Linear Precedence and Feature Vectors:

stemPre2fixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

pre2(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

rkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPre2fixed,
% X1 <-> X2 Equational Constraints
  X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
% X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------------------------------------

% Rule5             stemPre1fixed       -->  pre1  rkernel
% Example           zerquetsch          -->  zer   quetsch
% Example           beleidig      t     -->  be    leidig


% Linear Precedence and Feature Vectors:

stemPre1fixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

pre1(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),
rkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='|+',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPre1fixed,
  X0_PARTSTEM='+',

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,
% X1_NAT=X2_NAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='-',
% X1_ADJ='+',

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%-----------------------------------------------------------------------------------------------


% Rule6            stemPre1fixed --> prenn rkernel
% Example          rekapitulier  --> re    kapitulier


% Linear Precedence and Feature Vectors:
```

```
stemPre1fixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

prenn(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

rkernel(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='|+',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPre1fixed,
  X0_PARTSTEM='-',

% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_SCAT,
% X1_NAT=X2_NAT,

% Constraints on X1 Instantiations
% not(X1_LBND='lbndToBind'),
% X1_RBND=rbndToBind,
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='-',
% X1_ADJ='+',

% Constraints on X2 instantiations
  not(X2_LBND==lbndFree),
  X2_NAT='-',

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%----------------------------------------------------------------------------------------------


% Rule7a        rkernel --> stemSuffixed


% Linear Precedence and Feature Vectors:

rkernel(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemSuffixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),


% X0 Instantiations
  X0_MCAT=rkernel,

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.



% -----------------------------------------------------------------------------------------------------

% Rule7b            rkernel --> root


% Linear Precedence and Feature Vectors:

rkernel(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

root(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=rkernel,

% Constraints on X1 Instantiations:
  not(X1_LBND='lbndFree'),

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.



% -----------------------------------------------------------------------

% Rule8a            stemSuffixed   -->   stemSuffixed  sufn[comb+]
% Example           einheitlich    -->   einheit       lich


% Linear Precedence and Feature Vectors:

stemSuffixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemSuffixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

sufn(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='+',
  BRACKETING=left_bracket,

% X0 Instantions
  X0_MCAT=stemSuffixed,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_BCAT,
% X1_UMGL=X2_UMLD,
% X1_UMLB=X2_UMLD,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),

% Constraints on X2 Instantiations
  X2_COMB_SUF='+',
% X2_NAT='+',

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,
% X1_PARTSTEM=X0_PARTSTEM,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
% X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

% -----------------------------------------------------------------------

% Rule8b        stemSuffixed   -->  lkernel[nat-] sufn[comb+]
% Example       sozialistisch  -->  sozialist     isch


% Linear Precedence and Feature Vectors:

stemSuffixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-
```

```
lkernel(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

sufn(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),


% Boundary Symbol and Bracketing
  BOUNDARY='+',
  BRACKETING=left_bracket,

% X0 Instantions
  X0_MCAT=stemSuffixed,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_BCAT,
  X1_UMGL=X2_UMLD,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),
  X1_NAT='-',

% Constraints on X2 Instantiations
  X2_COMB_SUF='+',
% X2_NAT='+',

% X1 <-> X0 Feature Percolation
  X0_LBND=X1_LBND,
  X0_PARTSTEM=X1_PARTSTEM,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
% X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


% ---------------------------------------------------------------------

% Rule8c        stemSuffixed   -->   stemSuffixed sufnn
% Example       solidarit"at   -->   solidar      it"at
% Linear Precedence and Feature Vectors:

stemSuffixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemSuffixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

sufnn(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='+',
  BRACKETING=left_bracket,

% X0 Instantions
  X0_MCAT=stemSuffixed,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_BCAT,
  X1_UMGL=X2_UMLD,
% X1_NAT=X2_NAT,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),
  X1_NAT='-',
% X1_UMLB=X1_UMGL,  %% dieser constraint besagt, dass
           %% immer umgelautet wird, wenn umlautbar.

% Constraints on X2 Instantiations
% X2_NAT='-',

% X1 <-> X0 Feature Percolation
  X0_LBND=X1_LBND,
  X0_PARTSTEM=X1_PARTSTEM,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
% X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
% X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------------------------------------

% Rule8d              stemSuffixed  -->  lkernel   suf
% Example             neuheit       -->  neu       heit
% Example             interessier   -->  interess  ier
% Example             verarbeitung  -->  verarbeit ung


% Linear Precedence and Feature Vectors:

stemSuffixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

lkernel(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

suf(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='+',
  BRACKETING=left_bracket,

% X0 Instantions
  X0_MCAT=stemSuffixed,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_BCAT,
  X1_NAT=X2_NAT,
% X1_UMGL=X2_UMLD,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),

% Constraints on X2 Instantiations
X2_INTERF_ARG='-',
X2_PRET='-',

% X1 <-> X0 Feature Percolation
  X0_LBND=X1_LBND,
  X0_PARTSTEM=X1_PARTSTEM,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
% X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
% X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------------------------------

% Rule8e        stemSuffixed   -->   stemInterfixed  sufnn
% Example       information    -->   informat        ion


% Linear Precedence and Feature Vectors:

stemSuffixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemInterfixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

sufnn(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='+',
  BRACKETING=left_bracket,

% X0 Instantions
  X0_MCAT=stemSuffixed,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_BCAT,
% X1_NAT=X2_NAT,
% X1_INTERF=X2_INTERF_ARG,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),
  X1_NAT='-',

% Constraints on X2 Instantiations
  X2_NAT='-',
% X1 <-> X0 Feature Percolation
  X0_LBND=X1_LBND,
  X0_PARTSTEM=X1_PARTSTEM,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
% X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
% X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

% ----------------------------------------------------------------------

% Rule8f             stemSuffixed   -->   stemInterfixed  sufn[comb+]
% Example            theoretisch    -->   theoret         isch


% Linear Precedence and Feature Vectors:

stemSuffixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemInterfixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

sufn(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='+',
  BRACKETING=left_bracket,

% X0 Instantions
  X0_MCAT=stemSuffixed,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_BCAT,
% X1_UMGL=X2_UMLD,
% X1_UMLB=X2_UMLD,
% X1_INTERF=X2_INTERF_ARG,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),

% Constraints on X2 Instantiations
  X2_COMB_SUF='+',
% X2_NAT='+',

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,
% X1_PARTSTEM=X0_PARTSTEM,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
% X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%---------------------------------------------------------------------------------------------

% Rule9a              stemInterfixed   -->   lkernel interf
% Example             theroet          -->   theor   et
% Example             chines           -->   chin    es


% Linear Precedence and Feature Vectors:

stemInterfixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

lkernel(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

interf(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
```

```
  BOUNDARY='+',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemInterfixed,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_BCAT,
  X1_NAT=X2_NAT,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),
  not(X1_INTERF=='-'),

% Constraints on X2 Instantiations

% X1 <-> X0 Feature Percolation
  X0_LBND=X1_LBND,
  X0_PARTSTEM=X1_PARTSTEM,


% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
% X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
% X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------------------------

% Rule10              lkernel  -->  root


% Linear Precedence and Feature Vectors:

lkernel(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

root(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),
% X0 Instantiations
X0_MCAT=lkernel,

% Constraints on X1 Instantiations:
not(X1_RBND==rbndFree),

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.

% ----------------------------------------------------------------------------

% Rule10b           lkernel   -->   stemPrefixed


% Linear Precedence and Feature Vectors:

lkernel(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemPrefixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
X0_MCAT=lkernel,

% Constraints on X1 Instantiations:
not(X1_RBND==rbndFree),

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


% ----------------------------------------------------------------------------

% Rule11          stem   -->   stemSuffixed


% Linear Precedence and Feature Vectors:

stem(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemSuffixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
X0_MCAT=stem,

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


% ---------------------------------------------------------------------------
% Rule11b          stem   -->   stemPrefixed


% Linear Precedence and Feature Vectors:

stem(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemPrefixed(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
X0_MCAT=stem,

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.

% ---------------------------------------------------------------------------

% Rule11c          stem   -->   root


% Linear Precedence and Feature Vectors:

stem(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

root(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),
% X0 Instantiations
  X0_MCAT=stem,

% Constraints on X1 Instantiations
% not(X1_LBND==lbndToBind), % Gegenbeispiel: 'erstaunlicherweise': die adv-root '-weise'
                            % ist als lbndToBind klassifiziert.
                            % Vielleicht f"ur -weise u.a. eine extra Regel einf"uhren.

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


%------------------------------------------------------------------------------

% Rule11d          stem   -->   stemCompound


% Linear Precedence and Feature Vectors:

stem(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemCompound(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=stem,

% Constraints on X1 Instantiations
% not(X1_LBND==lbndToBind), % Gegenbeispiel: 'erstaunlicherweise': die adv-root '-weise'
                            % ist als lbndToBind klassifiziert.
                            % Vielleicht f"ur -weise u.a. eine extra Regel einf"uhren.

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


% -------------------------------------------------------------------------

% Rule12a         stemSuffixed[adj]  -->  word[participle]
% Example         gelaufen           -->  gelaufen
% Example         laufend            -->  laufend


% Linear Precedence and Feature Vectors:

stemSuffixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

word(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=stemSuffixed,
  X0_SCAT=a,
  X0_FLEX=adjective_flex,
  X0_PARTSTEM=conversed_participle,
  X0_RBND=rbndOpt,
  X0_LINK=linkOpt,
  X0_LINK_MID=['er_lm_adv'],
  X0_SUFFIXCLASS='ADJECTIVAL',
  X0_FLEX=adjective_flex,


% Constraints on X1 Instantiations
  X1_SCAT==v,
  sublistmember("non-finit-part",X1_FLEX),
% not(X1_LBND==lbndToBind),

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
% X1_RBND=X0_RBND,
% X1_BCAT=X0_BCAT,
% X1_SCAT=X0_SCAT,
% X1_NAT=X0_NAT,
% X1_LINK=X0_LINK,
% X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
% X1_STEMCLASS=X0_STEMCLASS,
% X1_SUFFIXCLASS=X0_SUFFIXCLASS,
% X1_FLEX=X0_FLEX,
% X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


% ------------------------------------------------------------------------------------

% Rule13a        past_stem --> stem[v,PRET=+,REG=-]


% Linear Precedence and Feature Vectors:

past_stem(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations:
  X0_MCAT=past_stem,

% Constraints on X1 Instantiations
% not(X1_LBND==lbndToBind),
% not(X1_RBND==rbndFree),
  X1_SCAT=v,
  X1_PRET='+',
  member(X1_VROOT,['BLIEB','HIE"S','KAM','K"AM','SCHOB','SCH"OB','GING','FIEL',
'FAND','F"AND','SAH','S"AH','FUHR','F"UHR','NAHM','N"AHM','WURD','W"URD','RANN',
'DARF']),
  X1_REG='-',
% X1_STEMCLASS=strong,      % entsprechende repraesentation, aber braucht man nicht, die
                            % information ist schon in der X1_VROOT-Gleichung enthalten.

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_SCAT=X0_SCAT,
  X1_NAT=X0_NAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_INTERF_ARG=X0_INTERF_ARG,
  X1_GEN=X0_GEN,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_FLEX=X0_FLEX,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.

% ---------------------------------------------------------------------------------------------


% Rule13a          past_stem   -->   stem[v,REG+,PRET=-] past
% Example          sagt        -->   sag                 t

% This rule won't be used, as past and person suffixes are
% merged into one inflectional suffix.

% Linear Precedence and Feature Vectors:

past_stem(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

past(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),


% Boundary Symbol and Bracketing
  BOUNDARY='#+',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=past_stem,
  X0_PRET='+',

% X1 <-> X2 Equational Constraints

% Constraints on X1 Instantiations
  not(X1_RBND==rbndFree),
% not(X1_LBND==lbndToBind),
  X1_SCAT=v,
  X1_PRET='-',
  X1_VROOT='FRAG',
  % X1_REG='reg+',
  % member(X1_SUFFIXCLASS,['SAGEN','WARTEN','BEGEISTERN','FASSEN']),

% Constraints on X2 Instantiations
% X1 <-> X0 Feature Percolation
  not(X1_RBND==rbndFree),
  X1_LBND=X0_LBND,
  X1_NAT=X0_NAT,
  X1_SCAT=X0_SCAT,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_STEMCLASS=X0_STEMCLASS,
% X1_INTERF=X0_INTERF,
% X1_GEN=X0_GEN,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS,

% X2 <-> X0 Feature Percolation
  X2_RBND=X0_RBND,
% X2_PRET=X0_PRET,
  X2_FLEX=X0_FLEX.


% ------------------------------------------------------------------------------

% Rule14a          word        -->  stem      inflection
% Example          nimmst      -->  nimm      st
% Example          w"orter     -->  w"ort     er
% Example          sch"oneres  -->  sch"oner  es

% Linear Precedence and Feature Vectors:

word(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

inflection(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#+',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=word,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_SCAT,
  X1_VROOT=X2_VROOT,
  X1_SUFFIXCLASS=X2_SUFFIXCLASS,
  X1_REG=X2_REG,
  X1_PARTSTEM=X2_PARTSTEM,

% Constraints on X1 Instantiations
  not(X1_LBND=='lbndToBind'),
  not(X1_RBND=='rbndFree'),
  X1_PRET='-',

% Constraints on X2 Instantiations
  not(X2_RBND==rbndToBind),

% X1 <-> X0 Feature Percolation
  X1_NAT=X0_NAT,
  X1_SCAT=X0_SCAT,
  X1_LBND=X0_LBND,
  X1_GEN=X0_GEN,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
% X1_PRET=X0_PRET,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_INTERF=X0_INTERF,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS,


% X2 <-> X0 Feature Percolation
  X2_FLEX=X0_FLEX,
  X2_RBND=X0_RBND.

% ------------------------------------------------------------------------------

% Rule14b             word   -->   past_stem   v_infl


% Linear Precedence and Feature Vectors:

word(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

past_stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

v_infl(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#+',
  BRACKETING=left_bracket,
% X0 Instantiations
  X0_MCAT=word,

% X1 <-> X2 Equational Constraints
  X1_SCAT=X2_SCAT,
  X1_SUFFIXCLASS=X2_SUFFIXCLASS,
  X1_VROOT=X2_VROOT,
% the constraints on STEMCLASS and SUFFIXCLASS muessen ueberdacht werden
% FLEX must be unified

% Constraints on X1 Instantiations
  not(X1_LBND=='lbndToBind'),
  X1_PRET='+',

% Constraints on X2 Instantiations
  not(X2_RBND==rbndToBind),

% X1 <-> X0 Feature Percolation
  X1_NAT=X0_NAT,
  X1_GEN=X0_GEN,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_LBND=X0_LBND,
  X1_SCAT=X0_SCAT,
  X1_REG=X0_REG,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_VROOT=X0_VROOT,
  X1_INTERF=X0_INTERF,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS,

% X2 <-> X0 Feature Percolation
  X2_FLEX=X0_FLEX,
  X2_RBND=X0_RBND.


% ---------------------------------------------------------------------------------------

% Rule15a          stemCompound        -->  stem    stem
% Example          Terminverschiebung  -->  termin  verschiebung


% Linear Precedence and Feature Vectors:

stemCompound(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stem(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemCompound,

% X1 <-> X2 Equational Constraints

% Constraints on X1 Instantiations
  not(X1_LINK==linkObl),
  not(X1_RBND=='rbndFree'),

% Constraints on X2 Instantiations
  not(X2_LBND=='lbndFree'),

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


% -------------------------------------------------------------------------------

% Rule15b          stemCompound   -->   stemLm   stem
% Example          arbeitsamt     -->   arbeits  amt


% Linear Precedence and Feature Vectors:

stemCompound(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemLm(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stem(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemCompound,

% X1 <-> X2 Equational Constraints
% X1_SCAT=X2_SCAT,

% Constraints on X1 Instantiations
% not(X1_RBND=='rbndFree'),

% Constraints on X2 Instantiations
  not(X2_LBND=='lbndFree'),

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


% --------------------------------------------------------------------------

% Rule16a                stemLm    -->   stem   lm
% Example                arbeits   -->   arbeit s


% Linear Precedence and Feature Vectors:

stemLm(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

       stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

       lm(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),


% Boundary Symbol and Bracketing
  BOUNDARY='#+',
  BRACKETING=left_bracket,

% X0 Instantiations
% X0_MCAT=stemLm,

% X1 <-> X2 Equational Constraints
% X1_UMGL=X2_UMLD,
% X1_SCAT=X2_BCAT,

% Constraints on X1 Instantiations
  not(X1_LINK==linkImp),
  inst_member(X2_MID, X1_LINK_MID),

% Constraints on X2 Instantiations

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,
  X1_SCAT=X0_SCAT,
% X1_NAT=X0_NAT,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_PRET=X0_PRET,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_GEN=X0_GEN,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS,

% X2 <-> X0 Feature Percolation
  X2_RBND=X0_RBND,
  X2_FLEX=X0_FLEX.

% what about INTERF

% ----------------------------------------------------------------------------

% Rule16b                 stemLm[NA]   -->   stem[adj]  lm[en_lm_a]
% Example                 blinden      -->   blind      en
% Examples                (Blindenschule, Totensonntag, Behindertenpolitik)


% Linear Precedence and Feature Vectors:

stemLm(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

       stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

       lm(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),


% Boundary Symbol and Bracketing
  BOUNDARY='#+',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemLm,

% X1 <-> X2 Equational Constraints
% X1_UMGL=X2_UMLD,
% X1_SCAT=X2_BCAT,

% Constraints on X1 Instantiations
  not(X1_LINK==linkImp),
  X1_SCAT='a',

% Constraints on X2 Instantiations

  X2_MID=='en_lm_a',

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,
  X1_SCAT=X0_SCAT,
% X1_NAT=X0_NAT,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_PRET=X0_PRET,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_GEN=X0_GEN,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS,

% X2 <-> X0 Feature Percolation
  X2_RBND=X0_RBND,
  X2_FLEX=X0_FLEX.

% what about INTERF

%---------------------------------------------------------------------------


% Rule17           word  -->  stem[rbndOpt/rbndFree]


% Linear Precedence and Feature Vectors:
```

```
word(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=word,

% Constraints on X1 Instantiations
  not(X1_RBND==rbndToBind),
  not(X1_LBND==lbndToBind),

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_SCAT=X0_SCAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_NAT=X0_NAT,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_FLEX=X0_FLEX,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_INTERF=X0_INTERF,
  X1_GEN=X0_GEN,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.


% ---------------------------------------------------------------------------------------

% Rule18a             suf   -->   sufn
% Example             ung   -->   ung


% Linear Precedence and Feature Vectors:

suf(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

sufn(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
```

```
  X0_MCAT=suf,

% Constraints on X1 Instantiations
  X1_NAT='+',

% X1 <-> X0 (Head) Feature Percolation
  X1_NAT=X0_NAT,
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_STR=X0_STR,
  X1_ADJ=X0_ADJ,
  X1_UMLD=X0_UMLD,
  X1_ABLD=X0_ABLD,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_SCAT=X0_SCAT,
  X1_FLEX=X0_FLEX,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_GEN=X0_GEN,
  X1_INTERF=X0_INTERF.


% ---------------------------------------------------------------------------

% Rule18b             suf   -->   sufnn
% Example             ier   -->   ier


% Linear Precedence and Feature Vectors:

suf(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

sufnn(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=suf,

% Constraints on X1 Instantiations
  X1_NAT='-',

% X1 <-> X0 (Head) Feature Percolation
  X1_NAT=X0_NAT,
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_BCAT=X0_BCAT,
  X1_STR=X0_STR,
  X1_ADJ=X0_ADJ,
  X1_UMLD=X0_UMLD,
  X1_ABLD=X0_ABLD,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_SCAT=X0_SCAT,
  X1_FLEX=X0_FLEX,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_GEN=X0_GEN,
  X1_INTERF=X0_INTERF.

%------------------------------------------------------------------------------

% Rule19a          inflection  -->  v_infl


% Linear Precedence and Feature Vectors:

inflection(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

v_infl(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=inflection,

% Constraints on X1 Instantiations
  X1_SCAT=v,

% X1 <-> X0 Feature Percolation
  X1_MID=X0_MID,
  X1_PRET=X0_PRET,
  X1_FLEX=X0_FLEX,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_RBND=X0_RBND,
  X1_LBND=X0_LBND,
  X1_NAT=X0_NAT,
  X1_SCAT=X0_SCAT,
  X1_REG=X0_REG.


%------------------------------------------------------------------------------

% Rule19b          inflection  -->  a_infl

inflection(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

a_infl(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),
% X0 Instantiations
  X0_MCAT=inflection,

% Constraints on X1 Instantiations
  X1_SCAT=a,

% X1 <-> X0 Feature Percolation
  X1_MID=X0_MID,
  X1_PRET=X0_PRET,
  X1_FLEX=X0_FLEX,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_RBND=X0_RBND,
  X1_LBND=X0_LBND,
  X1_NAT=X0_NAT,
  X1_SCAT=X0_SCAT,
  X1_REG=X0_REG.

%-----------------------------------------------------------------------------


% Rule19c          inflection  -->  n_infl


% Linear Precedence and Feature Vectors:

inflection(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

n_infl(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=inflection,

% Constraints on X1 Instantiations
  X1_SCAT=n,

% X1 <-> X0 Feature Percolation
  X1_MID=X0_MID,
  X1_PRET=X0_PRET,
  X1_FLEX=X0_FLEX,
  X1_VROOT=X0_VROOT,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_RBND=X0_RBND,
  X1_LBND=X0_LBND,
  X1_NAT=X0_NAT,
  X1_SCAT=X0_SCAT,
  X1_REG=X0_REG.

%-------------------------------------------------------------------------


% Rule20        stemCompound        -->   stemHyphened   stem
% Example       fahr-bereitschaft   -->   fahr-          bereitschaft


% Linear Precedence and Feature Vectors:

stemCompound(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemHyphened(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stem(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemCompound,

% X1 <-> X2 Equational Constraints

% Constraints on X1 Instantiations
  not(X1_LINK==linkObl),

% Constraints on X2 Instantiations

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_SCAT=X0_SCAT,
  X2_RBND=X0_RBND,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_NAT=X0_NAT,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_FLEX=X0_FLEX,
  X2_VROOT=X0_VROOT,
  X2_REG=X0_REG,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_INTERF=X0_INTERF,
  X2_GEN=X0_GEN,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%------------------------------------------------------------------
```

```
% Rule21a        stemHyphened   -->   stem  hyphen
% Example        fahr-          -->   fahr  -


% Linear Precedence and Feature Vectors:

stemHyphened(MID,X0_MCAT,COMB_SUF,LBND,LBND_ARG,RBND,RBND_ARG,BCAT,SCAT,NAT,SEP,
STR,ADJ,LINK,LINK_MID,UMGL,UMLB,UMLD,ABGL,ABLB,ABLD,ABL_SUF_MID,PRET,REG,INTERF,
INTERF_ARG,GEN,HEAD_MID,ARG_MID,VROOT,STEMCLASS,SUFFIXCLASS,FLEX,PARTSTEM,PSTEMS,
OSTEMS):-

stem(MID,MCAT,COMB_SUF,LBND,LBND_ARG,RBND,RBND_ARG,BCAT,SCAT,NAT,SEP,STR,ADJ,LINK,
LINK_MID,UMGL,UMLB,UMLD,ABGL,ABLB,ABLD,ABL_SUF_MID,PRET,REG,INTERF,INTERF_ARG,GEN,
HEAD_MID,ARG_MID,VROOT,STEMCLASS,SUFFIXCLASS,FLEX,PARTSTEM,PSTEMS,OSTEMS),

hyphen(_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),


% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemHyphened.

%----------------------------------------------------------------------

% Rule2b        stemHyphened   -->   stemLm   hyphen
% Example       arbeits-       -->   arbeits  -


% Linear Precedence and Feature Vectors:

stemHyphened(MID,X0_MCAT,COMB_SUF,LBND,LBND_ARG,RBND,RBND_ARG,BCAT,SCAT,NAT,SEP,
STR,ADJ,LINK,LINK_MID,UMGL,UMLB,UMLD,ABGL,ABLB,ABLD,ABL_SUF_MID,PRET,REG,INTERF,
INTERF_ARG,GEN,HEAD_MID,ARG_MID,VROOT,STEMCLASS,SUFFIXCLASS,FLEX,PARTSTEM,PSTEMS,
OSTEMS):-

stemLm(MID,MCAT,COMB_SUF,LBND,LBND_ARG,RBND,RBND_ARG,BCAT,SCAT,NAT,SEP,STR,ADJ,
LINK,LINK_MID,UMGL,UMLB,UMLD,ABGL,ABLB,ABLD,ABL_SUF_MID,PRET,REG,INTERF,INTERF_ARG,
GEN,HEAD_MID,ARG_MID,VROOT,STEMCLASS,SUFFIXCLASS,FLEX,PARTSTEM,PSTEMS,OSTEMS),

hyphen(_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),


% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemHyphened.


%----------------------------------------------------------------------

% Rule22        stem[n] --> word[v,infinitive]


% Linear Precedence and Feature Vectors:

stem(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

word(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=stem,
  X0_SCAT=n,
  X0_FLEX='akk,sg;dat,sg;nom,sg',
  X0_SUFFIXCLASS='Nomen_Treffen',
  X0_GEN='n',
  X0_LINK=linkOpt,
  X0_LINK_MID=['s_lm_n'],
  X0_RBND=rbndOpt,

% Constraints on X1 Instantiations
  X1_RBND==rbndFree,
% X1_PARTSTEM=inf,
  prefix_of_atom("non-finit-inf",X1_FLEX),
% not(X1_LBND==lbndToBind),

% X1 <-> X0 (Head) Feature Percolation
  X1_MID=X0_MID,
  X1_PRET=X0_PRET,
  X1_LBND=X0_LBND,
  X1_NAT=X0_NAT,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG.

%---------------------------------------------------------------------------------


% Rule23a        word[n] --> word[adj]


% Linear Precedence and Feature Vectors:

word(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

word(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=word,
  X0_SCAT=n,
  X0_FLEX=X1_FLEX,
  X0_SUFFIXCLASS='NA',
% X0_GEN  is somewhere in the FLEX Specification
  X0_LINK=linkOpt,

% Constraints on X1 Instantiations
  X1_SCAT==a,
  X1_RBND=rbndFree,          % since the whole inflected word is converted.
  not(X1_LBND==lbndToBind),

% X1 <-> X0 (Head) Feature Percolation
  X1_MID=X0_MID,
  X1_PRET=X0_PRET,
  X1_RBND=X0_RBND,
  X1_LBND=X0_LBND,
  X1_NAT=X0_NAT,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG.

%-----------------------------------------------------------------------------------


% Rule23b        word[n] --> stem[adj,unflekt]


% Linear Precedence and Feature Vectors:

word(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

word(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=word,
  X0_RBND=rbndFree,
  X0_SCAT=n,
  X0_FLEX=X1_FLEX,
  X0_SUFFIXCLASS='NA',
% X0_GEN  is somewhere in the FLEX Specification
  X0_LINK=linkOpt,

% Constraints on X1 Instantiations
  X1_SCAT==a,
  not(X1_RBND=rbndFree),          % complementary to rule 23a
  not(X1_LBND==lbndToBind),
  sublistmember("unflekt",X1_FLEX),

% X1 <-> X0 (Head) Feature Percolation
  X1_MID=X0_MID,
  X1_PRET=X0_PRET,
  X1_LBND=X0_LBND,
  X1_NAT=X0_NAT,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG.

%-----------------------------------------------------------------------------------

% Rule24a            stemPrefixed       -->  part  root[CCAT==v,SCAT==n]
% Example            anstand            -->  an    stand


% Linear Precedence and Feature Vectors:
```

```
stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

root(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X2 Equational Constraints
  X1_BCAT=X2_BCAT,

% Constraints on X1 Instantiations
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',

% Constraints on X2 instantiations
  X2_SCAT==n,
  X2_BCAT==v,               %
  not(X2_LBND==lbndFree),

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.
```

```
%-----------------------------------------------------------------------------------

% Rule24b             stemPrefixed        -->  part  stemPre1fixed [CCAT==v,SCAT==n]
% Example             anbetracht          -->  an    betracht


% Linear Precedence and Feature Vectors:

stemPrefixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPre1fixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPrefixed,

% X1 <-> X2 Equational Constraints
  X1_BCAT=X2_BCAT,

% Constraints on X1 Instantiations
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',

% Constraints on X2 instantiations
  X2_SCAT==n,
  X2_BCAT==v,               %
  not(X2_LBND==lbndFree),

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%-----------------------------------------------------------------------

% Rule24c             stemPre1fixed        -->   pre1  root[CCAT==v,SCAT==n]
% Example             verstand             -->   ver   stand


% Linear Precedence and Feature Vectors:

stemPre1fixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

pre1(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

root(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='|+',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPre1fixed,

% X1 <-> X2 Equational Constraints
  X1_BCAT=X2_BCAT,

% Constraints on X1 Instantiations
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='-',
% X1_ADJ='+',

% Constraints on X2 instantiations
  X2_SCAT==n,
  X2_BCAT==v,               %
  not(X2_LBND==lbndFree),

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_BCAT=X0_BCAT,
  X2_SCAT=X0_SCAT,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_VROOT=X0_VROOT,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_FLEX=X0_FLEX,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-------------------------------------------------------------------------------

% Rule24d             stemPre1fixed        -->   pre1[ver|be]  stemPrefixed[CCAT==v,
SCAT==n]
% Example             verbeamten           -->   ver           beamten
% Example             bevorzugen           -->   be            vorzugen


% Linear Precedence and Feature Vectors:

stemPre1fixed(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

pre1(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemPrefixed(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='|+',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemPre1fixed,
  X0_SCAT=v,
  X0_PRET='-',
  X0_REG='+',
  X0_PARTSTEM='+',
  X0_VROOT='FRAG',
  X0_STEMCLASS='FRAGEN',
  member(X0_SUFFIXCLASS,['SAGEN','WARTEN','FASSEN','BEGEISTERN']),
% X1 <-> X2 Equational Constraints
% X1_BCAT=X2_BCAT,

% Constraints on X1 Instantiations
  inst_member(X1_MID,[be_pre1_v_v,ver_pre1_v_v]),
% X1_SEP='-',
% X1_NAT='+',
% X1_STR='-',
% X1_ADJ='+',

% Constraints on X2 instantiations
  X2_SCAT==n,
  X2_BCAT==v,              %
  not(X2_LBND==lbndFree),

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_RBND=X0_RBND,
  X2_NAT=X0_NAT,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_INTERF=X0_INTERF,
  X2_INTERF_ARG=X0_INTERF_ARG,
  X2_GEN=X0_GEN,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------------

% Rule25          word   -->   stemInflected
% Example         dabist -->   dabist


% Linear Precedence and Feature Vectors:

word(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stemInflected(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

% X0 Instantiations
  X0_MCAT=word,

% Constraints on X1 Instantiations
  X1_RBND==rbndFree,
  not(X1_LBND==lbndToBind),

% X1 <-> X0 (Head) Feature Percolation
  X1_LBND=X0_LBND,
  X1_RBND=X0_RBND,
  X1_SCAT=X0_SCAT,
  X1_LINK=X0_LINK,
  X1_LINK_MID=X0_LINK_MID,
  X1_NAT=X0_NAT,
  X1_SUFFIXCLASS=X0_SUFFIXCLASS,
  X1_UMGL=X0_UMGL,
  X1_UMLB=X0_UMLB,
  X1_PRET=X0_PRET,
  X1_VROOT=X0_VROOT,
  X1_REG=X0_REG,
  X1_FLEX=X0_FLEX,
  X1_STEMCLASS=X0_STEMCLASS,
  X1_INTERF=X0_INTERF,
  X1_GEN=X0_GEN,
  X1_PARTSTEM=X0_PARTSTEM,
  X1_PSTEMS=X0_PSTEMS,
  X1_OSTEMS=X0_OSTEMS.

%----------------------------------------------------------------------

% Rule26             word            -->   part  stemInflected
% Example            dabist          -->   da    bist


% Linear Precedence and Feature Vectors:

ord(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

part(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemInflected(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=word,

% X1 <-> X2 Equational Constraints
  X1_BCAT=X2_SCAT,

% Constraints on X1 Instantiations
% X1_SEP='+',
% X1_NAT='+',
% X1_STR='+',
% X1_ADJ='-',

% Constraints on X2 Instantiations
  X2_SCAT=v,
% not(X2_LBND==lbndFree),            %% auskommentiert, da 'bist', 'warst' derzeit alle lbndFree -> aendern.
  X2_RBND=rbndFree,
% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_SCAT=X0_SCAT,
  X2_RBND=X0_RBND,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_NAT=X0_NAT,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_FLEX=X0_FLEX,
  X2_VROOT=X0_VROOT,
  X2_REG=X0_REG,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_INTERF=X0_INTERF,
% X2_GEN=X0_GEN,              % gen ist in dieser Regel irrelevant, da sie nur verben betrifft.
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.

%-------------------------------------------------------------------------

% Rule27            stemCompound     -->   stem[num]   stemUndNum
% Example           f"unfundzwanzig  -->   f"unf       undzwanzig


% Linear Precedence and Feature Vectors:

stemCompound(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

stem(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stemUndNum(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,
X2_BCAT,X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X2_UMLD,X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,
X2_GEN,X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

%% rule49 and 50 are necessary, as usually the category 'word' is not
%% allowed in compounds, neither is 'konj'.

% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemCompound,

% X1 <-> X2 Equational Constraints

% Constraints on X1 Instantiations
```

```
  X1_LBND=X0_LBND,
  not(X1_LINK==linkObl),
  not(X1_RBND=='rbndFree'),
  X1_SCAT=='num',

% Constraints on X2 instantiations
% not(X2_LBND=='lbndFree'),
  inst_member(X2_SCAT,[a,num]),          %% ordinal numbers are 'a'

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_SCAT=X0_SCAT,
  X2_RBND=X0_RBND,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_NAT=X0_NAT,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_FLEX=X0_FLEX,
  X2_VROOT=X0_VROOT,
  X2_REG=X0_REG,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_INTERF=X0_INTERF,
  X2_GEN=X0_GEN,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


% --------------------------------------------------------------------------------

% Rule28     stemUndNum   -->   word[und]   stem[num/a]
% Example    undzwanzig   -->   und         zwanzig


% Linear Precedence and Feature Vectors:

stemUndNum(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,X0_BCAT,
X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,X0_UMLD,
X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,X0_GEN,
X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

word(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,X1_BCAT,
X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,X1_UMLD,
X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,
X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stem(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),

%% rule49 and 50 are necessary, as usually the category 'word' is not
%% allowed in compounds, neither is 'konj'.

% Boundary Symbol and Bracketing
```

```
  BOUNDARY='#',
  BRACKETING=right_bracket,

% X0 Instantiations
  X0_MCAT=stemUndNum,

% X1 <-> X2 Equational Constraints

% Constraints on X1 Instantiations
% not(X1_LINK==linkObl),
% not(X1_RBND=='rbndFree'),
% X1_SCAT=='konj',
  X1_MID=='und',

% Constraints on X2 instantiations
  not(X2_LBND=='lbndFree'),
  inst_member(X2_SCAT,[a,num]),     %% ordinal numbers are 'a'

% X1 <-> X0 Feature Percolation
  X1_LBND=X0_LBND,

% X2 <-> X0 (Head) Feature Percolation
  X2_SCAT=X0_SCAT,
  X2_RBND=X0_RBND,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_NAT=X0_NAT,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_FLEX=X0_FLEX,
  X2_VROOT=X0_VROOT,
  X2_REG=X0_REG,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_INTERF=X0_INTERF,
  X2_GEN=X0_GEN,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------------

% Rule29              stemCompound        -->   wordHyphened   stem
% Example             silberner-habicht   -->   silberner-     habicht


% Linear Precedence and Feature Vectors:

stemCompound(X0_MID,X0_MCAT,X0_COMB_SUF,X0_LBND,X0_LBND_ARG,X0_RBND,X0_RBND_ARG,
X0_BCAT,X0_SCAT,X0_NAT,X0_SEP,X0_STR,X0_ADJ,X0_LINK,X0_LINK_MID,X0_UMGL,X0_UMLB,
X0_UMLD,X0_ABGL,X0_ABLB,X0_ABLD,X0_ABL_SUF_MID,X0_PRET,X0_REG,X0_INTERF,X0_INTERF_ARG,
X0_GEN,X0_HEAD_MID,X0_ARG_MID,X0_VROOT,X0_STEMCLASS,X0_SUFFIXCLASS,X0_FLEX,X0_PARTSTEM,
X0_PSTEMS,X0_OSTEMS):-

wordHyphened(X1_MID,X1_MCAT,X1_COMB_SUF,X1_LBND,X1_LBND_ARG,X1_RBND,X1_RBND_ARG,
X1_BCAT,X1_SCAT,X1_NAT,X1_SEP,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
X1_UMLD,X1_ABGL,X1_ABLB,X1_ABLD,X1_ABL_SUF_MID,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,
X1_GEN,X1_HEAD_MID,X1_ARG_MID,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,
X1_PSTEMS,X1_OSTEMS),

stem(X2_MID,X2_MCAT,X2_COMB_SUF,X2_LBND,X2_LBND_ARG,X2_RBND,X2_RBND_ARG,X2_BCAT,
X2_SCAT,X2_NAT,X2_SEP,X2_STR,X2_ADJ,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,X2_UMLD,
X2_ABGL,X2_ABLB,X2_ABLD,X2_ABL_SUF_MID,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,
X2_HEAD_MID,X2_ARG_MID,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,
X2_PSTEMS,X2_OSTEMS),


% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT=stemCompound,

% X1 <-> X2 Equational Constraints

% Constraints on X1 Instantiations
  X1_LBND=X0_LBND,

% Constraints on X2 instantiations
  X2_SCAT=X0_SCAT,
  X2_RBND=X0_RBND,
  X2_LINK=X0_LINK,
  X2_LINK_MID=X0_LINK_MID,
  X2_NAT=X0_NAT,
  X2_SUFFIXCLASS=X0_SUFFIXCLASS,
  X2_STEMCLASS=X0_STEMCLASS,
  X2_UMGL=X0_UMGL,
  X2_UMLB=X0_UMLB,
  X2_PRET=X0_PRET,
  X2_FLEX=X0_FLEX,
  X2_VROOT=X0_VROOT,
  X2_REG=X0_REG,
  X2_INTERF=X0_INTERF,
  X2_GEN=X0_GEN,
  X2_PARTSTEM=X0_PARTSTEM,
  X2_PSTEMS=X0_PSTEMS,
  X2_OSTEMS=X0_OSTEMS.


%-----------------------------------------------------------------

% Rule30a             wordHyphened    -->    word        hyphen
% Example             silberner-      -->    silberner   -


wordHyphened(MID,X0_MCAT,COMB_SUF,LBND,LBND_ARG,RBND,RBND_ARG,BCAT,SCAT,NAT,SEP,
STR,ADJ,LINK,LINK_MID,UMGL,UMLB,UMLD,ABGL,ABLB,ABLD,ABL_SUF_MID,PRET,REG,INTERF,
INTERF_ARG,GEN,HEAD_MID,ARG_MID,VROOT,STEMCLASS,SUFFIXCLASS,FLEX,PARTSTEM,PSTEMS,
OSTEMS):-

word(MID,MCAT,COMB_SUF,LBND,LBND_ARG,RBND,RBND_ARG,BCAT,SCAT,NAT,SEP,STR,ADJ,LINK,
LINK_MID,UMGL,UMLB,UMLD,ABGL,ABLB,ABLD,ABL_SUF_MID,PRET,REG,INTERF,INTERF_ARG,GEN,
HEAD_MID,ARG_MID,VROOT,STEMCLASS,SUFFIXCLASS,FLEX,PARTSTEM,PSTEMS,OSTEMS),

hyphen(_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_),


% Boundary Symbol and Bracketing
  BOUNDARY='#',
  BRACKETING=left_bracket,

% X0 Instantiations
  X0_MCAT='wordHyphened'.
```

```
%----------------------------------------------------------------------
```

C. Perl grammar-to-parser converter

```
#!/vol/gnu/bin/perl
# konvert.prl
# Harald L"ungen

## 30.9.98:
## The variable are instantiated or renamed as far as possible,
## and the annotated equatetions are removed as as far as possible




# konvert.prl converts a dcg-grammar in a certain format (cf
# 'grammar.format.readme.txt' into a PROLOG-left-corner-parser based on
# Matsumoto and Naumann/Langer 1994.


# open the following Streams:


open(MORPHOTACTICS,@ARGV[0]);          # open the grammar to read
open(LEFT_CORNER,">@ARGV[1]");         # open the left-corner.pl-file to write
open(LINK,">link.simple.pl");              # open the link-file to write
open(TERMINATION,">termination.pl");       # open to termination-file write
open(PS_SKELETON,">ps_skeleton.txt");      # open to write the ps-skeleton
                                               # which is implicitly contained in
                                               # the grammar into a separate file



# We have opened a separate file 'termination.pl', but its contents have
# to be  put into the same file as 'left_corner.pl', since all definitions of
# one predicate  have to be in the same file otherwise sicstus won't work.
# the merging is done in the shell script konvert.sh


$flag  = "begin";



#----------------------------------------------------------------------------

# read the first line of the grammar:

while($flag eq "begin"){

    $line = <MORPHOTACTICS>;
    while($line =~ /^\s*$/){        # if this line contains only white spaces
                            # read the next line
$line = <MORPHOTACTICS>;
unless($line){last}
    };
    unless($line){last};
    chop($line);

#---------------------------------------------------------------------------

## initialise the arrays for the annotations
## every time in this loop anew:
    @annotations=();
    @not=();
    @conditions=();


#-----------------------------------------------------------------------------

#  check if the current line is the skeletal specification of a ps-rule,
#  and put it into the string $rule:


    if($line =~ /^ *\% *[Rr]ule.*/){               #    "% rule lhs --> left right"
                                            # or "% rule lhs --> rhs"
$rule = $line;
$rule =~ s/\%//g;                           # remove '%' globally
$rule =~ s/ *//;                            # remove occurences of blank
                                            # at beginning of line,
                                            # in order for @rule[0] not to be empty


#  split the string rule into parts definined by the separator white space

@rule = split(/\s+/,$rule);            # /\s+/ are one or more occurences of
                                       # white spaces in perl.
    }

    else { printf("Error: skeletal rule was expected\n\n");}

#------------------------------------------------------------------------------


#  check if this skeletal rule is unary or binary
#  It is written as a PROLOG-comment like this:
#    binary rule:     <lhs> --> <rhs_left> <rhs_right>
#    unary rule:      <lhs< --> <rhs_left>



    if(@rule[4]){
$rule_sort = "binary"}
    else {if(@rule[3]) {$rule_sort = "unary"}
  else { printf("something is wrong with the rule format,
               it is neither binary nor unary\n\n")
 }
      };


# ----------------------------------------------------------------------------

# read the true lhs of the rule (i.e. the PROLOG) predicate


    $line = <MORPHOTACTICS>;
    while ("\n" eq $line || $line =~ /^ *\%.*/) {
$line = <MORPHOTACTICS>;
unless($line){last}
    };
    unless($line){last};
    chop($line);

    $line =~ s/^\s*//;
    $line =~ s/\:-\.*//;
    @lhs = split(/[\s,()]+/, $line);
```

```
# ---------------------------------------------------------------------------

#         read rhs_left of rule (i.e. the PROLOG) predicate
#         when rule is unary, rhs_left stand for rhs.

    $line = <MORPHOTACTICS>;
    while ("\n" eq $line) {
$line = <MORPHOTACTICS>;
unless($line){last}
    };
    unless($line){last};
    chop($line);


    $line =~ s/^\s*//;
    @rhs_left = split(/[\s,()]+/, $line);

# ---------------------------------------------------------------------------


#     read rhs_right of rule, if rule was binary

    if($rule_sort eq "binary"){

$line = <MORPHOTACTICS>;
while ("\n" eq $line) {
    $line = <MORPHOTACTICS>;
    unless($line){last}
};
unless($line){last};
chop($line);


$line =~ s/^\s*//;
@rhs_right = split(/[\t ,()]+/, $line);

    }

#----------------------------------------------------------------------------

# read the annotational equations,

    $line = <MORPHOTACTICS>;
    unless($line){last};
    chop($line);

    $i=0;
    $n=0;
    until($line =~ /\s*\%+\s*\-+\s*/){      # i.e. until the separation line "%-------------"
$line =~ s/(\%.*)+//;               # remove comments

if($line =~ /^\s*$/){               # keep skipping empty lines
    $line = <MORPHOTACTICS>;
    unless($line){last};
    chop($line);
}

if($line =~ /.+\=.+/ || $line =~ /member/  || $line =~ /prefix/){

    # if line contains an equation
    # or a prefix statement
    # or a member statement (nothing else is possible):
```

```
    $line =~ s/^\s*//;                             # cut off white spaces at begin of line.
    $line =~ s/(\%.*)+//;                          # remove comments
    $line =~ s/[\.,]\s*$|[\.,]\s*\%.*$//;          # cut off everything after comma,
 if it is white spaces
    # or white spaces followed by comment
    # followed by end-of-line

# If the annotation contains a not(...) statement concerning an X2_Variable,
# or a member statement concerning an X2_Variable,
# or a '==' - statement concerning an X2_Variable,
# it has to be placed after the call of the predicate of X2. So, here it hast to be
# stored in a separate array called @not:

    if(($line=~/.*not.*/ || $line=~/.*member.*/ || $line =~ /.*\=\=.*/) && $line=~/.*X2.*/){
@not[$n++]=$line;
    }

# Otherwise store the annotation in the array @annotations:

    else {
if ($line) {
    @annotations[$i++]=$line;
}
    };


    $line=<MORPHOTACTICS>;
    unless($line){last};
    chop($line);
}
    }

    unless($line){last};                        # otherwise $line contains %----
    # and can be skipped in the next loop.



##############################################################################
## split the annotations for instantiation and variable substitutionpurposes:#
##############################################################################


    @splarray=();
    @lequation=();
    @requation=();
    @substarray=();



    for($h=$uuu=$vvv=0;$annotations[$h];$h++){


$teststring=$annotations[$h];
unless(($teststring =~ /prefix/) || ($teststring =~ /member/) || $teststring =~ /not/ || $teststring =~ /\=\=/){
    @splarray=split("=",$annotations[$h]);
    $lequation[$vvv]=$splarray[0];
    $requation[$vvv++]=$splarray[1];

}

## remember: the not/member/'=='/prefix/ - statements in the @annotations
## must be kept, i.e. put into @conditions,
## as only the those not-statements concerning an X2_Variable
## are in the @not -array

else {
    $conditions[$uuu++]=$annotations[$h];
}
    }



## substitute the variables if in lequation through requation,
## i.e. create the a substitution_table @substarray[][]



## the following huge iteration checks if there is a
## substitution chain for all the variables stored now in
## @lequation[$i].
## the variable in @lequation[$i] ist stored in
## @substarray[$i][0], and the rest of the Variables or
## values in the chain are store successively in the
## columns @substarray[$i][$j]; with $j > 0.

## care is taken that no variables are mentioned
## twice and it is checked with each new loop
## where the varaible in question has been
## seen before.

## the subroutine &chain_flag checks
## whether the iteration could find all
## elements of the substitution chain.
## it could miss something, if the
## chain given through the annotated
## equations in morphotaktik.pl is
## established through more than Three
## equation i.e.:
##   A=B
##   B=C
##   C=D
##   D=E
## will cause the E to be missed in the chain,
## and if that is the case,
## find_chain will print an error message to stdout!
## this was done
## since it is very very unlikely
## that such chains longer than 3 equations
## occur in a morphotactics of the kind
## of morphotaktik.pl:


      for($h=$g=0;$lequation[$h] ne "" ;$h++){
$s=0;
unless (&already_in_table($lequation[$h],@substarray)) {
    $substarray[$g][$s++]=$lequation[$h];
    for($r=$h;$lequation[$r] ne "" ;$r++){
if($lequation[$h] eq $lequation[$r]){
    $substarray[$g][$s++]=$requation[$r];
    unless($requation[$r] =~ /^[\'?a-z]/){
for($t=0;$lequation[$t] ne ""; $t++){
    if($requation[$r] eq $lequation[$t] && $t != $r){
$substarray[$g][$s++]=$requation[$t];
```

```
unless($requation[$t] =~ /^[\'?a-z]/){
    for($f=0;$lequation[$f] ne ""; $f++){
if($requation[$t] eq $lequation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$requation[$f];

    &chain_flag($substarray[$g][$s-1],$rule);

}
    }
    for($f=0;$requation[$f] ne ""; $f++){
if($requation[$t] eq $requation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$lequation[$f];

    &chain_flag($substarray[$g][$s-1],$rule);

}
    };
}



    }
}
for($t=0;$requation[$t] ne ""; $t++){
    if($requation[$r] eq $requation[$t] && $t != $r){
$substarray[$g][$s++]=$lequation[$t];



unless($lequation[$t] =~ /^[\'?a-z]/){
    for($f=0;$lequation[$f] ne ""; $f++){
if($lequation[$t] eq $lequation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$requation[$f];

    &chain_flag($substarray[$g][$s-1],$rule);

}
    }
    for($f=0;$requation[$f] ne ""; $f++){
if($lequation[$t] eq $requation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$lequation[$f];

    &chain_flag($substarray[$g][$s-1],$rule);

}
    };
      }



    }
};
    }
}
if($lequation[$h] eq $requation[$r]){
    $substarray[$g][$s++]=$lequation[$r];

    unless($lequation[$r] =~ /^[\'?a-z]/){
for($t=0;$lequation[$t] ne ""; $t++){
    if($lequation[$r] eq $lequation[$t] && $t != $r){
$substarray[$g][$s++]=$requation[$t];
```

```
unless($requation[$t] =~ /^[\'?a-z]/){
    for($f=0;$lequation[$f] ne ""; $f++){
if($requation[$t] eq $lequation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$requation[$f];

    &chain_flag($substarray[$g][$s-1],$rule);

}
  }
    for($f=0;$requation[$f] ne ""; $f++){
if($requation[$t] eq $requation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$lequation[$f];

  &chain_flag($substarray[$g][$s-1],$rule);

}
    };
}


    }
      }
for($t=0;$requation[$t] ne ""; $t++){
    if($lequation[$r] eq $requation[$t] && $t != $r){
$substarray[$g][$s++]=$lequation[$t];


unless($lequation[$t] =~ /^[\'?a-z]/){
    for($f=0;$lequation[$f] ne ""; $f++){
if($lequation[$t] eq $lequation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$requation[$f];

    &chain_flag($substarray[$g][$s-1],$rule);

}
    }
    for($f=0;$requation[$f] ne ""; $f++){
if($lequation[$t] eq $requation[$f] && $f != $t && $f != $r){
    $substarray[$g][$s++]=$lequation[$f];

    &chain_flag($substarray[$g][$s-1],$rule);

}
    };
}



    }
};
    }
}
    };
    $g++;                    ## this count for the array columns must be
                             ## increased now!
}
    }


#################################################
## Re-order the @substarray[][]:           #
#################################################
```

```
    for($g=0;$substarray[$g];$g++){

### first, some special cases have to be checked
### and to be kept in mind:
### BRACKETING
### BOUNDARY
### where it is known, that the second
### componenent of the array
### contains the atomic value.

if($substarray[$g][0] eq "BRACKETING"){
    $bracketing = $substarray[$g][1];
}
if($substarray[$g][0] eq "BOUNDARY"){
    $boundary = $substarray[$g][1];
}

###  establish, what is to be substituted
        ###  and store it as the first element of a column
        ###  it is either an atom or the first element:

if($v=&atom_or_listofatoms_in_column($g,@substarray)){
    $lemma=$substarray[$g][0];
    $substarray[$g][0]=$substarray[$g][$v];
    $substarray[$g][$v]=$lemma;
}

### X0_MCAT (is to be used as a special argument in the goal predicate
### of this rule as well, see below.

for($ga=0;$substarray[$g][$ga];$ga++){
    if($substarray[$g][$ga] eq "X0_MCAT"){
$x0_mcat = $substarray[$g][0];
    }
}

    }



########################################################################
### a print-out of the present @substarray[][]:
##
##
##    for($g=0;$substarray[$g] ;$g++){
## for($s=0;$substarray[$g][$s];$s++){
##     printf("%s\t",$substarray[$g][$s]);
## }
## print("\n");
##    }
##    printf("\nDas war die Regel %d",$regelnummer++);
##    printf("\n####################################\n");
########################################################################



###
##  a processing message for the impatient user:
###

    printf("  substituting variable names in %s: \t%s %s %s\n",
   $rule[0],$rule[1],$rule[2],$rule[3],$rule[4]);


#################################################################
### using the substarray to substitute
### Prolog Variable names in the @lhs, @rhs_left, and @rhs_right:
### and in @not and in @conditions
### substitute the first element of a column
### for all the elements of the column
### in the @lhs, @rhs_left, @rhs_right,@not and in @conditions
#################################################################

    ## note: In the @conditions and @not -annotations,
    ## the variables are substituted only by string_subtitution,
    ## since the variables are not stored componentwise in
    ## subarrays, but are hidden in the strings:
    ## hoping this does not lead to faulty substitutions...

    for($d=0; $substarray[$d]; $d++){
  for($e=1; $substarray[$d][$e]; $e++){
      @lhs =        &substitute($substarray[$d][0],$substarray[$d][$e],@lhs);
      @rhs_right =  &substitute($substarray[$d][0],$substarray[$d][$e],@rhs_right);
      @rhs_left =   &substitute($substarray[$d][0],$substarray[$d][$e],@rhs_left);
      @conditions = &string_substitute($substarray[$d][0],$substarray[$d][$e],@conditions);
      @not =        &string_substitute($substarray[$d][0],$substarray[$d][$e],@not);
  };
    };


#############################################################
## For all the variables not substitued,
## introduce anonymous Variables '_':
## into @lhs, @rhs_left, @rhs_right
#############################################################



## first, write all the substituted variable names
## ( to be found in $substarray[$a][0] )
## into the new array @substitutingarray:


    @substitutingarray =();
    for($a=0;$substarray[$a][0];$a++){
$substitutingarray[$a]=$substarray[$a][0];
    }



## The '_' can be inserted, if the variable in @lhs
## is not in @substitutingarray,
## and not string - contained in @not and @conditions:

    for($a=1;$lhs[$a];$a++){
unless(&element_strings($lhs[$a],@substitutingarray)||
       &substring_in_array($lhs[$a],@not) ||
       &substring_in_array($lhs[$a],@conditions)){
    $lhs[$a] = "_";
};
    }


## the same for @rhs_left:
    for($a=1;$rhs_left[$a];$a++){
unless(&element_strings($rhs_left[$a],@substitutingarray) ||
       &substring_in_array($rhs_left[$a],@not) ||
       &substring_in_array($rhs_left[$a],@conditions)) {
    $rhs_left[$a] = "_";
};
    }


## the same for @rhs_right:

    for($a=1;$rhs_right[$a];$a++){
unless(&element_strings($rhs_right[$a],@substitutingarray)||
       &substring_in_array($rhs_right[$a],@not) ||
       &substring_in_array($rhs_right[$a],@conditions)) {
    $rhs_right[$a] = "_";
};
    }




###################################################################
###################################################################
#####                                                       #####
#####   C O N S T R U C T I N G   T H E   O U T P U T       #####
#####                                                       #####
###################################################################
###################################################################


# ------------------------------------------------------------------------------------

# print ps-rule into a separate file as well:

  printf(PS_SKELETON "%% %s\n",  $rule);

# ------------------------------------------------------------------------------------



# print-out der left-corner-format Regel:


# a.) printout von @rhs_left comes first, since it's
#     the left corner:


    printf(LEFT_CORNER "%% %s\n\n",  $rule);
    printf(LEFT_CORNER "%s(G,X,Y,\n\n",@rhs_left[0]);
    printf(LEFT_CORNER "[");
    $k=1;
    while(@rhs_left[$k+1]){
printf(LEFT_CORNER "%s,",@rhs_left[$k++])}
    printf(LEFT_CORNER "%s],\n",@rhs_left[$k]);
    printf(LEFT_CORNER "\tFsOut,\n\n");
    printf(LEFT_CORNER "PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,
PhonStructOut,OrthStructIn,OrthStructOut):-\n\n");


# Print out the conditions (member, prefix-statements)
# stored in @conditions:
    for($j=0;$conditions[$j];$j++){
printf(LEFT_CORNER "%s,\n",$conditions[$j]);
    };


# print the link-relation-condition:

    printf(LEFT_CORNER "\n");
    printf(LEFT_CORNER "link(%s,G),\n",@lhs[0]);

# if the rule was binary, here the goal-predicate must follow:

    if($rule_sort eq "binary"){

printf(LEFT_CORNER "goal(%s,X,Z,\n",@rhs_right[0]);
printf(LEFT_CORNER "_,\n");
printf(LEFT_CORNER "[");
#  printf(LEFT_CORNER "],\n\t_,\n");

$l=1;
while(@rhs_right[$l+1]){
    printf(LEFT_CORNER "%s,",@rhs_right[$l++])}
printf(LEFT_CORNER "%s",@rhs_right[$l]);
printf(LEFT_CORNER "],\n");
printf(LEFT_CORNER "PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,
OrthStructIn,OrthStructZ,$bracketing,$boundary,$x0_mcat),\n\n");

# still if the rule was binary, here follows the printout of the @not-array (s.a.):

$p=0;
if($n!=0){                          # i.e. if something exists in @not
    while($p<$n){
printf(LEFT_CORNER "%s,\n",@not[$p++])
}
}

# still if the rule was binary, the @lhs has to follow in the following format:

printf(LEFT_CORNER "\n%s(G,Z,Y,\n",@lhs[0]);
printf(LEFT_CORNER "[");
$m=1;
while(@lhs[$m+1]){
    printf(LEFT_CORNER "%s,",@lhs[$m++])}
printf(LEFT_CORNER "%s],\n",@lhs[$m]);
printf(LEFT_CORNER "\tFsOut,\n\n");
printf(LEFT_CORNER "PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,
OrthStructZ,OrthStructOut).\n\n\n");

    }

# if the rule was unary, the printout of the @lhs follows here in the
# following format:


    if($rule_sort eq "unary"){

# if there is something in the @not-array, print it out
# here, namely after the other annotations:

$p=0;
if($n!=0){                          # i.e. if something exists in @not
```

```
    while($p<$n){
printf(LEFT_CORNER "%s,\n",@not[$p++])
}
}

printf(LEFT_CORNER "\n%s(G,X,Y,\n",@lhs[0]);
printf(LEFT_CORNER "[");

$l=1;
while(@lhs[$l+1]){
    printf(LEFT_CORNER "%s,",@lhs[$l++])}
printf(LEFT_CORNER "%s",@lhs[$l]);
printf(LEFT_CORNER "],\n\n");
printf(LEFT_CORNER "\tFsOut,\n\n");
printf(LEFT_CORNER "PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,
OrthStructIn,OrthStructOut).\n\n");
    }


# write simpleLink-predicates in link-file:

    printf(LINK "simpleLink(%s,%s).\n",@rhs_left[0],@lhs[0]);


    printf(LEFT_CORNER "%%---------------------------------------------------------------------------------------\n\n");


# Terminierungspraedikate
# write in terminal-file:


    if($rule_sort eq "unary"){
printf(TERMINATION "%s(%s,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).\n",@rhs_left[0],@rhs_left[0]);
printf(TERMINATION "%s(%s,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).\n",@lhs[0],@lhs[0]);}
    if($rule_sort eq "binary"){
printf(TERMINATION "%s(%s,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).\n",@rhs_right[0],@rhs_right[0]);
printf(TERMINATION "%s(%s,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).\n",@rhs_left[0],@rhs_left[0]);
printf(TERMINATION "%s(%s,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).\n",@lhs[0],@lhs[0])}

# das termination-file muss anschliessend uniq sortiert werden.


}


#--------------------------------------------------------------------------

printf("\n\n    grammar conversion finished!\n");
printf("    writing the goal-predicates...\n\n");

#--------------------------------------------------------------------------


# goal Praedikate.
# Unlike in Naumann/Langer 1994 there have to be two different versions of the
# goal-predicate, since a bracketing structure is to be constructed, and
# the bracketing-building-predicate constrStruct/6 works differently and has
# to be embedded differently in the goal-predicate according to whether it is
# a right-bracketing structure of a left-bracketing structure:

# write goal-predicate for left-bracketing structure-building:

printf(LEFT_CORNER "goal(G,X,Z,FsIn,FsOut,PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,
PhonStructIn,PhonStructOut,OrthStructIn,OrthStructOut,left_bracket,BOUNDARY,X0_MCAT):-\n
\tdict(MCAT,X,Y,_,_,FsIn,PhonSeg,OrthSeg,PhonStruct,OrthStruct),\n
\tconstrSeg(PhonSegIn,PhonSeg,PhonSegZ,BOUNDARY),
\tconstrSeg(OrthSegIn,OrthSeg,OrthSegZ,BOUNDARY),
\tconstrStruct(PhonStructIn,PhonStruct,PhonStructZ,left_bracket,X0_MCAT,MCAT),
\tconstrStruct(OrthStructIn,OrthStruct,OrthStructZ,left_bracket,X0_MCAT,MCAT),\n
\tP =.. [MCAT,G,Y,Z,FsIn,FsOut,PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,
PhonStructOut,OrthStructZ,OrthStructOut],\n
\tcall(P).\n\n\n");

#write goal-prediccate for right-bracketing structure-building:

printf(LEFT_CORNER "goal(G,X,Z,FsIn,FsOut,PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,
PhonStructIn,PhonStructOut,OrthStructIn,OrthStructOut,right_bracket,BOUNDARY,X0_MCAT):-\n
\tdict(MCAT,X,Y,_,_,FsIn,PhonSeg,OrthSeg,PhonStruct,OrthStruct),\n
\tconstrStruct([],PhonStruct,PhonStructMCAT,_,_,MCAT),
\tconstrStruct([],OrthStruct,OrthStructMCAT,_,_,MCAT),\n
\tP =.. [MCAT,G,Y,Z,FsIn,FsOut,PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructMCAT,
PhonStructZ,OrthStructMCAT,OrthStructZ],\n\n
\tconstrSeg(PhonSegIn,PhonSeg,PhonSegZ,BOUNDARY),
\tconstrSeg(OrthSegIn,OrthSeg,OrthSegZ,BOUNDARY),\n
\tcall(P),\n
\tconstrStruct(PhonStructIn,PhonStructZ,PhonStructOut,right_bracket,X0_MCAT,MCAT),
\tconstrStruct(OrthStructIn,OrthStructZ,OrthStructOut,right_bracket,X0_MCAT,MCAT).\n\n");


close();


#-----------------------------------------------------------------------------------

####################################################################
####################################################################
#####                                                          #####
#####            S U B R O U T I N E S                         #####
#####                                                          #####
####################################################################
####################################################################


###################################################
##            &chain_flag                         #
###################################################

sub chain_flag{

    my ($value, $regel) = @_;

    printf("\nkonvert.v.prl: WARNING:\nvariable matching chain at %s
              could not be explored further in\n%s\n",
    $value,$regel);
    return;
}


###################################################
##            &already_in_tabel                   #
###################################################
```

```
sub already_in_table{
    @table = ();
    my ($item, @table) = @_;

    for ($spd=0;$table[$spd][0];$spd++) {
for ($gr=0;$table[$spd][$gr];$gr++) {
    if($item eq $table[$spd][$gr]) {
return 1;
    }
};
    };
    return 0;
}


##################################################
##             &atom_or_listofatoms_in_column                  #
##################################################


sub atom_or_listofatoms_in_column{

    @column=();
    my ($c, @column) = @_;

    for($zl=1;$column[$c][$zl];$zl++){

if($column[$c][$zl]=~/^\[?[a-z]/ || $column[$c][$zl]=~/^\[?\'./){
   ##  i.e. an atom or a quoted atom,
   ##  or a list (\[) of atoms.
## if($column[$c][$zl]=~/^[a-z]/ || $column[$c][$zl]=~/^\'./){
    return($zl);
};
    };
    return 0;
}


##################################################
##             &substitute                     #
##################################################

sub substitute{
    @array=();
    my ($substituting,$substituted,@array) = @_;

    for ($q=0;$array[$q];$q++) {
if($array[$q] eq $substituted){
    $array[$q]=$substituting;
}
    }
    return @array;
}

##################################################
##             &string_substitute              #
##################################################

sub string_substitute{
    @array=();
    my ($substituting,$substituted,@array) = @_;
    for ($q=0;$array[$q];$q++) {
if($array[$q] =~  $substituted){
    $array[$q] =~ s/$substituted/$substituting/g;
##     print "$q $substituted $substituting $array[$q]";
}
    }
    return @array;
}



##################################################
##            &element_string                    #
##################################################

sub element_strings{
    @array=();
    my ($item, @array) = @_;

    $element_flag = 0;
    foreach $element (@array){
if($item eq $element){
    $element_flag=1;
}
    }

    return $element_flag;
}


##################################################
##            &substring_in_array                #
##################################################

sub substring_in_array($item,@array) {
    @array=();
    my ($item, @array) = @_;

    $element_flag = 0;
    foreach $element (@array){
if($element =~ $item){
    $element_flag=1;
}
    }

    return $element_flag;
}
```

D. The BUP parser (generated code)

```
% Rule1a                stemPrefixed  -->  part stemPre2fixed

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(stemPre2fixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%------------------------------------------------------------------------------------------

% Rule1b                stemPrefixed  -->  part  stemPre1fixed

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(stemPre1fixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),
not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule1c              stemPrefixed  -->  part stemPre1Zufixed

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(stemPre1Zufixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,_,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,infinitive_flex,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule1d             stemPrefixed    -->  pre2  stemPre1Zufixed

pre2(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(stemPre1Zufixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,_,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),
stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,infinitive_flex,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%------------------------------------------------------------------------------------------

% Rule2              stemPre1Zufixed -->  infin[zu]  stemPre1fixed

infin(G,X,Y,

[zu_infin,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPre1Zufixed,G),
goal(stemPre1fixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPre1Zufixed),

not(X2_LBND==lbndFree),

stemPre1Zufixed(G,Z,Y,
[_,stemPre1Zufixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%------------------------------------------------------------------------------------------

% Rule3a             stemPrefixed       -->  part  rkernel

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(rkernel,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%----------------------------------------------------------------------------------------

% Rule3b             stemPrefixed    -->  part  zuRkernel

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(zuRkernel,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,_,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,infinitive_flex,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%----------------------------------------------------------------------------------------

% Rule3c             stemPrefixed    -->  pre2  zuRkernel

pre2(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(zuRkernel,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,_,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,infinitive_flex,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule3e              zuRkernel  -->  infin[zu]  rkernel

infin(G,X,Y,

[zu_infin,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(zuRkernel,G),
goal(rkernel,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',zuRkernel),

not(X2_LBND==lbndFree),

zuRkernel(G,Z,Y,
[_,zuRkernel,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule3f               stemPrefixed  -->  pre2  stemPre1fixed

pre2(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-
```

```
link(stemPrefixed,G),
goal(stemPre1fixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule3g             stemPrefixed --> stemPre2fixed

stemPre2fixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),

stemPrefixed(G,X,Y,
[_,stemPrefixed,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule3h               stemPrefixed --> stemPre1fixed

stemPre1fixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
stemPrefixed(G,X,Y,
[_,stemPrefixed,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule4              stemPre2fixed       -->  pre2  rkernel

pre2(G,X,Y,

[_,_,_,X1_LBND,_,_,_,X1_BCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPre2fixed,G),
goal(rkernel,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,X1_BCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPre2fixed),

not(X2_LBND==lbndFree),

stemPre2fixed(G,Z,Y,
[_,stemPre2fixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,X1_BCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule5              stemPre1fixed       -->  pre1  rkernel

pre1(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPre1fixed,G),
goal(rkernel,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,'+',X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'|+',stemPre1fixed),

not(X2_LBND==lbndFree),

stemPre1fixed(G,Z,Y,
[_,stemPre1fixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,'+',X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule6            stemPre1fixed --> prenn rkernel

prenn(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPre1fixed,G),
goal(rkernel,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,'-',_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,'-',X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'|+',stemPre1fixed),

not(X2_LBND==lbndFree),

stemPre1fixed(G,Z,Y,
[_,stemPre1fixed,_,X1_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,'-',_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,'-',X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule7a        rkernel --> stemSuffixed

stemSuffixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-
```

```
link(rkernel,G),

rkernel(G,X,Y,
[_,rkernel,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule7b            rkernel --> root

root(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LBND='lbndFree'),

link(rkernel,G),

rkernel(G,X,Y,
[_,rkernel,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule8a             stemSuffixed   -->   stemSuffixed  sufn[comb+]

stemSuffixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),

link(stemSuffixed,G),
goal(sufn,X,Z,
_,
[_,_,'+',_,_,X2_RBND,_,X1_SCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'+',stemSuffixed),
stemSuffixed(G,Z,Y,
[_,stemSuffixed,_,X1_LBND,_,X2_RBND,_,X1_SCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule8b        stemSuffixed   -->  lkernel[nat-] sufn[comb+]

lkernel(G,X,Y,

[_,_,_,X0_LBND,_,X1_RBND,_,_,X1_SCAT,'-',_,_,_,_,_,X1_UMGL,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_,_,X0_PARTSTEM,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),

link(stemSuffixed,G),
goal(sufn,X,Z,
_,
[_,_,'+',_,_,X2_RBND,_,X1_SCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,X1_UMGL,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'+',stemSuffixed),


stemSuffixed(G,Z,Y,
[_,stemSuffixed,_,X0_LBND,_,X2_RBND,_,X1_SCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X0_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule8c        stemSuffixed   -->   stemSuffixed sufnn

stemSuffixed(G,X,Y,

[_,_,_,X0_LBND,_,X1_RBND,_,_,X1_SCAT,'-',_,_,_,_,_,X1_UMGL,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_,_,X0_PARTSTEM,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),

link(stemSuffixed,G),
goal(sufnn,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,X1_SCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
X1_UMGL,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'+',stemSuffixed),


stemSuffixed(G,Z,Y,
[_,stemSuffixed,_,X0_LBND,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X0_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------------

% Rule8d              stemSuffixed  -->  lkernel   suf

lkernel(G,X,Y,

[_,_,_,X0_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,_,X0_PARTSTEM,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),

link(stemSuffixed,G),
goal(suf,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,X1_SCAT,X2_SCAT,X1_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,'-',X2_REG,X2_INTERF,'-',X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,
X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'+',stemSuffixed),


stemSuffixed(G,Z,Y,
[_,stemSuffixed,_,X0_LBND,_,X2_RBND,_,_,X2_SCAT,X1_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,'-',X2_REG,X2_INTERF,'-',X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X0_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------------

% Rule8e        stemSuffixed   -->   stemInterfixed  sufnn

stemInterfixed(G,X,Y,

[_,_,_,X0_LBND,_,X1_RBND,_,_,X1_SCAT,'-',_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,X0_PARTSTEM,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-
```

```
not(X1_RBND==rbndFree),

link(stemSuffixed,G),
goal(sufnn,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,X1_SCAT,X2_SCAT,'-',_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'+',stemSuffixed),


stemSuffixed(G,Z,Y,
[_,stemSuffixed,_,X0_LBND,_,X2_RBND,_,_,X2_SCAT,'-',_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X0_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule8f             stemSuffixed   -->   stemInterfixed  sufn[comb+]

stemInterfixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),

link(stemSuffixed,G),
goal(sufn,X,Z,
_,
[_,_,'+',_,_,X2_RBND,_,X1_SCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'+',stemSuffixed),


stemSuffixed(G,Z,Y,
[_,stemSuffixed,_,X1_LBND,_,X2_RBND,_,X1_SCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule9a              stemInterfixed   -->   lkernel interf

lkernel(G,X,Y,

[_,_,_,X0_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,X1_INTERF,
_,_,_,_,_,_,_,_,X0_PARTSTEM,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),
not(X1_INTERF=='-'),

link(stemInterfixed,G),
goal(interf,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,X1_SCAT,X2_SCAT,X1_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'+',stemInterfixed),


stemInterfixed(G,Z,Y,
[_,stemInterfixed,_,X0_LBND,_,X2_RBND,_,_,X2_SCAT,X1_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X0_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule10             lkernel  -->  root

root(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),

link(lkernel,G),

lkernel(G,X,Y,
[_,lkernel,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule10b           lkernel   -->   stemPrefixed

stemPrefixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),

link(lkernel,G),

lkernel(G,X,Y,
[_,lkernel,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule11          stem   -->   stemSuffixed

stemSuffixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stem,G),

stem(G,X,Y,
[_,stem,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule11b          stem   -->   stemPrefixed

stemPrefixed(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stem,G),
stem(G,X,Y,
[_,stem,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule11c          stem   -->   root

root(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stem,G),

stem(G,X,Y,
[_,stem,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule11d          stem   -->   stemCompound

stemCompound(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stem,G),

stem(G,X,Y,
[_,stem,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,
X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,
X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%---------------------------------------------------------------------------------

% Rule12a         stemSuffixed[adj]  -->  word[participle]

word(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,X1_SCAT,_,_,_,_,_,_,X1_UMGL,X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,
X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,_,_,X1_FLEX,_,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

X1_SCAT==v,
sublistmember("non-finit-part",X1_FLEX),

link(stemSuffixed,G),

stemSuffixed(G,X,Y,
[_,stemSuffixed,_,X1_LBND,_,rbndOpt,_,_,a,_,_,_,_,linkOpt,['er_lm_adv'],X1_UMGL,
X1_UMLB,_,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,_,
'ADJECTIVAL',adjective_flex,conversed_participle,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%---------------------------------------------------------------------------------

% Rule13a        past_stem --> stem[v,PRET=+,REG=-]

stem(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,v,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
_,_,_,_,_,'+','-',X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

member(X1_VROOT,['BLIEB','HIE"S','KAM','K"AM','SCHOB','SCH"OB','GING','FIEL','FAND',
'F"AND','SAH','S"AH','FUHR','F"UHR','NAHM','N"AHM','WURD','W"URD','RANN','DARF']),

link(past_stem,G),

past_stem(G,X,Y,
[_,past_stem,_,X1_LBND,_,X1_RBND,_,X1_BCAT,v,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,'+','-',X1_INTERF,X1_INTERF_ARG,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%---------------------------------------------------------------------------------

% Rule13a          past_stem   -->   stem[v,REG+,PRET=-] past
```

```
stem(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,v,X1_NAT,_,_,_,_,_,X1_UMGL,X1_UMLB,_,_,_,_,_,'-',X1_REG,
_,_,_,_,_,'FRAG',X1_STEMCLASS,X1_SUFFIXCLASS,_,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_RBND==rbndFree),
not(X1_RBND==rbndFree),

link(past_stem,G),
goal(past,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,X2_FLEX,
_,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#+',past_stem),


past_stem(G,Z,Y,
[_,past_stem,_,X1_LBND,_,X2_RBND,_,_,v,X1_NAT,_,_,_,_,_,X1_UMGL,X1_UMLB,_,_,_,_,
_,'+',X1_REG,_,_,_,_,_,'FRAG',X1_STEMCLASS,X1_SUFFIXCLASS,X2_FLEX,X1_PARTSTEM,X1_PSTEMS,
X1_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-------------------------------------------------------------------------------------

% Rule14a          word        -->  stem      inflection

stem(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
_,_,_,_,_,'-',X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
_,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LBND=='lbndToBind'),
not(X1_RBND=='rbndFree'),

link(word,G),
goal(inflection,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,_,X1_SCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,X1_REG,_,_,_,_,_,X1_VROOT,
_,X1_SUFFIXCLASS,X2_FLEX,X1_PARTSTEM,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#+',word),

not(X2_RBND==rbndToBind),

word(G,Z,Y,
[_,word,_,X1_LBND,_,X2_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,_,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X2_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,
PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule14b             word   -->   past_stem   v_infl

past_stem(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
_,_,_,_,_,'+',X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
_,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LBND=='lbndToBind'),

link(word,G),
goal(v_infl,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,_,X1_SCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,X1_VROOT,
_,X1_SUFFIXCLASS,X2_FLEX,_,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#+',word),

not(X2_RBND==rbndToBind),

word(G,Z,Y,
[_,word,_,X1_LBND,_,X2_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,'+',X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X2_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule15a           stemCompound         -->  stem    stem

stem(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,_,_,_,_,_,X1_LINK,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LINK==linkObl),
not(X1_RBND=='rbndFree'),

link(stemCompound,G),
goal(stem,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#',stemCompound),
not(X2_LBND=='lbndFree'),

stemCompound(G,Z,Y,
[_,stemCompound,_,X1_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%---------------------------------------------------------------------------------------

% Rule15b          stemCompound   -->   stemLm   stem

stemLm(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemCompound,G),
goal(stem,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#',stemCompound),

not(X2_LBND=='lbndFree'),

stemCompound(G,Z,Y,
[_,stemCompound,_,X1_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%---------------------------------------------------------------------------------------

% Rule16a                 stemLm    -->   stem   lm

stem(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,X1_SCAT,_,_,_,_,X1_LINK,X1_LINK_MID,_,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,_,X1_PARTSTEM,X1_PSTEMS,
X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LINK==linkImp),

link(stemLm,G),
goal(lm,X,Z,
_,
[X2_MID,_,_,_,_,X2_RBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,X2_FLEX,
_,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#+',stemCompound),

inst_member(X2_MID, X1_LINK_MID),

stemLm(G,Z,Y,
[_,_,_,X1_LBND,_,X2_RBND,_,_,X1_SCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,X1_REG,_,
_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X2_FLEX,X1_PARTSTEM,X1_PSTEMS,
X1_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule16b                 stemLm[NA]   -->   stem[adj]  lm[en_lm_a]

stem(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,'a',_,_,_,_,X1_LINK,_,_,_,_,_,_,_,_,X1_PRET,X1_REG,_,_,X1_GEN,
_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,_,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LINK==linkImp),

link(stemLm,G),
goal(lm,X,Z,
_,
[X2_MID,_,_,_,_,X2_RBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,X2_FLEX,
_,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#+',stemLm),

X2_MID=='en_lm_a',

stemLm(G,Z,Y,
[_,stemLm,_,X1_LBND,_,X2_RBND,_,_,'a',_,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,X1_REG,
_,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X2_FLEX,X1_PARTSTEM,X1_PSTEMS,
X1_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule17           word  -->  stem[rbndOpt/rbndFree]

stem(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-
```

```
not(X1_RBND==rbndToBind),
not(X1_LBND==lbndToBind),

link(word,G),

word(G,X,Y,
[_,word,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule18a             suf   -->   sufn

sufn(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,'+',_,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,
_,_,X1_UMLD,_,_,X1_ABLD,_,_,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(suf,G),

suf(G,X,Y,
[_,suf,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,'+',_,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,
_,_,X1_UMLD,_,_,X1_ABLD,_,_,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,_,_,_],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule18b             suf   -->   sufnn

sufnn(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,'-',_,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,
_,_,X1_UMLD,_,_,X1_ABLD,_,_,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(suf,G),

suf(G,X,Y,
[_,suf,_,X1_LBND,_,X1_RBND,_,X1_BCAT,X1_SCAT,'-',_,X1_STR,X1_ADJ,X1_LINK,X1_LINK_MID,
_,_,X1_UMLD,_,_,X1_ABLD,_,_,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,
X1_SUFFIXCLASS,X1_FLEX,_,_,_],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule19a          inflection  -->  v_infl

v_infl(G,X,Y,

[X1_MID,_,_,X1_LBND,_,X1_RBND,_,_,v,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,X1_REG,
_,_,_,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(inflection,G),

inflection(G,X,Y,
[X1_MID,inflection,_,X1_LBND,_,X1_RBND,_,_,v,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,_,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,_,_,_],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule19b          inflection  -->  a_infl

a_infl(G,X,Y,

[X1_MID,_,_,X1_LBND,_,X1_RBND,_,_,a,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,X1_REG,
_,_,_,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(inflection,G),

inflection(G,X,Y,
[X1_MID,inflection,_,X1_LBND,_,X1_RBND,_,_,a,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,_,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,_,_,_],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule19c          inflection  -->  n_infl

n_infl(G,X,Y,
[X1_MID,_,_,X1_LBND,_,X1_RBND,_,_,n,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,X1_REG,
_,_,_,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(inflection,G),

inflection(G,X,Y,
[X1_MID,inflection,_,X1_LBND,_,X1_RBND,_,_,n,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,_,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,X1_FLEX,_,_,_],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule20        stemCompound        -->   stemHyphened   stem

stemHyphened(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,X1_LINK,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LINK==linkObl),

link(stemCompound,G),
goal(stem,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,
X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#',stemCompound),


stemCompound(G,Z,Y,
[_,stemCompound,_,X1_LBND,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule21a        stemHyphened   -->   stem   hyphen

stem(G,X,Y,

[_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemHyphened,G),
goal(hyphen,X,Z,
_,
[_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#',stemHyphened),


stemHyphened(G,Z,Y,
[_,stemHyphened,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%---------------------------------------------------------------------------------------

% Rule2b        stemHyphened   -->   stemLm   hyphen

stemLm(G,X,Y,

[_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemHyphened,G),
goal(hyphen,X,Z,
_,
[_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#',stemHyphened),


stemHyphened(G,Z,Y,
[_,stemHyphened,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%---------------------------------------------------------------------------------------

% Rule22        stem[n] --> word[v,infinitive]

word(G,X,Y,

[X1_MID,_,_,X1_LBND,_,X1_RBND,_,_,_,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,X1_REG,
_,_,_,_,_,X1_VROOT,_,_,X1_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

X1_RBND==rbndFree,
prefix_of_atom("non-finit-inf",X1_FLEX),
link(stem,G),

stem(G,X,Y,
[X1_MID,stem,_,X1_LBND,_,rbndOpt,_,_,n,X1_NAT,_,_,_,linkOpt,['s_lm_n'],_,_,_,_,
_,_,_,X1_PRET,X1_REG,_,_,'n',_,_,X1_VROOT,_,'Nomen_Treffen','akk,sg;dat,sg;nom,
sg',_,_,_],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule23a        word[n] --> word[adj]

word(G,X,Y,

[X1_MID,_,_,X1_LBND,_,rbndFree,_,_,X1_SCAT,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,_,_,_,X1_VROOT,_,_,X0_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

X1_SCAT==a,
not(X1_LBND==lbndToBind),

link(word,G),

word(G,X,Y,
[X1_MID,word,_,X1_LBND,_,rbndFree,_,_,n,X1_NAT,_,_,_,linkOpt,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,_,_,_,X1_VROOT,_,'NA',X0_FLEX,_,_,_],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-------------------------------------------------------------------------------------

% Rule23b        word[n] --> stem[adj,unflekt]

word(G,X,Y,

[X1_MID,_,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,_,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,_,_,_,X1_VROOT,_,_,X0_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

X1_SCAT==a,
not(X1_RBND=rbndFree),
not(X1_LBND==lbndToBind),
sublistmember("unflekt",X0_FLEX),

link(word,G),

word(G,X,Y,
[X1_MID,word,_,X1_LBND,_,rbndFree,_,_,n,X1_NAT,_,_,_,linkOpt,_,_,_,_,_,_,_,X1_PRET,
X1_REG,_,_,_,_,_,X1_VROOT,_,'NA',X0_FLEX,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule24a            stemPrefixed       -->  part  root[CCAT==v,SCAT==n]

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,X1_BCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(root,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X1_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

X2_SCAT==n,
X1_BCAT==v,
not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X1_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule24b            stemPrefixed       -->  part  stemPre1fixed [CCAT==v,SCAT==n]

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,X1_BCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPrefixed,G),
goal(stemPre1fixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X1_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemPrefixed),

X2_SCAT==n,
X1_BCAT==v,
not(X2_LBND==lbndFree),

stemPrefixed(G,Z,Y,
[_,stemPrefixed,_,X1_LBND,_,X2_RBND,_,X1_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule24c             stemPre1fixed        -->   pre1  root[CCAT==v,SCAT==n]

pre1(G,X,Y,

[_,_,_,X1_LBND,_,_,_,X1_BCAT,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemPre1fixed,G),
goal(root,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X1_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'|+',stemPre1fixed),

X2_SCAT==n,
X1_BCAT==v,
not(X2_LBND==lbndFree),

stemPre1fixed(G,Z,Y,
[_,stemPre1fixed,_,X1_LBND,_,X2_RBND,_,X1_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,X2_VROOT,
X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule24d             stemPre1fixed        -->   pre1[ver|be]  stemPrefixed[CCAT==v,
SCAT==n]

pre1(G,X,Y,

[X1_MID,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

member(X0_SUFFIXCLASS,['SAGEN','WARTEN','FASSEN','BEGEISTERN']),
inst_member(X1_MID,[be_pre1_v_v,ver_pre1_v_v]),

link(stemPre1fixed,G),
goal(stemPrefixed,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,X2_BCAT,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,_,_,X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,_,_,_,_,_,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'|+',stemPre1fixed),

X2_SCAT==n,
X2_BCAT==v,
not(X2_LBND==lbndFree),

stemPre1fixed(G,Z,Y,
[_,stemPre1fixed,_,X1_LBND,_,X2_RBND,_,_,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,
X2_UMLB,_,_,_,_,_,'-','+',X2_INTERF,X2_INTERF_ARG,X2_GEN,_,_,'FRAG','FRAGEN',X0_SUFFIXCLASS,
_,'+',X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%-----------------------------------------------------------------------------------------

% Rule25          word   -->   stemInflected

stemInflected(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,X1_UMLB,
_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

X1_RBND==rbndFree,
not(X1_LBND==lbndToBind),

link(word,G),

word(G,X,Y,
[_,word,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,X1_NAT,_,_,_,X1_LINK,X1_LINK_MID,X1_UMGL,
X1_UMLB,_,_,_,_,_,X1_PRET,X1_REG,X1_INTERF,_,X1_GEN,_,_,X1_VROOT,X1_STEMCLASS,X1_SUFFIXCLASS,
X1_FLEX,X1_PARTSTEM,X1_PSTEMS,X1_OSTEMS],

FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut).

%-----------------------------------------------------------------------------------------

% Rule26            word           -->   part  stemInflected

part(G,X,Y,

[_,_,_,X1_LBND,_,_,_,v,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,
PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(ord,G),
goal(stemInflected,X,Z,
_,
[_,_,_,_,_,rbndFree,_,_,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,_,_,
_,_,_,X2_PRET,X2_REG,X2_INTERF,_,_,_,_,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,X2_FLEX,
X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',word),


ord(G,Z,Y,
[_,word,_,X1_LBND,_,rbndFree,_,_,v,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,_,_,_,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,
X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%----------------------------------------------------------------------------------------

% Rule27            stemCompound     -->   stem[num]   stemUndNum

stem(G,X,Y,

[_,_,_,X1_LBND,_,X1_RBND,_,_,X1_SCAT,_,_,_,_,X1_LINK,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

not(X1_LINK==linkObl),
not(X1_RBND=='rbndFree'),
X1_SCAT=='num',

link(stemCompound,G),
goal(stemUndNum,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,
X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemCompound),

inst_member(X2_SCAT,[a,num]),

stemCompound(G,Z,Y,
[_,stemCompound,_,X1_LBND,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%----------------------------------------------------------------------------------------
```

```
% Rule28     stemUndNum   -->   word[und]   stem[num/a]

word(G,X,Y,

[X1_MID,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-

X1_MID=='und',

link(stemUndNum,G),
goal(stem,X,Z,
_,
[_,_,_,X2_LBND,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,
X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
right_bracket,'#',stemUndNum),

not(X2_LBND=='lbndFree'),
inst_member(X2_SCAT,[a,num]),

stemUndNum(G,Z,Y,
[_,stemUndNum,_,X1_LBND,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%---------------------------------------------------------------------------------

% Rule29            stemCompound        -->   wordHyphened   stem

wordHyphened(G,X,Y,

[_,_,_,X1_LBND,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(stemCompound,G),
goal(stem,X,Z,
_,
[_,_,_,_,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,X2_UMGL,X2_UMLB,
_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,X2_SUFFIXCLASS,
X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#',stemCompound),


stemCompound(G,Z,Y,
[_,stemCompound,_,X1_LBND,_,X2_RBND,_,_,X2_SCAT,X2_NAT,_,_,_,X2_LINK,X2_LINK_MID,
X2_UMGL,X2_UMLB,_,_,_,_,_,X2_PRET,X2_REG,X2_INTERF,_,X2_GEN,_,_,X2_VROOT,X2_STEMCLASS,
X2_SUFFIXCLASS,X2_FLEX,X2_PARTSTEM,X2_PSTEMS,X2_OSTEMS],
FsOut,
PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%--------------------------------------------------------------------------------------------

% Rule30a             wordHyphened    -->    word        hyphen

word(G,X,Y,

[_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
FsOut,

PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,OrthStructIn,
OrthStructOut):-


link(wordHyphened,G),
goal(hyphen,X,Z,
_,
[_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_],
PhonSegIn,PhonSegZ,OrthSegIn,OrthSegZ,PhonStructIn,PhonStructZ,OrthStructIn,OrthStructZ,
left_bracket,'#','wordHyphened'),


wordHyphened(G,Z,Y,
[_,'wordHyphened',_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,_,
_,_,_],
FsOut,

PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,PhonStructOut,OrthStructZ,OrthStructOut).


%--------------------------------------------------------------------------------------------

goal(G,X,Z,FsIn,FsOut,PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,
OrthStructIn,OrthStructOut,left_bracket,BOUNDARY,X0_MCAT):-

dict(MCAT,X,Y,_,_,FsIn,PhonSeg,OrthSeg,PhonStruct,OrthStruct),

constrSeg(PhonSegIn,PhonSeg,PhonSegZ,BOUNDARY),
constrSeg(OrthSegIn,OrthSeg,OrthSegZ,BOUNDARY),
constrStruct(PhonStructIn,PhonStruct,PhonStructZ,left_bracket,X0_MCAT,MCAT),
constrStruct(OrthStructIn,OrthStruct,OrthStructZ,left_bracket,X0_MCAT,MCAT),

P =.. [MCAT,G,Y,Z,FsIn,FsOut,PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructZ,
PhonStructOut,OrthStructZ,OrthStructOut],

call(P).


goal(G,X,Z,FsIn,FsOut,PhonSegIn,PhonSegOut,OrthSegIn,OrthSegOut,PhonStructIn,PhonStructOut,
OrthStructIn,OrthStructOut,right_bracket,BOUNDARY,X0_MCAT):-

dict(MCAT,X,Y,_,_,FsIn,PhonSeg,OrthSeg,PhonStruct,OrthStruct),

constrStruct([],PhonStruct,PhonStructMCAT,_,_,MCAT),
constrStruct([],OrthStruct,OrthStructMCAT,_,_,MCAT),

P =.. [MCAT,G,Y,Z,FsIn,FsOut,PhonSegZ,PhonSegOut,OrthSegZ,OrthSegOut,PhonStructMCAT,
PhonStructZ,OrthStructMCAT,OrthStructZ],


constrSeg(PhonSegIn,PhonSeg,PhonSegZ,BOUNDARY),
constrSeg(OrthSegIn,OrthSeg,OrthSegZ,BOUNDARY),

call(P),

constrStruct(PhonStructIn,PhonStructZ,PhonStructOut,right_bracket,X0_MCAT,MCAT),
constrStruct(OrthStructIn,OrthStructZ,OrthStructOut,right_bracket,X0_MCAT,MCAT).

a_infl(a_infl,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
hyphen(hyphen,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
infin(infin,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
inflection(inflection,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
interf(interf,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
lkernel(lkernel,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
lm(lm,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
n_infl(n_infl,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
ord(ord,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
part(part,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
past(past,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
past_stem(past_stem,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
pre1(pre1,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
pre2(pre2,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
prenn(prenn,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
rkernel(rkernel,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
root(root,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
stem(stem,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
stemCompound(stemCompound,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
stemHyphened(stemHyphened,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
stemInflected(stemInflected,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,
PhonStruct,OrthStruct,OrthStruct).
stemInterfixed(stemInterfixed,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,
PhonStruct,OrthStruct,OrthStruct).
stemLm(stemLm,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
stemPre1Zufixed(stemPre1Zufixed,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,
PhonStruct,OrthStruct,OrthStruct).
stemPre1fixed(stemPre1fixed,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,
PhonStruct,OrthStruct,OrthStruct).
stemPre2fixed(stemPre2fixed,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,
PhonStruct,OrthStruct,OrthStruct).
stemPrefixed(stemPrefixed,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
stemSuffixed(stemSuffixed,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
```

```
stemUndNum(stemUndNum,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
suf(suf,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
sufn(sufn,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
sufnn(sufnn,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
v_infl(v_infl,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
word(word,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,OrthStruct,
OrthStruct).
wordHyphened(wordHyphened,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
zuRkernel(zuRkernel,X,X,F,F,PhonSeg,PhonSeg,OrthSeg,OrthSeg,PhonStruct,PhonStruct,
OrthStruct,OrthStruct).
```

E. Morph database sample

```
$1_MID $2_MCAT $3_PHON $4_ORTH $5_COMB_SUF $6_LBND $7_LBND_ARG $8_RBND $9_RBND_ARG,f \
$10_BCAT $11_SCAT $12_NAT $13_SEP $14_STR $15_ADJ $16_LINK $17_LINK_MID $18_UMGL \
$19_UMLB $20_UMLD $21_ABGL $22_ABLB $23_ABLD $24_ABL_SUF_MID $25_PRET $26_REG $27_INTERF \
$28_INTERF_ARG $29_GEN $30_HEAD_MID $31_ARG_MID $32_VROOT $33_STEMCLASS $34_SUFFIXCLASS \
$35_FLEX $36_PSEG $37_OSEG $38_STEMS_ORTH $39_STEMS_PHON $40_BEISPIEL $41_ORIGIN
"a"s_root_v root ?'E:s "a"s ~ lbndOpt d,c rbndToBind f ~ v nat+ ~ str+ ~ linkImp \
~ umgl+ umlb+ ~ ~ ~ ~ ~ pret- reg- interf- ~ ~ ~ ~ N"AHM NEHMEN ESSEN praes-imp-sg-2 \
?'E:s "a"s e"s,i"s,a"s,"a"s,ge"s ?'Es,?'Is,?'a:s,?'E:s,g'Es "a"se generated
"ah_root_interj root ?'E: "ah ~ lbndFree ~ rbndFree ~ ~ interj nat+ ~ str+ ~ linkImp \
~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ ~ ~ ?'E: "ah ~ ~ "ah tuebcd6-13
"ahn_root_n root ?'E:n "ahn ~ lbndOpt d,c rbndToBind d ~ n nat+ ~ str+ ~ linkImp \
~ umgl+ umlb+ ~ ~ ~ ~ ~ ~ ~ interf- ~ n ~ ~ ~ ~ Nomen_Mensch ~ ?'E:n "ahn ~ ~ ~ \
eval96
"ander_root_v root ?'End@r "ander ~ lbndOpt d,c rbndToBind d,c,f ~ v nat+ ~ str+ \
~ linkImp ~ umgl+ umlb+ ~ ~ ~ ~ ~ pret- reg+ interf- ~ ~ ~ ~ FRAG FRAGEN BEGEISTERN \
praes-imp-sg-2 ?'End@r "ander "ander ?'End@r ~ eval96
"appel_root_v root ?'Ep@l "appel ~ lbndToBind d rbndOpt d,c,f n v nat+ ~ str+ ~ \
linkImp ~ ~ umlb- ~ ~ ~ ~ ~ pret- reg+ interf- ~ ~ ~ ~ FRAG FRAGEN BEGEISTERN praes-imp-sg-2 \
?'Ep@l "appel "appel ?'Ep@l ver"appeln synthese
"arger_root_n root ?'Erg@r "arger ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ str+ ~ \
linkImp ~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ m ~ ~ ~ ~ Nomen_Fehler akk,pl;akk,sg;dat,sg;gen,pl;nom,pl;nom,sg \
?'Erg@r "arger ~ ~ ~ eval96
"arger_root_v root ?'Erg@r "arger ~ lbndOpt d,c rbndOpt d,c,f n v nat+ ~ str+ ~ \
linkImp ~ ~ umlb- ~ ~ ~ ~ ~ pret- reg+ interf- ~ ~ ~ ~ FRAG FRAGEN BEGEISTERN praes-imp-sg-2 \
?'Erg@r "arger "arger ?'Erg@r ~ eval96
"arzt_root_n root ?'E6tst "arzt ~ lbndOpt d,c rbndToBind d,f ~ n nat+ ~ str+ ~ \
linkObl e_lm_n umgl+ umlb+ ~ ~ ~ ~ ~ ~ ~ interf- ~ m ~ ~ ~ ~ Nomen_Fall ~ ?'Ertst \
"arzt ~ ~ "Arzteschaft tuebcd6-13
"ather_root_n root ?'Et@r "ather ~ lbndOpt d,c rbndOpt d,c,f ~ n nat- ~ str+ ~ \
linkImp ~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ m ~ ~ ~ ~ Nomen_Fehler akk,pl;akk,sg;dat,sg;gen,pl;nom,pl;nom,sg \
?'Et@r "ather ~ ~ "Ather cd04
"au"ser_root_v root ?'OYs@r "au"ser ~ lbndOpt d,c rbndToBind d,c,f ~ v nat+ ~ str+ \
~ linkImp ~ umgl+ umlb+ ~ ~ ~ ~ ~ pret- reg+ interf- ~ ~ ~ ~ FRAG FRAGEN BEGEISTERN \
praes-imp-sg-2 ?'OYs+@r "aus+er "aus+er ?'OYs+@r ~ eval96
"odi_root_n root ?'2:dj "odi ~ lbndToBind d rbndToBind d,f ~ n nat- ~ str+ ~ linkOpt \
en_lm_n ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ f ~ ~ ~ ~ Nomen_Famili-e ~ ?'2:dj "odi \
~ ~ Kom"odie hh98
"odip_root_n root ?'2:dIp "odip ~ lbndOpt d,c rbndToBind d ~ n nat- ~ str+ ~ linkImp \
~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ m ~ ~ ~ ~ ~ ~ ?'2:dIp "odip ~ ~ "Odipus cd05
"offen_root_a root ?'9f@n "offen ~ lbndOpt d,c rbndToBind d,f ~ a nat+ ~ str+ ~ \
linkOpt er_lm_adv umgl+ umlb+ ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ ADJECTIVAL unflekt \
?'9f@n "offen ~ ~ ~ eval96
"offn_root_v root ?'9fn "offn ~ lbndOpt d,c rbndToBind d,f a v nat+ ~ str+ ~ linkImp \
~ umgl+ umlb+ ~ ~ ~ ~ ~ pret- reg+ interf- ~ ~ ~ ~ FRAG FRAGEN WARTEN ~ ?'9fn "offn \
"offn ?'9fn ~ eval96
"ogai_root_n root ?'2:gaI "ogai ~ lbndOpt c rbndOpt d,c,f ~ n nat+ ~ str+ ~ linkImp \
~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ f ~ ~ ~ ~ Nomen_Kamera akk,sg;dat,sg;gen,sg;nom,sg \
?'2:gaI "ogai ~ ~ ~ eval96
"oko_root_a root ?'2:ko: "oko ~ lbndOpt d,c rbndToBind d,c,f ~ a nat- ~ str+ ~ \
linkOpt er_lm_adv ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ ADJECTIVAL unflekt \
?'2:ko: "oko ~ ~ "okologisch cd12
"ol_root_n root ?'2:l "ol ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ str+ ~ linkImp \
~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ n ~ ~ ~ ~ Nomen_Abend akk,sg;dat,sg;nom,sg ?'2:l \
"ol ~ ~ ~ eval96
"ol_root_n root ?'2:l "ol ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ str+ ~ linkImp \
~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ n ~ ~ ~ ~ Nomen_Jahr akk,sg;nom,sg ?'2:l "ol \
~ ~ ~ eval96
"ort_root_n root ?'9rt "ort ~ lbndOpt d,c rbndToBind d,f ~ n nat+ ~ str+ ~ linkImp \
~ umgl+ umlb+ ~ ~ ~ ~ ~ ~ ~ interf- ~ m ~ ~ ~ ~ Nomen_Gott ~ ?'9rt "ort ~ ~ "Orter,"ortlich \
cd04
"ost_root_n root ?'9st "ost ~ lbndOpt d,c rbndToBind d ~ n nat+ ~ str+ ~ linkImp \
~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ m ~ ~ ~ ~ ~ ~ ?'9st "ost ~ ~ "ostlich tuebcd6-13
"ub_root_v root ?'y:b "ub ~ lbndOpt d,c rbndToBind d,c,f ~ v nat+ ~ str+ ~ linkImp \
~ ~ umlb- ~ ~ ~ ~ ~ pret- reg+ interf- ~ ~ ~ ~ FRAG FRAGEN SAGEN praes-imp-sg-2 \
?'y:b "ub "ub ?'y:b ~ eval96
"ubel_root_a root ?'y:b@l "ubel ~ lbndOpt d,c rbndOpt d,c,f ~ a nat+ ~ str+ ~ linkOpt \
er_lm_adv ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ ADJECTIVAL unflekt ?'y:b@l \
"ubel ~ ~ ~ eval96
"ubr_root_a root ?'y:br "ubr ~ lbndOpt d,c rbndToBind d,f ~ a nat+ ~ str+ ~ linkOpt \
er_lm_adv ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ ADJECTIVAL unflekt ?'y:br "ubr \
~ ~ ~ eval96
$"A_root_n root ?'E: $"A ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ \
umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'E: $"A \
~ ~ $A-$B-$C ADDED_HL
$"O_root_n root ?'2: $"O ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ \
umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'2: $"O \
~ ~ $A-$B-$C ADDED_HL
$"S_root_n root ?Ests'Et $"S ~ lbndOpt c rbndOpt cf ~ n nat+ ~ str+ ~ linkImp ~ \
~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?Ests'Et \
$"S ~ ~ $A-$B-$C ADDED_HL
$"U_root_n root ?'y: $"U ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ \
umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'y: $"U \
~ ~ $A-$B-$C ADDED_HL
$A_root_n root ?'a: $A ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'a: $A ~ ~ $A-$B-$C \
ADDED_HL
$B_root_n root b'e: $B ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg b'e: $B ~ ~ $A-$B-$C \
ADDED_HL
$C_root_n root ts'e: $C ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ts'e: $C ~ ~ \
$A-$B-$C ADDED_HL
$D_root_n root d'e: $D ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg d'e: $D ~ ~ $A-$B-$C \
ADDED_HL
$E_root_n root ?'e: $E ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'e: $E ~ ~ $A-$B-$C \
ADDED_HL
$F_root_n root ?'Ef $F ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'Ef $F ~ ~ $A-$B-$C \
ADDED_HL
$G_root_n root g'e: $G ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg g'e: $G ~ ~ $A-$B-$C \
ADDED_HL
$H_root_n root h'a: $H ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg h'a: $H ~ ~ $A-$B-$C \
ADDED_HL
$I_root_n root ?'i: $I ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'i: $I ~ ~ $A-$B-$C \
ADDED_HL
$J_root_n root j'Ot $J ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg j'Ot $J ~ ~ $A-$B-$C \
ADDED_HL
$K_root_n root k'a: $K ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg k'a: $K ~ ~ $A-$B-$C \
ADDED_HL
$L_root_n root ?'El $L ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'El $L ~ ~ $A-$B-$C \
ADDED_HL
$M_root_n root ?'Em $M ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'Em $M ~ ~ $A-$B-$C \
ADDED_HL
```

```
$N_root_n root ?'En $N ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'En $N ~ ~ $A-$B-$C \
ADDED_HL
$O_root_n root ?'o: $O ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'o: $O ~ ~ $A-$B-$C \
ADDED_HL
$P_root_n root p'e: $P ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg p'e: $P ~ ~ $A-$B-$C \
ADDED_HL
$Q_root_n root k'u: $Q ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg k'u: $Q ~ ~ $A-$B-$C \
ADDED_HL
$R_root_n root ?'Er $R ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'Er $R ~ ~ $A-$B-$C \
ADDED_HL
$SZ_root_n root ?Ests'Et $SZ ~ lbndOpt c rbndOpt cf ~ n nat+ ~ str+ ~ linkImp ~ \
~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?Ests'Et \
$SZ ~ ~ $A-$B-$C ADDED_HL
$S_root_n root ?'Es $S ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Jahr akk,sg;nom,sg ?'Es $S ~ ~ $A-$B-$C \
ADDED_HL
$S_root_n root ?'Es $S ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Abend akk,sg;dat,sg;nom,sg ?'Es $S ~ ~ \
$A-$B-$C ADDED_HL
$S_root_n root ?'Es $S ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Jahr akk,sg;nom,sg ?'Es $S ~ ~ $A-$B-$C \
ADDED_HL
$T_root_n root t'e: $T ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg t'e: $T ~ ~ $A-$B-$C \
ADDED_HL
$U_root_n root ?'u: $U ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'u: $U ~ ~ $A-$B-$C \
ADDED_HL
$V_root_n root f'aU $V ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg f'aU $V ~ ~ $A-$B-$C \
ADDED_HL
$W_root_n root v'e: $W ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg v'e: $W ~ ~ $A-$B-$C \
ADDED_HL
$X_root_n root ?'Iks $X ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Jahr akk,sg;nom,sg ?'Iks $X ~ ~ $A-$B-$C \
ADDED_HL
$X_root_n root ?'Iks $X ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Abend akk,sg;dat,sg;nom,sg ?'Iks $X ~ ~ \
$A-$B-$C ADDED_HL
$X_root_n root ?'Iks $X ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Jahr akk,sg;nom,sg ?'Iks $X ~ ~ $A-$B-$C \
ADDED_HL
$Y_root_n root ?YpsIlOn $Y ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ \
~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ?'Ypsi:lOn \
$Y ~ ~ $A-$B-$C ADDED_HL
$Z_root_n root ts'Et $Z ~ lbndOpt c rbndOpt c,f ~ n nat+ ~ str+ ~ linkImp ~ ~ umlb- \
~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ Nomen_Auto akk,sg;dat,sg;nom,sg ts'Et $Z ~ ~ \
$A-$B-$C ADDED_HL
a"s_root_v root ?'a:s a"s ~ lbndOpt d,c rbndOpt d,c,f ~ v nat+ ~ str+ ~ linkImp \
~ umgl- umlb+ ~ ~ ~ ~ ~ pret+ reg- interf- ~ ~ ~ ~ NAHM NEHMEN ESSEN praes-imp-sg-2 \
?'a:s a"s e"s,i"s,a"s,"a"s,ge"s ?'Es,?'Is,?'a:s,?'E:s,g'Es a"s generated
abend_root_n root ?'a:b@nd abend ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ str+ ~ \
linkImp ~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ n ~ ~ ~ ~ Nomen_Abend akk,sg;dat,sg;nom,sg \
?'a:b@nd abend ~ ~ ~ eval96
abend_root_n root ?'a:b@nd abend ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ str+ ~ \
linkImp ~ ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ n ~ ~ ~ ~ Nomen_Jahr akk,sg;nom,sg ?'a:b@nd \
abend ~ ~ ~ eval96
```

```
absurd_root_a root ?apz'Urd absurd ~ lbndOpt d,c rbndOpt d,c,f ~ a nat- ~ str+ \
~ linkOpt er_lm_adv ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ ~ ~ ~ ~ ~ ADJECTIVAL unflekt \
?apz'Urd absurd ~ ~ absurderweise cd20
abteil_root_n root ?apt'aIl abteil ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ str+ \
~ linkOpt s_lm_n ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ n ~ ~ ~ ~ Nomen_Abend akk,sg;dat,sg;nom,sg \
?ap#t'aIl ab#teil ~ ~ Raucherabteil ADDED_HL
abteil_root_n root ?apt'aIl abteil ~ lbndOpt d,c rbndOpt d,c,f ~ n nat+ ~ str+ \
~ linkOpt s_lm_n ~ umlb- ~ ~ ~ ~ ~ ~ ~ interf- ~ n ~ ~ ~ ~ Nomen_Jahr akk,sg;nom,sg \
?ap#t'aIl ab#teil ~ ~ Raucherabteil ADDED_HL
```

F. Mclass output sample (eval mode)

```
"Ubernachtung "Ub+er#+n#acht+ung ?'y:.b+6#+n#?'ax.t+UN ?y:b6n?axtUN "Ubernachtung \
_1853 Nomen_Frau "Ub+er#+n#acht+ung,n
"uber "uber ?'y:.b6 ?y:b6 "uber _1793 dummy_suffixclass      "uber,prep
"uberall "uber#all ?'y:.b6#?''al ?y:b6?al "uberall _1805 dummy_suffixclass "uber#all,adv
"ubermorgen "uber#morg+en ?'y:.b6#m''O6.g+@n ?y:b6mO6g@n "ubermorgen _1817 dummy_suffixclass \
"uber#morg+en,adv
"ubernachten "ub+er+n#acht#+en ?'y:.b+6+n#?'ax.t#+@n ?y:b6n?axt@n "ubernachten \
FRAGEN WARTEN "ub+er+n#acht#+en,v
$I-$C-$E $I-$C-$E ?'i:#ts'e:#?'e: ?i:tse:?e: $I-$C-$E _1843 Nomen_Auto $I-$C-$E,n
Adresse Adress#+e ?a.dr'E.s#+@ ?adrEs@ Adresse _1817 Nomen_Famili-e Adress#+e,n
Ahnung Ahn+ung ?'a:.n+UN ?a:nUN Ahnung _1811 Nomen_Frau      Ahn+ung,n
Arbeit Arbeit ?'a6.baIt ?a6baIt Arbeit _1811 Nomen_Frau      Arbeit,n
Assistentin Assist+ent+in ?'a.sIs.t+'En.t+In ?asIstEntIn Assistentin _1841 Nomen_Frau \
Assist+ent+in,n
Aufenthalt Auf#ent+halt ?'aUf#?Ent.+h'alt ?aUf?Enthalt Aufenthalt _1835 Nomen_Abend \
Auf#ent+halt,n
Auto Auto ?'aU.to: ?aUto: Auto _1799 Nomen_Auto      Auto,n
Bahn Bahn b'a:n ba:n Bahn _1799 Nomen_Frau      Bahn,n
Bahnhof Bahn#hof b'a:n#h'o:f ba:nho:f Bahnhof _1817 Nomen_Fall Bahn#hof,n
Bar Bar b'a:6 ba:6 Bar _1793 Nomen_Kamera      Bar,n
Beispiel Bei#spiel b'aI#Sp'i:l baISpi:l Beispiel _1823 Nomen_Jahr Bei#spiel,n
Berg Berg b'E6k bE6k Berg _1799 Nomen_Jahr      Berg,n
Berge Berg#+e b'E6.g#+@ bE6g@ Berg _1805 Nomen_Jahr      Berg#+e,n
Buchungsbest"atigung Buch+ung#+s#be+st"at+ig+ung b'u:.x+UN#+s#b@.+St'E:.t+I.g+UN \
bu:xUNsb@StE:tIgUN Buchungsbest"atigung _1895 Nomen_Frau Buch+ung#+s#be+st"at+ig+ung,n
Computer Comput+er kOmp.j'u:.t+6 kOmpju:t6 Computer _1823 Nomen_Fehler Comput+er,n
Computeranlage Comput+er#an#lag#+e kOmp.j'u:.t+6#?'an#l'a:.g#+@ kOmpju:t6?anla:g@ \
Computeranlage _1859 Nomen_Famili-e Comput+er#an#lag#+e,n
D"usseldorf D"us+sel#dorf d'y:s.+z@l#d'O6f dy:sz@ldO6f D"usseldorf _1841 Nomen_Gott \
D"us+sel#dorf,n
Dampfbad Dampf#bad d'ampf#b'a:t dampfba:t Dampfbad _1823 Nomen_Gott Dampf#bad,n
Dank Dank d'aNk daNk Dank _1799 Nomen_No_pl_Kram      Dank,n
Daten Dat#+en d'a:.t#+@n da:t@n Datum _1805 Nomen_Dat-um      Dat#+en,n
Deutschland Deutsch#land d'OYtS#l'ant dOYtSlant Deutschland _1841 Nomen_Gott Deutsch#land,n
Dienstreise Dienst#reis#+e d'i:nst#r'aI.z#+@ di:nstraIz@ Dienstreise _1841 Nomen_Famili-e \
Dienst#reis#+e,n
Doppelzimmer Doppel#zimmer d'O.p@l#ts'I.m6 dOp@ltsIm6 Doppelzimmer _1847 Nomen_Fehler \
Doppel#zimmer,n
Einzelzimmer Einz+el#zimmer ?'aIn.ts+@l#ts'I.m6 ?aInts@ltsIm6 Einzelzimmer _1847 \
Nomen_Fehler Einz+el#zimmer,n
Entschuldigung Ent+schuld+ig+ung ?Ent.+S'Ul.d+I.g+UN ?EntSUldIgUN Entschuldigung \
_1859 Nomen_Frau Ent+schuld+ig+ung,n
Erdgescho"s  no_result
Ersatzsauna Er+satz#saun#+a ?E6.+z'ats#z'aU.n#+a: ?E6zatszaUna: Ersatzsauna _1841 \
Nomen_Mens-a Er+satz#saun#+a,n
Etage Etag#+e ?e:.t'a:.Z#+@ ?e:ta:Z@ Etage _1805 Nomen_Famili-e Etag#+e,n
Fahrkarte Fahr#kart#+e f'a:6#k'a6.t#+@ fa:6ka6t@ Fahrkarte _1829 Nomen_Famili-e \
Fahr#kart#+e,n
Fahrtzeit Fahr+t#zeit f'a:6+t#ts'aIt fa:6ttsaIt Fahrtzeit _1829 Nomen_Frau Fahr+t#zeit,n
Fahrzeit Fahr#zeit f'a:6#ts'aIt fa:6tsaIt Fahrzeit _1823 Nomen_Frau Fahr#zeit,n
Fernsehkanal Fern#seh#kanal f'E6n#z'e:#ka:.n'a:l fE6nze:ka:na:l Fernsehkanal _1847 \
Nomen_Anfang Fern#seh#kanal,n
Flu"s Flu"s fl'Us flUs Flu"s _1805 Nomen_Fall      Flu"s,n
Flug Flug fl'u:k flu:k Flug _1799 Nomen_Fall      Flug,n
Flugzeug Flug#zeug fl'u:k#ts'OYk flu:ktsOYk Flugzeug _1823 Nomen_No_pl_Kram Flug#zeug,n
Frage Frag#+e fr'a:.g#+@ fra:g@ Frage _1805 Nomen_Famili-e      Frag#+e,n
Frau Frau fr'aU fraU Frau _1799 Nomen_Frau      Frau,n
Freund Freund fr'OYnt frOYnt Freund _1811 Nomen_Jahr      Freund,n
Fu"s Fu"s f'u:s fu:s Fu"s _1799 Nomen_Fall      Fu"s,n
```

```
Garage Garag#+e ga.ra:.Z#+@ gara:Z@ Garage _1811 Nomen_Famili-e Garag#+e,n
Gegend Gegend g'e:.g@nt ge:g@nt Gegend _1811 Nomen_Frau      Gegend,n
Gl"uck Gl"uck gl'Yk glYk Gl"uck _1811 Nomen_Jahr      Gl"uck,n
Hamburg Ham#burg h'a:m#b'U6k ha:mbU6k Hamburg _1874 Nomen_Frau HAM#burg,n
Hannover Hannover ha.n'o:.f6 hano:f6 Hannover _1823 Nomen_Auto Hannover,n
Hauptbahnhof Haupt#bahn#hof h'aUpt#b'a:n#h'o:f haUptba:nho:f Hauptbahnhof _1847 \
Nomen_Fall Haupt#bahn#hof,n
Hektik Hekt+ik h'Ek.t+i:k hEkti:k Hektik _1811 Nomen_Frau      Hekt+ik,n
Herr Herr h'E6 hE6 Herr _1799 Nomen_Herr      Herr,n
Hinfahrt Hin#fahr+t h'In#f'a:6+t hInfa:6t Hinfahrt _1823 Nomen_Frau Hin#fahr+t,n
Hotel Hotel ho:.t'El ho:tEl Hotel _1805 Nomen_Auto      Hotel,n
Hotels Hotel#+s ho:.t'El#+s ho:tEls Hotel _1811 Nomen_Auto      Hotel#+s,n
Ihnen ihn#+en ?i:.n#+@n ?i:n@n ihn _2005 dummy_suffixclass      Ihn#+en,pron
Ihr ihr ?i:6 ?i:6 ihr _1997 dummy_suffixclass      Ihr,det
Ihre ihr#+e ?i:.r#+@ ?i:r@ ihr _2001 dummy_suffixclass      Ihr#+e,det
Ihrer ihr#+er ?i:.r#+6 ?i:r6 ihr _2005 dummy_suffixclass      Ihr#+er,det
Information In+form+at+ion ?'In.+f'O6.m+'a:.ts+j'o:n ?InfO6ma:tsjo:n Information \
_1841 Nomen_Frau In+form+at+ion,n
Jansen Jansen j'an.z@n janz@n Jansen _1868 Jahr      JANSEN,n
Japan Japan j'a:.pa:n ja:pa:n Japan _1805 Nomen_Auto      Japan,n
Juni Juni j'u:.ni: ju:ni: Juni _1799 Nomen_Auto      Juni,n
Kasse Kass#+e k'a.s#+@ kas@ Kasse _1805 Nomen_Famili-e      Kass#+e,n
Kind Kind k'Int kInt Kind _1799 Nomen_Kind      Kind,n
```

Gedruckt auf alterungsbeständigem Papier ∞ ISO 9706.