Theorie des Lexikons

Arbeiten des Sonderforschungsbereichs 282

Nr. 62

A Computational Model of Minimalist Morphology

Ray Fabri, Christof Rumpf,
Martina Urbas, Markus Walther

Februar 1995

Projekt: B3/B4

Seminar für Allgemeine Sprachwissenschaft
Universitässtr.1
40225 Düsseldorf

Tel: 0211-311-5280
E-Mail: {fabri,walther}@ ling.uni-duesseldorf.de

ISSN-0939-995X

A Computational Model of Minimalist Morphology

Table of Contents

1. Introduction 1
2. The implementation of linguistic theories..... 2
3. Overview of Minimalist Morphology 3
3.1 Stems and affixes 4
3.2 Principles 6
3.3 Paradigms 8
4. The computational model 10
4.1 General considerations 10
4.2 Building blocks for formalisation 11
4.3 Formalising the universe of discourse 13
4.4 Formalising verb stems..... 16
4.5 Formalising affixes 19
4.6 The Morphogical Combination Operation 21
4.7 Constraining morphological combination 23
4.8 Feature Co-occurrence Restrictions 24
5. Implementing the Paradigm Mechanism 25
5.1 Ranking candidate word forms 27
5.2 Paradigm insertion 28
6. A comparison with Network Morphology..... 31
7. Conclusion 32
References 33

Appendix 1: Paradigm of German verb inflection 36
Appendix 2: Transfer of paradigm information into CG' 37
Appendix 3: The CUF grammar for German verb inflection (CG) 39
Appendix 4: The source code of the Paradigm Mechanism (PM) 46
Appendix 5: Sample session using CG and PM 61

A Computational Model of Minimalist Morphology[1]

Ray Fabri, Christof Rumpf, Martina Urbas, Markus Walther

Abstract

This report describes a co-operative undertaking by a group of theoretical and computational linguists from the University of Düsseldorf, Germany, whose aim was to formalise and implement a computational model of Minimalist Morphology (Wunderlich/Fabri 1994). Minimalist Morphology, a lexicalist theory focusing on inflection, places special emphasis on the economical feature specification of stems and affixes, which combine in a principle-guided fashion. An active notion of recursive paradigms is used to filter overgenerated candidate word forms.The computer implementation of stems, affixes, and morphological combination is realised using the constraint-based linguistic formalism CUF (Dörre/Dorna 1993). Output candidate word forms are then processed by a complex, parametrised paradigm tool, written in PROLOG, that embodies active ranking principles to select the well-formed forms. It is shown that Minimalist Morphology can be made explicit to a sufficient degree to provably generate all and only the correct inflected forms of both regular and irregular verbs in Modern High German. This case study demonstrates the nontrivial potential of computer implementation as an aid in the rigid testing of contemporary linguistic theories, whose subtleties often defy a classical pencil-and-paper approach.

1. Introduction

This paper presents the results of a co-operative project involving theoretical and computational linguists whose aim was to implement a model of inflection based on an approach known as Minimalist Morphology (from now on MM, see Wunderlich 1992 and Wunderlich/Fabri 1994). In particular, the task was to develop a reliable and constructive computational model that could be used to test the weak generative capacity of MM, i.e., a program whose output was the set of inflected (word) forms generated by a particular MM analysis, as well as the features associated with those forms. Moreover, by systematically modelling the relevant representations, principles and conditions involved, the theory of MM itself was to be put to the test.

The fundamental questions raised provided a basis for fruitful discussion, as well as a concrete opportunity to find out to what extent it would be possible for theoretical linguists and computational linguists to work together on a common project and to 'unify' their often varying approaches to linguistic problems. Experience involving similar 'joint ventures' has in fact often shown them to be a delicate matter, with the parties involved sometimes being simply unable to understand each other's problems and points of view. Even in this group, it became

[1] For helpful comments and discussions, we would like to thank Petra Barg, Birgit Gerlach, Jim Kilbury, Albert Ortmann, Carsten Steins and Dieter Wunderlich. This work has been carried out within the Sonderforschungsbereich 282 "Theorie des Lexikons", which is funded by the German Research Foundation DFG. The two projects involved are B4 *Lexikalische Fundierung der Kongruenz* ('The Lexical Basis of Agreement') and B3 *Dynamische Erweiterung des Lexikons* ('The Dynamic Extension of the Lexicon'). Electronic mail correspondence should be directed to {fabri, walther} @ling.uni-duesseldorf.de.

obvious from the beginning that we tended to use certain terms (such as monotonicity) in slightly different senses and we realised that we often simply did not speak the same jargon, even though we all spoke the same language. In time we got over the jargon barrier and could therefore concentrate on the essentials of our task: the implementation of the model of Minimalist Morphology, which we want to describe in some detail in this paper. Key elements in establishing a common basis to overcome the jargon barrier will be mentioned throughout the text.

2. The implementation of linguistic theories

One of the main weaknesses of linguistic theories is that they are often inadequately formalised, which often makes it difficult or even impossible to compute their generative capacity and to test their formal properties. Bird (1991) postulates a notion of *formal adequacy*, which imposes the basic requirement that a linguistic description language should possess an explicit formal syntax and semantics not unlike the mathematical definitions of various logics and computer languages. In order to fulfil the criterion of formal adequacy (as well as descriptive, explanatory, and empirical adequacy, see Chomsky 1957), a theory must be formulated within a formal framework that allows its notions, principles and constraints to be explicitly defined and represented. In the last few years a number of tools have been devised which allow the linguist to formulate a theory within a given computational framework. The advantage of such tools is that they can be effectively run on computers and thus be used to test (grammatical) theories and analyses for internal consistency and coverage. At the same time they also force the theoretical linguist to be absolutely explicit about the notions involved in the particular theory being modelled, thus often exposing possible weaknesses and gaps. Furthermore, claims about certain formal properties such as monotonicity may be proven or disproven once and for all, since they are evaluated for the framework itself rather than on the basis of a particular analysis.

The questions we wanted to find answers to were the following:

1. *Consistency:* How consistent is the MM model formally? If inconsistencies exist, how can we repair them, while still remaining faithful to our original motivations? Thus, for example, it might turn out that certain principles or assumptions may actually be in conflict with each other.

2. *Explicitness:* How explicit is the theory? Are there any hidden assumptions, that is assumptions which are taken for granted but not explicitly stated as such? The linguistically informed theorist often unconsciously relies on such hidden assumptions, which, however, are inaccessible to a blind machine evaluation.

3. *Redundancy:* Are there any redundancies within the system? Could certain principles turn out to be superfluous, or perhaps be reduced to other, more general principles? Is there any overlap between principles?

4. *Formalisation:* How can the assumptions and principles required for MM be expressed within a given formal framework? Which formal framework can be used to adequately model the approach? Is it possible to use a single homogeneous framework, or do different frameworks have to be combined in order to cover all the requirements of the model? How powerful computationally are the formalisms required ?

These questions set the basis for a discussion of the formal and theoretical properties of the MM model which will be described in the following sections.

The paper is organised as follows. In section 3 we provide an overview of the original conception of MM, followed by a discussion in sections 4 and 5 of why we adopted the particular computational framework that was chosen for our implementation, then describing how we have in detail gone about implementing the notions, representations and principles involved. Finally, in section 6 we contrast our approach with that of Corbett/Fraser (1993) and Fraser/Corbett (to appear), who have developed a computational model for the inflectional morphology of Russian known as Network Morphology, before we conclude with section 7.

3. Overview of Minimalist Morphology

The brief description of the MM model given in this section is based on the presentation in Wunderlich/Fabri (1994) and does not take into account any modifications in connection with the computer implementation to be discussed later in this paper. For details and justification of the assumptions described here, the reader should refer to the original text and to Wunderlich (1992).

The term Minimalist Morphology is meant to reflect the fact that the notion of economy plays a prominent role in the design of the model. MM is considered to offer an alternative to other approaches to morphology, which often underestimate the importance of economy, thus suffering from an undesirable proliferation of basic entities such as levels in *Lexical (Morpho-) Phonology* (Kiparsky 1982), rules and rule types in *Distributed Morphology* (Halle/Marantz 1993/94) or arbitrary features in *Deconstructing Morphology* (Lieber 1992). According to Goodman (1943:107): "The motives for seeking economy in the basis of a system are much the same as the motives for constructing the system itself. A given idea *A* need be left as primitive in a system only so long as we have discovered no relationship intimate enough to permit defining *A* in terms of them; hence the more the set of primitives can be reduced without being inadequate, the more comprehensively will the system exhibit the network of interrelationships that comprise its subject matter." MM explicitly identifies considerations of economy as fundamental to its program. (See Chomsky 1994 for a similar approach in syntax).

The MM model of inflection is designed to account for both regular as well as irregular (or more exactly subregular) verb formation. In the case of German verbal inflection, this comprises the weak, strong and mixed verb classes, of which the following are examples:

(1)

	infinitive	**past(1sg)**	**participle**	
weak:	träum-en	träum-te	ge-träum-t	'dream'
strong:	geb-en	gab	ge-geb-en	'give'
mixed:	bring-en	brach-te	ge-brach-t	'bring'

The main entities required by the model are the base lexical entries (stems and affixes), which combine to form possible inflected verb forms. The output of the combinatorial process, which produces word forms out of stems and affixes (and which overgenerates massively), is restricted by a number of general principles and by the notion of a paradigm structure which together serve to eliminate unwanted forms. The elements leaving the morphology module proper are words which can be projected into the syntax and which are subject to specific syntactic principles such as X-Bar Theory, for example. In agreement with other approaches, such as that of *Deconstructing Morphology* and that adopted by DiSciullo/Williams (1987), and in contrast with other models, such as *Distributed Morphology* and *A-Morphous Morphology* (Anderson 1992), the MM model adopts an early (lexical) insertion analysis.

3.1 Stems and affixes

The basic lexical entries are stems and affixes. Affixes consist of the exponent, i.e., the phonological form, together with an input condition and an output specification. The input condition expresses the sort of contextual (or subcategorisation) information that the affix is sensitive to, and which can be either phonological (moraic structure, thematic vowel, etc.) or morphological (categorial information). The output component adds morphological information in the form of positively specified features. Thus, for example, the suffix /e/ in German is restricted to [-pret, -subj, -pl] verbal stems, as can be seen in (2) below, and adds the morphological feature specification [+1] (eg. *ich schlaf-e* 'I sleep'). Note that affixes are also allowed to have an 'empty' output, as is the case for the 3sg present indicative form /t/ and the infinitive /n/. (See, however, the discussion of these two affixes below.)

In terms of categorial grammar, affixes can be formally conceived of as functors, with the input condition specifying their domain and the output their range. Regular formation (i.e. affixation) is assumed to be strictly information-adding (see *Strict Monotonicity* below). In the categorial rules in (2), the information on the right, following the slash, is the input, and that on the left, preceding the slash, is the output. Note that major features such as [+V] in the input, restricting application of the affixes to verbs, have been ignored here. The feature [+C] is used a convenient means to refer to the imperative, which is considered to be syntactically an instance of the C(omplemetiser) category. The parentheses for the imperative are used because in most cases the affix is optional (thus, for example, *sing*/ *singe* 'sing', but *arbeite/*arbeit* 'work').

(2) The inflectional affixes of the German verb[2]

	AFFIX	OUTPUT		INPUT
1	(/e/)	[+C]	/	[-pret,-subj]
2	/te/	[+pret]	/	[ ]
3	/e/	[+subj]	/	[ ]
4	/t/	[+2,+pl]	/	[ ]
5	/n/	[+pl]	/	[ ]
6	/st/	[+2]	/	[ ]
7	/e/	[+1]	/	[-pret,-subj,-pl]
8	/t/	[ ]	/	[-pret,-subj,-pl]
9	/t/	[+part]	/	[ ]
10	/n/	[ ]	/	[-agr]

In terms of morphological combinatorics, stems are non-complex entities, since, unlike affixes, they do not display the input/output distinction. While regular stems only encode morphological information about their major category class (verb, noun, adjective, preposition), irregular stems bear more explicit, idiosyncratic and non-predictable information about their phonological form and morphological (and syntactic) categories. Thus, for example, together with the information about their major category [+V], the irregular stems *wirf* and *warf* also bear the disjunctive feature set [-1] v [+C] and [+pret] respectively (see Appendix 1 for an illustration of the relevant German verb paradigms). In contrast, the regular stem *werf* only has the information [+V].

The irregular stem forms of a lexeme are assumed to be listed in the lexicon together with the basic entry of that lexeme. For instance, the listed forms for the lexeme *werf* consist of the basic form itself *werf*, as well as the forms *wirf, warf, wuerfe* and *worfn*. The listed forms are assumed to be systematically related to each other, with the relevant relations being represented by means of non-monotonic default inheritance trees (see Wunderlich/Fabri 1994). Note that in this paper we will not go into any details concerning the merits and weaknesses of the inheritance conception used for subregular and irregular stems in MM. Subregular and irregular stems will be listed in the lexicon together with the base, as shown in (3) for *werfen*. (The fact that all of these forms are [+V] by inheritance has not been explicitly notated in (3) and will be taken for granted in what follows.)

(3)

/verf/	[+V]
/virf/	[-1] v [+C]
/varf/	[+pret]
/vorfn/	[+part]
/vyrfe/	[+subj, +pret]

As can be seen in (2) and (3), the relevant morpho-syntactic information is encoded in the form of features. Morpho-syntactic categories are represented by means of sets of feature specifications consisting of the pair <feature name, feature value>, with *feature value* always being binary, i.e. "+" or "-". Thus, for example, the set in (4a) represents the category 'second person feminine singular. (4b) shows the

---

[2] Note that in Wunderlich/Fabri (1994), the affixes /X/ and /tX/ are listed instead of /e/ and /te/ for affixes 1, 2, 3 and 7. X stands for a predictable, underspecified phonological segment, which is optional in certain contexts and is realised as schwa in German after further phonological conditions are met. At the expense of accuracy, but for the sake of simplicity of presentation, we use /e/ here.

simpler notational equivalent which will be used throughout this paper (see Gazdar et al. 1985 for the same notational format).

(4)a. { <2,+>, <f,+>, <plu, -> }
b. [+2, +f, -plu]

3.2 Principles

The combinatory potential of the affixes and the stems is constrained by a number of principles, some of which refer to the hierarchy of functional categories in (5), whose linguistic validity is justified both on ontological and empirical grounds (see Bybee 1985, Wunderlich 1993).

(5) verb < voice < aspect < tense < mood < gender < number < person < status

Any particular element in the hierarchy in (5) is considered to be more highly ranked than any of the elements to its left. The term "status" (German *Satzmodus*, lit. 'sentence mood') refers to categories such as imperative and evidential, which are assumed to be operative in the C(omplementiser) position in syntax. Since gender and aspect are irrelevant morphologically in the German verb system, they will be ignored in the following.

The constraining effects of the hierarchy follow from its interplay with the following constraints:

(6)a. *Fusion:* Only categories which are adjacent in the hierarchy can be fused into one affix.

b. *Affix order:* The order of affixes must conform to the hierarchy, i.e. affixes which express lower ranked categories must be attached first.

The domain of application of *Fusion* is diachronic rather than synchronic and consequently will play no significant role in this paper. *Affix Order* is meant to have default character, since due to certain (language specific) historical developments, the ordering facts at a certain stage of development might appear to go counter to the spirit of this constraint. Usually, however, in such cases one observes a strong tendency for the morphology to be attracted to a hierarchy-conforming stable state,[3] so that the structures do conform to the hierarchy in the next historical phase (see Haspelmath 1993 for a discussion of some interesting examples).

Apart from the above hierarchy-related principles, the principles in (7) below are also assumed.

(7)a. *Strict Monotonicity:* The output must be more informative than the input, relative to a paradigm structure.

b. *Underspecification:* (i) The output contains only "+" values.
(ii) Any information not specified in either the input or the output is "-" valued.

---

[3] Interesting speculations about language modelling by chaotic dynamics and their corresponding perturbation-invariant attractors could follow here, but would go beyond the scope of this paper.

c. *Input Specificity* : An affix applies to the most specific input forms available

The first two principles are different in character from the third in that they explicitly refer to morphological operations, i.e., in the case of German to affixation. In fact these principles partially define what a possible affix might be. Note that affixes 8 (/t/) and 10 (/n/) in (2) above do not violate the *Strict Montonicity* Principle, since it is assumed that within the paradigm defined by the complete set of affixes (see section 3.3), suffix 8 contrasts with suffix 7 (/e/) in having the negatively specified information [-1], and suffix 10 is in contrast with suffix 9 (/t/) in having the feature [-part]. These affixes therefore are informative in virtue of the paradigm structure, which represents the full informational potential of the affix system as a whole, and which will be discussed below.

The *Specificity* Principle systematically restricts the combinatorial potential of stems and affixes in the case of conflicting forms. That is, if two or more forms bear the same information, the one with the most specific stem takes precedence and occupies the relevant cell in the paradigm. An example in German is the pair *warf-st* and **werf-te-st*, the latter of which does not occur. Both of these forms are [+pret, +2], and therefore compete for a paradigm cell. The idea is that *warf-st* is the preferred form because its stem *warf*, being [+pret], is more specific than that of *werf-te-st*, the latter stem displaying no minor features.

Finally, the Principle of *Non-Redundancy* in (8) is assumed to be weaker than *Strict Monotonicity* in allowing the attachment of affixal elements which are assumed not to introduce any morphological information, such as *ge-* in participial forms in German (eg. *ge-arbeit-et* 'worked'). Strictly speaking, therefore, *ge-* prefixation, which seems to be merely a prosodic well-formedness constraint relevant for German[4] (see Wiese (in press)), violates *Strict Monotonicity*, but satisfies *Non-Redundancy*.

(8) *Non-Redundancy*
Particular information can only be added once.
(The output information must not be contained in the input.)

Apart from the general principles just described, the following co-occurrence restrictions, which explicitly disallow certain feature combinations, serve to limit potential affix and stem combinations.

(9) a. [+part] v [-part] → [-agr]
b. [+pret] v [-pret] → [+agr]

Note that [agr] in (9) is equivalent to the INFL category in approaches like that of Chomsky's *Government & Binding* and *Principles & Parameters Theory* (see Chomsky 1981). It refers to agreement involving at least person marking, which is typical of the verbal system (in some languages also preposition and possessor agreement), and is opposed to agreement involving only gender and/or number (but not person), which often occurs DP-internally in adjective-noun agreement and determiner-noun, for example.

[4] This contrasts with *ge*-prefixation in Dutch, which is not prosodically conditioned.

3.3 Paradigms

The final notion to be discussed briefly here is that of a paradigm. A paradigm is a multi-dimensional, potentially recursive matrix whose dimensions are defined by features such as [+/-1], [+/- 2], [+/-pl], etc (see, for example, (12a) below). Paradigms are conceived of as being operative at the interface level between the lexicon and the syntax, taking in candidate word forms and outputting actual word forms (X^0 elements) which project into the syntax.

(10) Paradigms constitute the interface between inflectional morphology and syntax.

The paradigm is a selective device that determines which of the potential stem-affix combinations actually get projected into the syntax as possible inflected words, and also determines which particular feature sets these forms bear by providing default specifications. The stem-affix combinations specified in a minimalist fashion generally lack negative features, unless they contain a stem that provides some negative feature specifications, such as *wirf-t*, for example, that obtains the feature [-1] (through the disjunction [-1] v [+C] from *wirf*). If there is no paradigm structure and a form like *wirf-t* leaves the lexicon underspecified, i.e., without minus features being assigned, it would be able to combine syntactically (i.e., agree or unify) with any subject DP that is [-1], i.e., for example with both *du* 'you' and *er* 'he'. However, empirically only *er* is correct in this case.

One way of overcoming this problem is to devise a simple context-free default-assigning procedure which provides negative feature values for all the feature attributes not having a positive value, so that *wirft* would then be [-C, -1, -2, -pl, -subj, -pret]. This is correct for *wirft* but produces the wrong results with other forms since, for example, the form *werf-n* with the [+pl] suffix /n/ would then obtain the feature set [+pl, -C, -1, -2, -subj, -pret] and would thus be restricted to combinations with 3rd person plural DPs. A simple combination such as *wir werfen* 'we throw' could never be generated. In fact *werf-n* should be left unspecified for either 1st or 3rd person, so that it can be allowed to combine both with *wir* 'we' and with *sie* 'they'. This means that the (negative) default features are not just added 'blindly', and that the distribution of form and feature combinations is best regulated by independent principles. Note that although *werf-n* is unspecified for first and third person, it cannot be allowed to combine syntactically with the second person plural pronoun *ihr* 'you(pl)', since in this case *werf-t* [+pl, +2, -C, -1, -subj, -perf] is the correct form empirically, a fact which must also be accounted for.

The idea of the MM-type paradigm mechanism was designed to cope with the problems mentioned above. Unlike traditional paradigms, in which all the potential cells are occupied by the relevant forms as in (11), so that identical forms often occur, the notion of a recursive paradigm in MM explicitly allows for syncretism. This is accomplished by allowing sub-paradigms to be embedded within larger paradigms, as shown in (12a) for the non-preterite, non-subjunctive full inflectional forms in German. Note that *werf-n* in (12a) is not specified for the dimension [+/-1], i.e., it is not located within the subparadigm defined by this dimension.

(11) a. Traditional present paradigm:

	-pl	+pl
1	werfe	werfen
2	wirfst	werft
3	wirft	werfen

b. Traditional preterite paradigm:

	-pl	+pl
1	warf	warfen
2	warfst	warft
3	warf	warfen

(12) a. MM present paradigm:

	+pl	-pl	
+2	werf-t		wirf-st
-2	werf-n	+1	werf-e
		-1	wirf-t

b. MM preterite paradigm:

	+pl	-pl
+2	warf-t	warf-st
-2	warf-n	warf

Paradigm construction is conceived of as being affix-driven, i.e., the resulting paradigms directly reflect the speaker's knowledge about the distribution of her affixes within the system:

(13) Paradigms are constituted by the combinatory force of the inflectional affixes.

Thus the paradigms in (12) are built up as follows: the most specific affix /t/ [+2, +pl] opens two dimensions and thus defines a paradigm with four cells. Due to the *Specificity* Principle in (14), which states that (with respect to output information) specific forms take precedence over less specific ones, the [+2, +pl] cell is occupied by /t/, since it is more specific than /st/ or /n/.

(14) *Specificity:* A more specific form is preferred to a less specific one.

Given the paradigm with the <2> and <pl> dimensions, the affixes /st/ [+2] and /n/ [+pl] can occupy the cells [+2, -pl] and [-2, +pl], respectively. The affix /e/ in (12a) then introduces the <1> dimension, which creates a sub-paradigm in the remaining empty cell [-2, -pl], to be occupied by /e/ and by /t/, as shown.

Finally, the three constraints shown in (15a), (15b) and (17) are also required. The principle of *Completeness* in (15a) disallows paradigms with empty cells, so that the [-2, -pl] cell in (12b), for example, must be occupied, and since there is no affixed form that provides the required information, the cell is taken up in German by the bare stem *warf*. Another paradigm related principle is the *Uniqueness* Principle in (15b), which ensures that only one form occupies a particular cell, so that a paradigm like that in (16), with three forms occupying one cell, is ruled out.

(15)a. *Completeness:* Every cell of a paradigm must be occupied.

b. *Uniqueness:* Every cell of a paradigm is uniquely occupied.

(16) A non-paradigm:

	+pl	-pl
+2	-t, -st, -n	-st
-2	-n	

The effect of *Uniqueness* is that a number of potential forms have to compete for a particular cell. Which form is taken is determined by the principles of *Specificity* in (14) and by *Simplicity* in (17).

(17) *Simplicity:* A simpler form is preferred to a more complex one.

The term "simple" here refers to morphological simplicity, in the sense that a form that consists of *n* morphemes is considered to be simpler than one that is made up of *n+m* morphemes, with *m* > 1. Thus, for example, although both *warf-n-st* and *warf-t* are [+pret, +pl, +2], the latter is preferred because it is simpler in the sense required here.

The interaction of the principles and representations discussed is assumed to generate all and only the correct inflectional forms of the German verb. This assumption can be tested by a computational model that reproduces in detail the scenario assumed in the MM approach. This model will be discussed in the following sections.

4. The computational model

4.1 General considerations

The initial step was to decide which formal framework was best suited for our purposes. Early on we realised that the formal framework had to display at least three basic properties: it had to be (a) non-deterministic, (b) monotonic and (c) declarative.

(a) *Non-determinism:* By requiring our framework to allow for non-determinism we require an evaluation strategy to be available which can compute *all* possible solutions to a particular query (e.g. what forms are compatible with such and such morphological specifications?). In contrast, a deterministic strategy, as understood here, would prune the space of possible solutions in some way, rendering any solutions other than the selected one inaccessible. As a consequence, the DATR default inheritance formalism (Evans/Gazdar 1990), which has been employed elsewhere to formalise morphological theories (see, e.g., Corbett/Fraser 1993, Gazdar 1992, Gibbon 1992) had to be excluded because of its built-in deterministic evaluation strategy. Being Turing-powerful (cf. Moser 1992) DATR could, of course, be made to mimic a non-deterministic evaluation strategy. The effort required, however, to emulate this non-built-in strategy would be vastly out of proportion.

(b) *Monotonicity:* By monotonicity we informally want to describe that property of a proof system, by which conclusions which may possibly be inferred from the premises may not be invalidated by adding further pieces of information as the proof proceeds. The relevance of montonicity to our enterprise comes from the availability of several logic-based linguistic formalisms frequently employed in computational linguistics. Computer programs that execute grammars written in such formalisms are, in fact, theorem provers which mechanically draw conclusions based on the premises written by the linguist. Now, if these formalisms allowed non-monotonic constructs to be employed, the theorem prover would have to somehow either hypothesise or remember deleted pieces of information during the course of the proof. This might not only lead to a combinatorial explosion, but also adversely affect reversibility of grammars, i.e., the capability of using a grammar both for recognition

and generation of valid linguistic expressions. Obviously in the theory-testing task at hand, we would like to be neutral with respect to the recognition of root-affix strings and to the generation of possible forms out of (underspecified) morphological specifications. Again, since DATR was conceived primarily with generation in mind, it could not be considered appropriate for our purposes. Although the issue of reversing DATR queries has recently received attention, the solution proposed by Langer (1994) is limited to a restrictive subset of DATR. Note that we are not concerned with an implementation of the base lexicon which, recall, contains subregular forms that original MM would like to capture by non-monotonic default inheritance trees.

(c) *Declarativity:* The notion of declarativity captures the basic requirement that a framework should abstract away from the procedural ordering of operations.[5] In other words, we do not want to specify *how* a well-formed object is arrived at, but rather to concentrate on formulating *what* constraints must hold of a well-formed object. This allows a competence description to be compatible with whatever serial ordering of operations is detected by psycholinguistic investigations into the performance domain. (Essentially the same point is emphasised in Pollard & Sag 1994:12-14.) Declarative descriptions, moreover, remain valid in the face of radical changes in the underlying processual paradigm, such as that of the adaptation to parallel modes of computation, for example. Furthermore, in a monostratal conception of grammar, such descriptions serve to avoid so-called rule conspiracies (see Kisseberth 1970), i.e., the fact that a certain well-formedness restriction is preserved by a number of rules interacting to maintain the constraint in question, whereas one could more economically express the restriction itself.

If we adopt the perspective of symbol processing rather than, say, a connectionist, subsymbolic perspective, we are left with a number of linguistic formalisms including, amongst others, CUF (Comprehensive Unification Formalism, Dörre/Dorna 1993) and QPATR (Kilbury 1990a). Compared with current QPATR, CUF has the additional benefit of providing a notion of *typing,* which will be most helpful in what follows. Therefore, CUF was adopted as our base formalism.

4.2 Building blocks for formalisation

To introduce some basic building blocks which we will employ in modelling MM, let us start with the typing concept just mentioned.

A type is a formal entity that in our context may be taken to stand for a coarse characterisation of some subset of the linguistic universe of discourse. Purely for the sake of illustration, we might imagine a partitioning of our universe in animate and inanimate objects, with the relevant inanimate objects being mountains and lakes. Since this is a static division in our toy ontology, we want to declare it once and for all. Note that in a formal account we are forced to state even such obvious distinctions, as they are only self-evident from the perspective of the *human* theorist. Formal types provide a means by which such distinctions can be stated.

Let us now introduce some formal notation. With the expression in (18a) we declare that <type> is exhaustively partitioned (" = ") into the N subtypes indicated,

---

[5] Procedural ordering is usually associated with a specific assumption according to which underlying machine architecture is presupposed, traditionally taken to be of the von Neumann type.

which are pairwise disjoint (" | "). The toy example described above can, therefore, be expressed as shown in (18b).[6]

(18)a. <type> = <subtype1> | <subtype2> | ... | <subtypeN>.

b. *universe = animate | inanimate*.

With the second construct shown in (19a), we can exhaustively specify members of a fully instantiated set of constants (" { ... } "), i.e., atomic types. In our example, this provides the expression in (19b).

(19)a. <type> = { <constant_type1>, ... , <constant_typeN> }.

b. *inanimate* = {*mountains*,*lakes*}.

Now usually one is not satisfied with simply describing a limited universe of the sort we have used as an example. The type system itself has limited expressivity, being basically propositional logic in disguise. To be more expressive, the formal language can be extended by features and values. A feature logic is strictly more expressive formally, constituting a subset of first-order predicate logic (see e.g. Johnson 1988).

A valid expression in our extended CUF sublanguage is then constituted by a type, a (logical) variable written in uppercase, or a <feature>:<value> expression, where <value> is a valid expression, thereby allowing for complex-valued features compared with the more limited SPE-type binary-valued features (see Chomsky/Halle 1968). Moreover, a set of logical connectives to construct new valid result expressions out of valid argument expressions is made available, namely, " & " standing for conjunction, " ; " for disjunction and " ~ " for negation. When necessary, we indicate which expressions take precedence by using parentheses. Examples of valid and consistent expressions appear under (20a) to (20c), syntactically coherent yet semantically inconsistent expressions are displayed under (20d) to (20f):

(20) a. `syn:cat:major:Unknown`

b. `X & ~Y`

c. `case:(`*`dat`*`;`*`acc`*`)`

d. *animate & non_animate*

e. `(case:`*`dat`*`) &` *acc*

f. `X & ~ X`

(20a) describes a feature structure with outermost feature `syn` whose value is again a complex, non-atomic feature structure containing at least the feature `cat` etc. (20b) states that the feature structures denoted by the placeholder variables `X` and `Y` are always different regardless of whether the feature structures remain unspecified or not. Note that variables always have uppercase initials in CUF. (20c) describes a feature structure with the feature case whose value is either the atomic type *dat*(ive) or *acc*(usative). In contrast (20d) does not describe any feature structure at all, since *animate* and *inanimate* have been declared to be disjoint types in (18d), while " & " requires the object sets they describe to have a non-empty intersection. Similarly, (20d) shows a clash between a complex feature structure [`case:`*dat*] and atomic *acc*. Finally, in (20f) the set of objects denoted by `X` is

[6]Here and elsewhere we adopt the typographic convention of writing CUF expressions in the `Courier` font, with CUF types additionally represented as *`Courier italic`*.

required to be different from itself, which, given the formal semantics of CUF, is impossible. The reader may have noted the crucial distinction between description and object domain which gives rise to the intensional character of CUF, rendering, e.g., the inclusion of negation (" ~ ") into the *description* language unproblematic.

To tie the static structure of our universe of discourse (modelled by types) closer to the more expressive capabilities given by valid expressions, we may declare features to be appropriate for a given type and restrict their values to specific types as well. The relevant notation is shown in (21).

(21) <appropriate_type> :: <feature1>:<restriction_type1>,
...
<featureN>:<restriction_typeN>.

Continuing with our toy example, animate beings can be characterised by a body temperature, which can be taken as the value of a feature `body_temperature`. This fact can be stated as shown in (22), with *`degree_celsius`* being further specified, for example, as a suitable interval of the real numbers.

(22) *`animate`*`::body_temperature:`*`degree_celsius`*`.`

Appropriateness declarations help the grammar writer to restrict the use of features to certain situations modelled by types, and thus force one to be explicit about situations and restrictions on values which might be added in other places in the grammatical description. Moreover, they also help in the early detection of inconsistencies that might arise during the development of the linguist's grammar. That is, the consistency of e.g. `body_temperature:`*`37 & animate`* would be provable by the CUF theorem prover from the above definitions, while `body_temperature:`*`26 & inanimate`* would be automatically detected as an inconsistent expression.

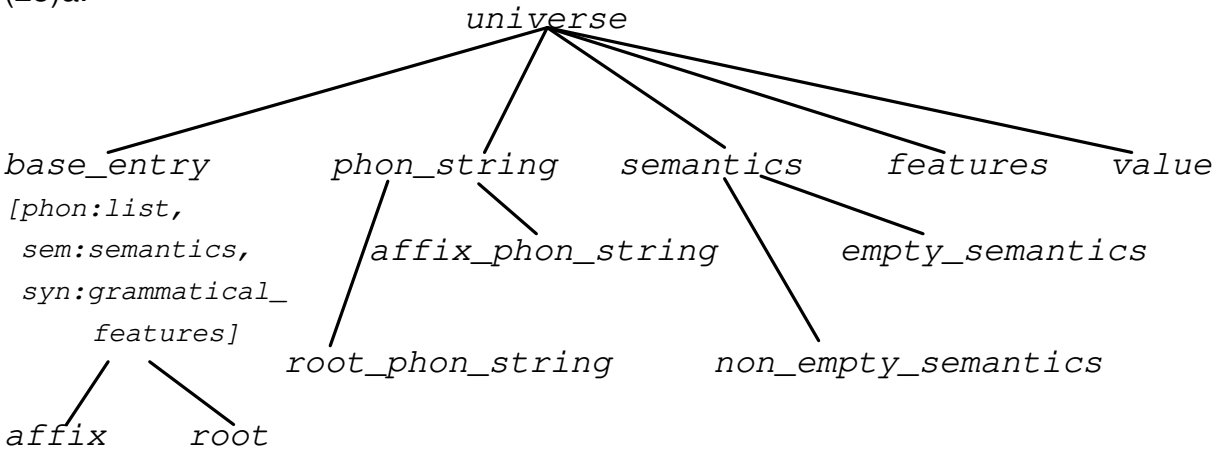
By introducing the restricted type description language presented so far, we established a common means of fixing the universe of discourse which both parties in the present enterprise wanted to talk about. It turned out, therefore, that starting with a type description of the ontology representing our background assumptions about admissible linguistic object types constituted an extremely useful initial step in overcoming the 'jargon barrier' alluded to in section 1.

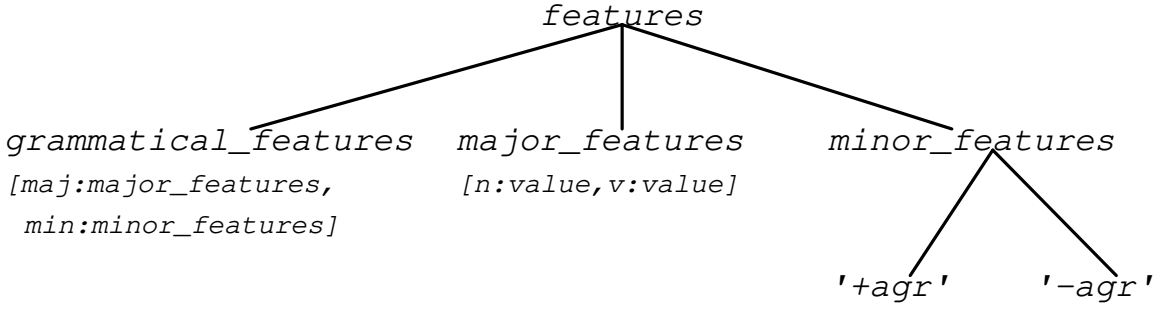
4.3 Formalising the universe of discourse

In this section we describe and justify the basic type inventory that we found necessary to define for the MM model we are implementing. We assume the types shown in (23), which illustrates the relevant parts of the full type hierarchy that we envisage, to be found in Appendix 3.[7] A detailed justification of this type hierarchy on theoretical linguistic grounds would go beyond the purposes of this paper and is irrelevant for our immediate aims. Note especially the formulation of type *`base_entry`* whose feature values are restricted to independently motivated types.

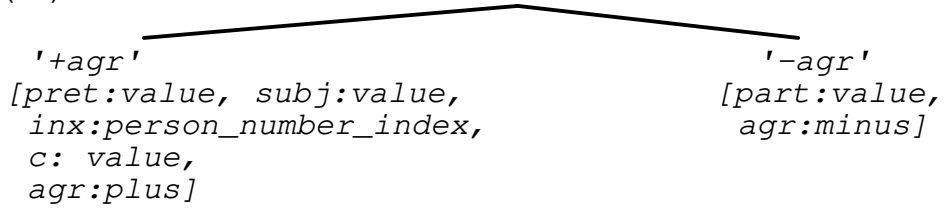
[7] In our graphical notation, appropriateness restrictions tied to a particular type are depicted below that type, enclosed in square brackets.

(23)a.



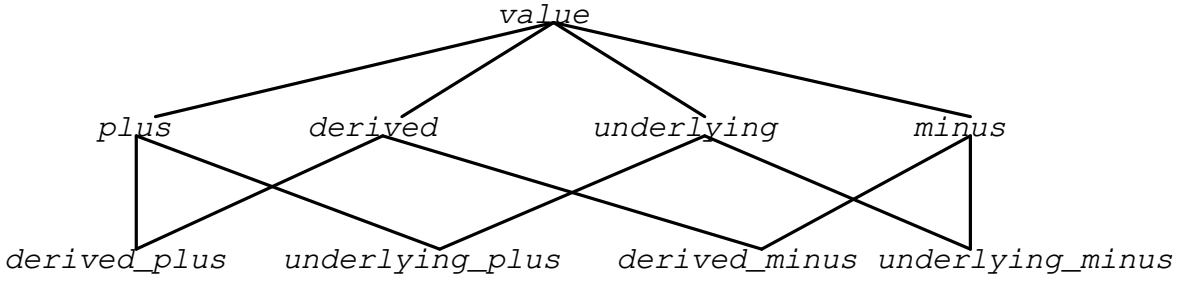


(23)b.



The type *value* formally declares the binary feature values "+" and "-" used in MM. However, recall that in the MM model categorial information originates from two main sources: it is either contained in the basic entry itself, being *underlying* in a sense, or it is introduced derivationally by some affix, i.e., *derived*. It is important for our purposes that this difference of origin can be captured in a declarative, non-derivational fashion by the use of two disjoint types *underlying* and *derived* as subtypes of value, and multiplying out the allowed cross classifications as seen below.

(24)



Crucially, we are able to dispense with any derivational connotations which might be present in the input/output naming scheme for information referral in affix 'rules'. The relevant observation is that a given feature is never referred to both in the input and in the output specification of affixes, i.e., the input/output or *underlying/derived* features are mutually exclusive for any given feature-value pair (cf. (2)). Thus our grammatical description remains monostratal despite the fact that the input/output distinction might suggest otherwise at first. For example, the entry for the preterite stem *warf* has the information [pret:*underlying_plus*] and the preterite affix /t/ has [pret: *derived_plus*]. If desired, we may eliminate the maximal type *derived_minus* from (24), in conformity with the Underspecification constraint (see (7bi) above), making *minus* and *underlying_minus* equivalent with each other. From a computational perspective, however, nothing crucial hinges on this omission.[8] The *Underspecification Constraint* (7bi) should therefore simply be viewed as a substantive rather than a formal restriction on lexicon entries.

The notion of a (morphological) hierarchy or scale, however, can not be recast using disjoint types in one and the same structure as above. The reason is that dynamically updating *a* scale position *within the same feature structure* would be impossible given the monotonic character of CUF. As seen in constraints *Affix Order* and *Fusion* (see (6a/b)), MM assumes a bijection between the degree to which certain features are present in a morphological feature structure and the corresponding scale position to be ascribed to that structure modulo a given scale order. Thus, scale position formally is a device to measure the degree of (under)specification of a given feature structure. This measurement needs to take place at a specified instant of time, since underspecified structures per definition are open to later fill-in.[9] This means that, given this temporal ingredient in the system, a non-monotonic property has been identified as being at the heart of the notion of a scale as conceived of in the MM model. Non-monotonicity arises here because one of the inferences possible with underspecified structures, namely, that they *can* be fully specified subject only to the appropriateness restrictions, is retracted because the paradigm mechanism, for example, assigns only *one* of the possible values, i.e. "-", in a context-dependent way. At first sight this might seem to clash with our attempt to remain within a monotonic, declarative framework. We will show later how this clash can be resolved, limiting ourselves for the moment to describing the scale itself, which, due to its static nature, will be formulated as a *type hierarchy*.

Our aim is to model membership of a scale position in a given scale interval via type containment. For this we compose the scale out of sections characterised by their lower end according to the schema in (25b) for all the positions distinguished in the hierarchy in (5) above. (The hierarchy in (5) is repeated in (25a) in a slightly modified form, including only the positions that are relevant for the German verb system and having the label C instead of 'status'.)

(25)a. verb < aspect < tense < mood < number < person < C

b. ´>=<*Position i*>´ = {*Position i*} | ´>=<*Position i*+1>´

c. ´>=*MOOD*´ = { ´*MOOD*´} | ´>=*NUMBER*´.

---

[8]Note, however, that the distinction is relevant in conjunction with the paradigm mechanism (see below).

[9]The reader is reminded that the logical perspective on underspecification holds that an underspecified feature value is simply the disjunction of all admissible values. E.g. underspecified number is equivalent to pl:(minus;plus).

(20c) describes an instance of the schema, namely the scale interval ranging from *MOOD* to *C*, which is decomposed into the lower edge mood *MOOD* itself or the interval starting from *NUMBER*. Thus, we have a non-disjunctive handle on intervals of the scale characterised by their left edge position.[10]

4.4 Formalising verb stems

In this section we illustrate the representations of the stems, taking the strong verb *werfen* 'throw' as an example. In order to take advantage of generalised structures, work at appropriate levels of abstraction, and reuse building blocks defined elsewhere in the grammar in a modular fashion, we make use of *templates* or, in CUF terminology, sorts.

A template may be thought of as an abbreviatory device in which a feature description is given a compact name to refer to. Moreover the CUF variant of templates allows one to fill in 'holes' in the template description in actual template usage, the 'holes' being referred to as arguments or parameters. Note that there are some important differences between PATR-II templates and CUF sorts. By admitting parameters CUF templates go beyond the expressive power of PATR-II templates, as introduced in Shieber (1986:55). Moreover, recursive templates are allowed in CUF, but not in PATR-II. Allowing for recursion gives the templates the formal power to capture regularities that are usually captured by phrase-structure grammar rules in PATR-II and other systems.

We may now use template names wherever possibly complex feature descriptions were allowed previously. Note that the functional notation whereby a template name with *n* parameters maps onto its body expression is only "syntactic sugar" for an *n+1*-place relation between valid expressions. Nevertheless, it is useful for the reader to think of template 'calls' or usages as being substituted by their bodies.

Templates typically have the following structure:[11]

(26) *Template Definition*

<template name> (<parameter 1 feature description >, ...,
<parameter n feature description>) :=
<template body feature description (f.d.)>.

Nothing crucial hinges on whether templates have a theoretical linguistic status or not since, being part of the *description* domain, they cannot be confused with the linguistic *objects* themselves. For our purposes they are simply extremely handy for abbreviating parts of complex descriptions and indispensible in the present

---

[10] Non-disjunctivity here refers to the avoidance of disjunctive feature descriptions to characterise the scale. Of course they may arise internally in the propositional formulas which underlie types in the CUF system, but the crucial point is that disjunctions in type descriptions are much more efficient than top-level disjunctions involving arbitrary feature descriptions. According to König (1994) type-only descriptions are to be preferred whenever no coindexation between feature structures is necessary.

[11] A fully formal definition can be found in the CUF language report (Dörre/Dorna 1993). Note that parameters are optional, in which case the parentheses are omitted in template definition and usage, as for example in `verb` (27b).

framework for encoding recursive relationships. As an example, witness the definition of the template for verb stems in (27a). *verbal_root* has the parameters PhonString and Semantics. Its body describes a feature structure with at least the features phon and sem, whose values are coindexed with the parameters as indicated. The *verb* template in (27b) describes a feature structure which presets the value of the major categorial information to [-N, +V]. The expression "...& verb" in (27a) means that the partial feature structure described by "phon:...& sem:..." is unified with the structure described by *verb* in (27b).

(27)a.
```
verbal_root(PhonString, Semantics) :=
    phon:PhonString|_PlaceHolderForMorePhonologicalMaterial]&
    sem:Semantics &
    verb.
```

b.
```
verb := syn:maj:(n:underlying_minus &
                 v:underlying_plus).
```

Note that in (27a) the repeated occurrence of (uppercase) variables serves to transfer or structure-share information between the template's parameters and its body. The abbreviatory list notation [FirstElement|RestList], borrowed from PROLOG, makes it possible to avoid writing nested first-rest ('F'-'R') feature descriptions in the CUF surface syntax to recursively compose lists of a first element (the value of ´F´) and a list tail (the value of ´R´). The second line of (27a) is equivalent to (28a), i.e., the phonological content of a word is formalised here as a list of component phonological strings corresponding to the morphemes participating both in content and surface order. (We refrain from going into the complexities of adequate phonological structuring in this paper.) The template in (27a) allows the human lexicon-compiler to write entries for verb stems like that shown in (28b) for *werf* 'throw'.

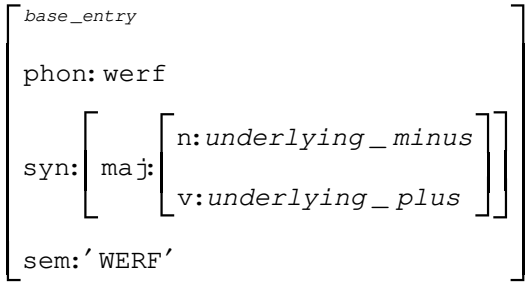
(28)a.
```
phon:('F':PhonString &
      'R':_PlaceHolderForMorePhonologicalMaterial)
```

b.
```
verbal_root(werf, 'WERF' )
```

Note that the trivial, atomic semantic content 'WERF' in (28b) is sufficient for our purposes since elaborate semantic formulas are not relevant for the aims of this paper.

By substituting template bodies for the template name 'calls' according to the formal semantics of CUF as described above, the specification in (28a) describes the structure in (29), depicted in the usual attribute value matrix (AVM) notation with type annotations in italics:

(29)



The template in (28a), however, still lacks one important piece of information required for a stem, since minor features for 'irregular' (strong) stems such as preterite *warf*, for example, which is classified as [+pret], must also be included. In (30a) `pret(input_plus)& agr(value)` evaluates to the expression in (30b).

(30)a. `verbal_root(warf,'WERF')& pret(input_plus)& agr(value).`

b. $$\left[\text{syn:}\left[\text{min:}\left[\begin{array}{l}\text{pret:}\textit{underlying\_plus}\\ \text{agr:}\textit{plus}\end{array}\right]\right]\right]$$

One more element, however, is required for the lexical entry of a verbal stem to be complete: the scale position associated with its morphological feature content, i.e., its degree of morphological specification. Using a distinguished argument of the `lex`/1 sort[12] which disjunctively specifies the lexical entries, i.e. morphemes, we can hold the scale position without storing it in the result feature structures themselves. The concrete values for scale positions are derived mechanically from the descriptions in the right-hand side of `lex`/1 clauses by noting which feature is highest according to the scale, thus underlining its merely technical status. Insertion into the CUF grammar is done manually at present.

The following are the entries required for the strong verb *werfen*:

(31)

```
lex('V')   := verbal_root(werf,  'WERF').
lex('ONE'):= verbal_root(wirf,  'WERF')
                & not_first_per(input_minus)
                & agr(value)
                & not_pret(value)
                & not_subj(value)
                & not_plu(value).

lex('C') := verbal_root(wirf,  'WERF')
                & not_first_per(input_minus)
                & c(input_plus)
                & sec_per(input_plus)
                & not_pret(value)
                & not_subj(value)
                & not_plu(value)
                & agr(value).

lex('TENSE') := verbal_root(warf,'WERF')
                & pret(input_plus)
                & agr(value).

lex('ASPECT') := verbal_root(worfn,'WERF')
                & part(input_plus)
                & not_agr(input_minus).
```

[12] The notation x/n here stands for "template-name/arity". Thus, in (31) the template-name is `lex` and its arity, i.e., the number of arguments it takes, is 1.

```
lex('MOOD') := verbal_root(wuerfe,'WERF')
               & pret(input_plus)
               & subj(input_plus)
               & agr(value).
```

Recall, finally, that MM proposes that subregular patterns should be represented by means of a non-monotonic default inheritance network, while the mechanism that creates regular formations (i.e. CG) is monotonic. Capturing subregular patterns in structured lexical entries has been omitted from the present study, as the focus was more on modelling principles in action. The only consequence is that entries which would be systematically related once a default network organised their interrelationship appear as unrelated members of the large lexicon disjunction at present. Crucially, the nonmonotonic default-based structuring of subregularites is of the compile-time or learning phase variety, which once again means that the lexicon available during parsing or generation is free from default-induced nonmonotonicity even if modelling of subregularities would be added to the implementation as it stands. The version of the lexicon after ´compiling out´ the subregularity default networks would be of the very same kind as what is presently employed.

4.5 Formalising affixes

The template for affixes is shown in (32). Recall that the affixes themselves, unlike the stems, do not have a Semantic Form. In the MM model they are conceived of as functors, in the sense of categorial grammar, that introduce features which are then interpreted semantically before lexical insertion, i.e. projection into the syntax, takes place.

(32) `affix(PhonString):= phon:[PhonString]&verb.`

Although the derivational connotations of having the morphology 'feed' the syntax would play no role if we adopted a declarative, monostratal model of syntax and semantics as well, (cf. e.g. Head-driven Phrase Structure Grammar, see Pollard/Sag 1987), we can still deal with the absence of the semantic contribution by underspecification.

(33) shows the full list of inflectional affixes required for German. As far as the type difference between *underlying* and *derived* is concerned, the information contained in the input of the affix is specified as their least common supertype *value*, which of course means that it is unifiable with both. Note that, for purely mnemonic reasons, we have also used parameterless templates `output`/0, `input`/0 etc., which shows that templates may also be employed for renaming purposes. (The text after " % " is a comment, hence does not form part of the formal description of feature structures.)

(33)
```
output_minus := derived_minus.
output_plus := derived_plus.
output := derived_plus.
input_plus := underlying_plus.
input_minus := underlying_minus.
input := value.
```

```
lex('C'):= affix(e)       & c(output)    %IMPERATIVE SG
                          & not_pret(input)
                          & not_subj(input)
                          & sec_per(output)
                          & agr(value).

lex('TENSE'):= affix(te) & pret(output)      %PRETERITE
                          & agr(value).

lex('MOOD') := affix(e)  & subj(output)      %SUBJUNCTIVE
                          & agr(value).


lex('TWO') := affix(t)   & plu(output)       %2PL
                          & sec_per(output)
                          & agr(value).

lex('NUMBER') := affix(n)& plu(output)       %1PL OR 3PL
                          & agr(value).

lex('TWO'):= affix(st)   & sec_per(output)   % 2SG
                          & agr(value).

lex('ONE') := affix(e)   & not_pret(input)   %1SG
                          & not_subj(input)
                          & not_plu(input)
                          & first_per(output)
                          & agr(value).

lex('C') := affix(t)     & c(output)    %IMPERATIVE PL
                          & plu(output)
                          & not_pret(input)
                          & not_subj(input)
                          & sec_per(output)
                          & agr(value).

lex('ASPECT') := affix(t)& part(output) %PARTICIPLE
                          & not_agr(output_minus).

lex('TWO') := affix(t)   & not_pret(input) %3SG
                          & not_sec_per(output_minus)
                          & not_first_per(minus)
                          & not_subj(input)
                          & not_plu(input)
                          & agr(value).

lex('ASPECT') := affix(n)& not_agr(output_minus). %INFIN.
```

Note that the fact that the imperative forms are restricted to 2nd person, expressed by the implicational co-occurrence restriction [+C] --> [+2] in Wunderlich/Fabri (1994), has been accounted for by explicitly specifying both the antecedent and the consequent in the entry. The same applies to the feature `agr`, since according to Wunderlich (1992: 3) "a verb form is [+AGR] if at least one of the features PRET, SUBJ, PER or PL is specified lexically".

4.6 The Morphological Combination Operation

The combination of the stems and affixes available in the base lexicon is carried out by the Morphological Combination Operation (MCO), which concatenates phonological strings and unifies the categorial information associated with those strings. For our present purposes we assume a rather simplistic model involving only concatenative morphology. Non-concatenative morphology, required for the generation of correct surface voice and aspect forms in languages like Arabic can, of course, be dealt with in a constraint-based setting like ours. For the present, however, we may neglect the necessary extensions (see, however, Klein 1993 for a constraint-based treatment of non-concatenative Sierra Miwok morphology in a formal language similar to CUF).

(34) The Morphological Combination Operation (MCO)

```
mco:= morph_combine(start_with_a_root,_,_).

morph_combine(start_with_a_root,Phonology,_) :=
   phon:Phonology &
   is_a_root(MinimallyRequiredScalePosForFirstAffix,Affixes)&
    morph_combine(incorporate_affix, Affixes,
   MinimallyRequiredScalePosForFirstAffix).

morph_combine(incorporate_affix,[Affix|MoreAffixes],
   ScalePosition):=
   is_an_affix(Affix,ScalePosition,NextPosition)&
   morph_combine(incorporate_affix,MoreAffixes,NextPosition).

morph_combine(incorporate_affix, [], _):= _.

is_a_root(next_position(ScalePosition),Affixes):
   lex(ScalePosition)&
   phon:[root_phon_string|Affixes].

is_an_affix(AffixPhon,OldPosition,next_position(OldPosition))
     :=
   syn:GramFeats &
   true(lex(OldPosition)& affix & phon:[AffixPhon]&
   syn:GramFeats).
```

The basic structures captured by the recursive CUF definitions can be informally described by the following phrase-structure rules:

(35) $mc_{start\text{-}.with\text{-}a\text{-}root} \rightarrow root\ mc_{incorporate\text{-}affix}$

$mc_{incorporate\text{-}affix} \rightarrow affix\ mc_{incorporate\text{-}affix}$

$mc_{incorporate\text{-}affix} \rightarrow \varnothing$

As (35) is a right-linear grammar and we know from formal language theory that grammars of this type define regular string sets, we may equivalently describe the allowable morpheme combinations by the regular expression **Root Affix***, with the superscripted Kleene-star operator * denoting zero or more concatenations. Note that sets of possible morpheme combinations need not necessarily be regular in the light of languages like Bambara, which apparently possess word formation processes that go beyond what is describable by regular sets.[13] However, formulating a suitably enriched MCO to cover such cases presents no problems in the CUF context.

As can be seen in (34),[14] the MCO is defined by invoking an auxiliary template morph_combine/3, specifying that it should start by expecting a suitable root morpheme. The recursive call to morph_combine in the first defining clause then limits further concatenation to affixes. The identification of a root is based on partitioning the set of phonological strings into those belonging to an affix and those belonging to a root (*root_phon_string*). Alternatively, we could partition the value of sem into the two types *non_empty_semantics* and *empty_semantics*, assigning inflectional affixes to the latter and drawing on this distinction for root selection.

Apart from lexicon access and transportation of its associated scale position (lex(ScalePosition)), the list of (segmented) affix strings is separated from the word's phonology string and percolated to the last argument of is_a_root/2 to guide the task of stripping affixes one-by-one and unifying their contents. This is done in the second clause of morph_combine, where an affix is required which is both compatible with whatever phonological string (Affix) is prescribed by the previous invocation *and* with the previously identified next admissible scale position (ScalePosition). Recursively, then, further affixes are chopped off. Of course, the case where no affix follows, which is identifiable when the empty list [] has been reached, has to be treated in a fully formal account as well, giving rise to the third clause.

Unlike the root case, in is_an_affix/3 only the syntactic major and minor features of an affix are unified with the structure built up so far, provided the incoming phonological information matches that stored in the lexical entry itself. This results from incompatible *empty_semantics* type assignment of its sem feature and the affix-only phonological material which would not map straightforwardly onto the material of the whole word contained in the built-up structure. Again with underspecification to type *semantics* and mechanisms for representing and unifying 'floating' affixes, i.e. affixes, that are not fully specified with respect to their linear precedence relationships, this selective information transport of syntactic features only may be supplanted by straightforward unification (see Engelkamp et al. 1992, Riehemann 1993). At present, we have to use the true/1 template, which shields the subcomputation in its argument from affecting the feature structure that is built up ´outside´. Note also the computation of the interval of next-higher scale positions relative to the old or current position (next_position(OldPosition)),

---

[13] Bambara displays an N-o-N composite noun construction having the meaning *whichever/whatever N*, which would have to be classified as context-sensitive in the formal language sense, akin to the clearly context-sensitive formal language *ww*, i.e. total reduplication of some word *w*. An example of an N-o-N construction is shown in the following (see Culy 1985):

vulu	-->	vulu-o-vulu
dog		whichever dog

[14] Note that, although for the sake of familiarity the following descriptions of the MCO and of the system in general are expressed in somewhat procedural language, the grammar itself is fully declarative, as has already been explained in section 3.

where `next_position/1` is a simple tabulation of relevant points and intervals of the scale using the type encoding discussed in (25b) above.

4.7 Constraining morphological combination

At the level of analysis described up to now, stems and affixes are allowed to combine freely, so that any restrictions on possible forms at this level result only from the input conditions present on some of the affixes. Thus, for example, a form like *arbeit-n-t* is excluded because the affix for 3sg /t/ cannot combine with a stem to which plural /n/ has already been attached since the former has the input condition [-pret, -subj, -pl]. A form like *arbeit-t-n;* however, which has the /t/ attached first and /n/ later, can be generated since it is not restricted by any input conditions.

A massive overgeneration of forms is hindered in MM by the general principles and constraints described in section 3 above which fall into the following classes:

(i) those constraining the structure of the entries for affixes:
Underspecification,
(ii) those restricting possible stem-affix combinations:
Strict Monotonicity, Affix Order, Non-Redundancy, and Feature Co-occurrence Restrictions,
(iii) those defining the structure of paradigms:
Uniqueness and *Completeness*,
(iv) those restricting the number of forms competing for paradigm cells:
Input Specificity, *Specificity* and *Simplicity*

The principles under (i) and (ii) will be discussed in this section, and (iii) and (iv) will be discussed in section 5. For convenience the relevant principles are repeated below:

(36)a. *Underspecification:* (i) The output contains only "+" values.
(ii) Any information not specified in either the input or the output is "-" valued.

b. *Strict Monotonicity:* The output must be more informative than the input, relative to a paradigm structure.

c. *Non-Redundancy:* Particular information can only be added once.

(36a) restricts lexical specifications in its (i) part, a substantive but not formal restriction, and formulates a requirement for underspecified cases in (ii) which become relevant in connection with the paradigm machinery, to be dealt with in the next section.

Note that in the entry for the affix /t/ for 3sg in (33), the output includes the feature [-2], which obviously is a violation of *Underspecification*. This inevitably follows from the specification of *wirf*, which is marked as [-1] underlyingly, hence requiring any affix to address a higher scale position than ´ONE´ in order to successfully combine with *wirf*. Therefore *-t* in *wirf-t* ´s/he throws' has to include the offending specification to achieve scale position ´TWO´, thereby rendering (36a) a violable constraint or tendency rather than an absolute requirement.

As far as (36b) is concerned, a better name perhaps would be Strict Subsumption, a requirement that, in straightforward translation, would result in a non-monotonic underlying logic from a formal point of view. However, the fact that information must be added is fulfilled by *Affix Order* , which is straightforwardly incorporated into the MCO by ensuring that whenever a combination with an affix takes place, the next higher position on the scale is available for further affixation: `next_position(Scale Position)` in (34). Note that the qualification '...relative to a paradigm structure' in (36b) was originally required because the affix specification for 3rd sg /t/ did not have a specific output and the 'missing' information was assumed to be supplied by the paradigm. However, since paradigms are affix-driven in the first place, apparently a paradox arises in this case. As we have seen above, though, we are forced to extend the 3rd sg /t/ feature specification on independent grounds, so the primary motivation for the qualification has gone now and we may safely omit it.

Just like (36b), (36c) is also covered by the proper realisation of *Affix Order*, which ensures that, since information originates with affixes, together with an appropriate specification of affix content ´no affix may leave the current scale position unchanged´, which we would claim is a proper reformulation of (36c). It is one of the virtues of formalisation that one is forced to give a precise formulation of the principles to be modelled, which as a consequence leads to possible conflation of some of them or a clarification of their status, as has just been shown.

4.8 Feature Co-occurrence Restrictions

The following are the FCRs mentioned in the original MM paper. (Recall that [C] is used to encode the imperative.):

(37)	a.	[+part] $\vee$ [-part]	$\rightarrow$	[-agr]
	b.	[+pret] $\vee$ [-pret]	$\rightarrow$	[+agr]
	c.	[+C,+V]	$\rightarrow$	[+2]
	d.	[+1]	$\rightarrow$	[-2]

(37a) and (37b) are implemented by appropriate type declarations, since `pret` is defined as appropriate for the type '*+agr*', while `part` falls under the disjoint '*−agr*' type. In fact, `agr` as a feature is a consequence of the implicit usage of an untyped feature calculus, which can be replaced by the subtypes of *minor_features* just mentioned, i.e. '*+agr*' and '*−agr*'. As the paradigm mechanism as currently implemented also lacks knowledge of types, we add appropriate `agr:`*plus* and `agr:` *minus* specifications to the lexical entries due to this technical requirement.

As for (37c), note that in Wunderlich/Fabri (1994) it was claimed that the attribute C for the imperative is only being used for convenience. It is important to keep this in mind since this terminology might otherwise lead to confusion. It is not strictly correct to say that, used in its traditional syntactic sense, the category C(omplementiser) itself is always [+2], since in German, for example, any verb form can move to the C-position in the syntax in verb-second constructions, and not just [+2] categories. So C here is to be interpreted simply as a convenient morphological feature name that reflects the fact that the imperative is a status (*Satzmodus*) category (see section 3). Note, moreover, that the imperative, which is meant to be characterised by this feature, is only one type of C category, together with other

possible categories such as evidential, for example (see Wunderlich/Fabri 1994).[15] Although the evidential is not grammaticalised in German, one could argue that it is grammaticalised in other languages in which it has morphological or syntactic effects (such as the so-called inverse construction in Georgian, for example, which typically occurs when something is reported by hearsay, see Fähnrich 1987). In these languages, however, evidentials are not restricted only to 2nd person, but rather to 2nd and 3rd person. A decision about what feature name is to be chosen for the imperative can only be taken once the categories that are relevant for the C-position are studied in more detail. Recall that this co-occurence restriction has been explictly expressed in the base entry of the relevant affixes (see section 4.5).

In addition to the FCRs (37a) to (37c), the restriction on the person categories in (37d), which derives (38) by *modus tollens*, is also required for German. In Wunderlich/ Fabri (1994) this FCR is taken for granted for German, but is explicitly stated for Arabic.

(38) [+2] → [-1]

In contrast to the other FCRs, (37d/38) is language specific. In fact, the 1st person inclusive category in languages such as Plains Cree (an Algonquian language, see Wolfart 1973), for example, can be described by the co-occurrence of the two features [+1,+2]. Finally, note that (37d) is relevant both as a substantive constraint on possible lexical entries for German as well as in paradigm build-up, as will be seen later.

5. Implementing the Paradigm Mechanism

The general scenario assumed in the MM model is that of a generating (in fact, overgenerating) source that outputs word forms, together with their morphological feature specification, and a filtering device, namely the paradigm mechanism, which extracts a subset of the forms according to paradigm construction principles. Having already described the generating source, a computer-executable CUF-grammar (CG), in this section we describe the paradigm mechanism (PM) and its implementation.

Note, first, that the overall system possesses a non-monotonic, default-based character, despite its monotonic CG part. This is due to the designated point of time at which (possibly) underspecified feature specifications are handed over to the PM, being effectively frozen as to their degree of specification. Moreover, at least in part, the PM may be regarded as a sophisticated context-sensitive default-fill-in device that is guided by its construction principles. The observed non-monontonicity, however, is less problematic with respect to the original aims of this paper than one might at first think, as should become clearer below.[16]

---

[15] Hence, C can be thought of as a short name for something like ´is admissible in syntactic C(omplementiser) position´.

[16] A problematic aspect that remains is that there appears to be a subtle dependency on being able to identify the underspecified status of a feature structure. This would be impossible at first sight if the logically equivalent encoding of an underspecified value in terms of a N-way disjunction between admissible feature values would be employed and disjuncts are enumerated one at a time in machine proofs, as usual (recall that the CUF interpreter is actually a kind of machine theorem prover). In such a scenario structures would be fully specified at each single enumeration step. We might rescue this unfortunate dependency on details of the actual implementation of feature structures by collecting all solutions to a feature value description instead, noting that the set of

Secondly, the CG part may be characterised by the attributes 'local' and 'serial'. This means that only the morphemes to be combined for a particular word form can be seen by the combinatory machinery, and any global reasoning involving other forms is impossible in the current CG set up. Forms may be generated one after the other, but the set of forms is not available simultaneously. In contrast, the PM is expressly designed to reason about the whole set of word forms in a 'global' fashion, which in a sense entails 'parallel' computation and information flow between individual forms. This might look rather expensive from a computational perspective, akin to parallel distributed processing models and other massively parallel models of morphology (see e.g. Anderson 1988). However, let us introduce a fundamental distinction between morphological combination time, at which words are actually parsed or generated (*run time* or *on-line* in computer science terminology), and a prior learning time, at which especially lexical knowledge is acquired (*compile time* or *off-line*). Drawing on that distinction we assign PM activation to the latter stage, thereby shifting the expensive and potentially time-consuming part of our model into a non-critical phase, in which enough time is assumed to be available. In other words, unlike a reasonable demand for, say, short reaction times when recognising a word, we would not postulate a similar demand for paradigm construction to be completed in a matter of milliseconds. Without claiming cognitive adequacy for our implementation we nevertheless feel satisfied that by introducing the distinction just described we may avoid a gross incongruency with what is known about human processing of words. Thus we may view paradigms as static devices that organise a fixed finite set of items once and for all.

In order to make the results of paradigm construction bear fruit during morpheme combination, the idea is then to transfer the information about well-formed occurrences and combinations of morphemes back into the CG, namely via appropriate type definitions derived from paradigm structure. Crucially, the resulting enriched CG' directly obeys the paradigm-induced restrictions without the need of a dynamically active notion of "paradigm", with its concomitant global and parallel side-effects. Moreover, as CG' is still expressed in a monotonic formalism, monotonicity at combination time is retained, making such demands as reversibility much easier to fulfill than they would be in a bipartite setup. From another angle, CG' is somehow akin to the adult grammar after L1 acquisition is finished. (A detailed exposition of CG´ construction can be found in Appendix 2.) Of course, major questions still remain as to how unknown words should be dealt with in this setting, but, unfortunately, consideration of such questions goes beyond the scope of this paper.

If one is willing to accept the discovery of hierarchical paradigm structure from an unstructured set of morphologically annotated and segmented word forms as a significant learning result, one could effectively regard the combination of CG with PM, and substitution of CG by CG', as a highly specialised learning mechanism. (See Barg 1994 for a more general, but theory-neutral machine learning approach applied, inter alia, to German nominal inflection). It will be seen that the learning mechanism needs the entire set of candidate forms as input, a mode of operation termed ´batch mode´in the learning literature. This contrasts with íncremental mode´learning, where the acquisition mechanism would be designed to cope with data as they come in, refining the generalisations inferred when further data

---

solutions contains more than one member (e.g. {plus,minus}) in cases of underspecification. Yet again this operation transcends what is available in a first order predicate logic language, hence has been termed ´extralogical´ in the logic programming literature. At present, we have to leave open the question of how to avoid extralogical operations arising from our close computer modelling of MM.

becomes available. Clearly, incremental mode adds to a model´s psycholinguistic adequacy. Also, our model deviates from a realistic scenario by not being robust with respect to noisy data, i.e., ungrammatical and incomplete forms being among the candidates to be processed. Finally, the concurrent induction of morphological feature annotations and the segmentation of words into morphemes from raw observations alone present major theoretical and computational challenges which cannot be adressed in the present study (see, however**,** Cartwright/Brent 1994). Despite these present limitations we believe the achievements in our setup to be a highly useful initial step.

The general work done by the PM consists of two major steps . In a first step the set of candidate word forms is preprocessed and ordered according to a number of principles which will be discussed in section 5.1. In the second step, described in section 5.2, candidate forms are taken, with the 'highest-ranking' form being chosen first to actually create the paradigms, so that relevant forms can be inserted into the cells thus being made available. For various reasons (see below), insertion may fail, so that paradigm insertion may be viewed as a violable constraint. The final result is a structured paradigm representation, whose cells contain the word forms which have not been rejected, and which are thus well-formed according to the principles.

In the next two sections we turn to the concrete details and the implementation of the principles guiding paradigm construction. Note that, due to the necessary interaction between CG and PM and the nature of CUF, which is built on top of the now standard PROLOG logic programming language, the PM has been implemented in PROLOG (more precisely QUINTUS PROLOG). Communication with the CG proceeds through a well-defined abstract data type for exporting CUF representations and operations, which is provided by the CUF system.

5.1 Ranking candidate word forms

Below is a list of paradigm ordering principles. Note that `precedence´ in (39) is assumed to be transitive and that the principles apply to morphologically annotated candidate word forms, despite some sloppy formulations below.

(39) a. More morphological feature specifications take precedence over fewer feature specifications.

b. Features with underlying values take precedence over those with derived values.

c. Shorter strings take precedence over longer ones.

Together these principles constitute an elaboration of the notion of specificity. (39b) is an implementation of the notion of *Input Specificity* and (39c) implements a modified version of the *Simplicity* constraint. *Simplicity* now refers to surface phonological length rather than word-internal morphological complexity. This produces the correct results, at least for German verb inflection.

Since these principles compete in establishing a ranked list of candidates, they have to be ordered. (To see the conflicting demands in need of resolution, imagine, e.g., a shorter form with fewer features compared with a longer form with more features). The ordering we have assumed is that given by the reading order in (39), with (39a) being the most highly ranked principle. In order to see how each of these principles is implemented using straightforward PROLOG techniques, see

Appendix 2. In essence, the higher precedences are mapped to PROLOG terms that are lower in the standard PROLOG sorting order, and then the list of compound PROLOG terms consisting of precedence information and genuine candidate content is simply sorted, thus bringing higher-ranking items towards the head of the candidate list.

In addition, some preprocessing is necessary before Phase II can be carried out. One linguistically uninteresting, technical step is the conversion of the CUF typed feature structures describing candidates into the format chosen for the PM. In another step features whose values may be (possibly trivially) set to '*minus*' are discarded. The reasoning here is that minus value assignment is generally precisely the default-filling mechanism alluded to above, and should hence be completely under control of the PM. Minus values arise in CG in the first place because they are a consequence of default minus insertion for features lower than the highest-ranking feature for a particular entry[17], thus limiting their combinatory potential from the outset.

5.2 Paradigm insertion

Since the MM conception of paradigms specifies that they should be affix-driven, paradigms are created on demand only, i.e., under the pressure of the word forms generated by CG. The insertion of forms within already existing paradigms means that the morphological specifications of insertion candidates must comply, i.e., unify with the featural coordinates which form the 'address' of a given paradigm cell. If the specification of a candidate demands insertion into a previously empty cell, and if that candidate carries feature specifications that have not yet been saturated by the already existing cell addresses, a new paradigm with N excessive, unsaturated features as coordinates is created which contains 2^N cells due to the binarity assumption for feature values. The first situation of this sort is given by the most highly ranked word form. The paradigm it creates, which we call 'the reference paradigm', contains all the other (sub)paradigms.

If, contrary to the situation just described, a form lacks certain features with respect to the current embedding paradigm (recall that paradigms are recursive), minus-valued features are supplied in accordance with MM principles. This is an instance of how the process of implemention has made an assumption of MM more explicit. Only through implementing the PM initially without fixing the spell-out of missing features and observing the unwanted multiplication of solution paradigms differing only in what value was ´guessed´ for these features led to the refinement with respect to minus insertion. Crucially, the role of minus value fill-in is seen here to *resolve ambiguity in paradigm cell placement*. Note that this refinement makes the strong (and falsifiable) prediction that it is always the same 'all minus' cell that is occupied in such circumstances. If the features of a form match the co-ordinates of a cell, that cell is occupied according to the features involved, possibly travelling down some nested paradigms.

At this point, two possible situations may arise: (a) the cell eventually arrived at is free or its content is identical to the candidate form, or (b) the cell is already occupied. In the former case, the candidate is simply inserted, while in the latter case the candidate is deleted, which means that paradigm insertion is violable, as

[17]This is akin to the Ranking Principle in the original MM conception, cf. Wunderlich (1993). Note, however, that this principle is violable, since lower-ranked features in lexical entries receive minus values only if no FCR interferes and only if the feature in question is not [agr] (cf. (33)).

has already been mentioned. Note that a cell is blocked from being occupied by a form either due to the presence of some other candidate which has precedence and is inserted at an earlier stage, or because some FCR rules out any insertion into that particular cell (as is the case, e.g. with the [+1, +2] cell in German). Thus, the reference paradigm constitutes only a virtual paradigm, which can be interpreted as a real paradigm, once the areas that are blocked by FCRs are subtracted.

Obviously, FCRs constitute a powerful device for the exclusion of cells from further consideration, and they should therefore be properly motivated. Apart from the FCRs described above the set of FCRs shown in (40) was also found to be necessary.[18]

(40)a. [+pret] → [+agr],
b. [+subj] → [+agr], [+C] → [+agr], [+plu] → [+agr], [+2] → [+agr],
c. [+1] → [+agr]

The motivation for these FCRs comes from the fact that paradigms as originally conceived of in the MM model do not have types associated with them but binary features. The fundamental division between the disjoint types '*−agr*' (and its single `part` feature), on the one hand, and '*+agr*' (with associated features `pret,subj, c,plu,two` and `one`), on the other, must therefore be recapitulated in the feature-only paradigm environment. This is the purpose of the FCRs in (40), as the reader may verify in the paradigm illustrated in Appendix 3. (Again, see Appendix 2 for details of the implementation.)

The following is an excerpt of paradigm construction derived from actual paradigm tool output. A full, unedited trace of the construction steps is found in Appendix 2.The preterite non-subjunctive forms generated by CG are first ordered as follows:

(41)
```
26:[agr(+),plu(+),pret(+),two(+)]:warft >>
27:[agr(+),plu(+),pret(+),two(+)]:warfnt >>
28:[agr(+),plu(+),pret(+),two(+)]:warfnst >>
30:[agr(+),plu(+),pret(+),two(+)]:werftet >>
31:[agr(+),plu(+),pret(+),two(+)]:werftent >>
32:[agr(+),plu(+),pret(+),two(+)]:werftenst >>
45:[agr(+),plu(+),pret(+)]:warfn >>
46:[agr(+),plu(+),pret(+)]:werften >>
53:[agr(+),pret(+),two(+)]:warfst >>
55:[agr(+),pret(+),two(+)]:werftest >>
62:[agr(+),pret(+)]:werfte >>
```

The next step is the construction of the paradigm in (42) by inserting candidates according to the ordering just established, as long as no principles or FCRs are violated. Note that the dimensions [pl,pret,two], instantiated by the most specific form *warft*, define the paradigm.

---

[18]The actual syntax used in the PM context has '>>' for the rightheaded arrow, replacing the [<Value><Feature>] notation by <Feature>(<Value>) and substituting one/two for 1/2, so that e.g. (40c) appears as one(+) >> agr(+), cf. the full documentation in Appendix 4.

(42) The [+pret, -subj] paradigm:

	+pl	-pl
+2		
-2		

The available forms now compete for the defined cells in accordance with the ordered principles discussed above. Thus, for example, the forms in (43) compete for the [+2,+pl, +pret] cell. *warft* is the form that has highest precedence on the list since it fulfills all the principles, while all the other forms with the same feature set do not. Therefore *warft* is inserted into this cell and the other forms are implicitly rejected, as the attempted insertion is blocked by the prior presence of *warft*. The section on the right in (43) shows which principle in (39) in section 5.1 is violated.

(43)
```
26:[agr(+),plu(+),pret(+),two(+)]:warft>>
27:[agr(+),plu(+),pret(+),two(+)]:warfnt>>        violates c
28:[agr(+),plu(+),pret(+),two(+)]:warfnst>>       violates c
30:[agr(+),plu(+),pret(+),two(+)]:werftet>>       violates b/c
31:[agr(+),plu(+),pret(+),two(+)]:werftent>>      violates b/c
32:[agr(+),plu(+),pret(+),two(+)]:werftenst>>     violates b/c
```

Recall that *Uniqueness* requires exactly one form to occupy a cell, so that all other forms are rejected. In our implementation *Uniqueness* comes for free, as the natural use of monotonic unification in a PROLOG environment suggests modelling cell-insertion as (term) unification, which by definition observes the property labelled *Uniqueness* in MM. For clarity, PM monitors such unification failures during insertion attempts, expressing them by machine messages such as the one in (44), which shows the example of the rejection of *warfnt*.

(44)
```
here (warfnt,was blocked by (warft,
at ([agr(+),plu(+),pret(+),subj(-),two(+)])))),
```

Finally**,** the PROLOG term shown in (45a) describes the completed preterite non-subjunctive paradigm illustrated graphically in (45b)[19].

(45)a. paradigm([plu,two],*warf,warfn,warfst,warft)*

b. The [+pret, -subj] paradigm:

	+pl	-pl
+2	warft	warfst
-2	warfn	warf

[19]The paradigm depicted is a projection of the actual 5-dimensional paradigm displayed in Appendix 5. As can be verified by comparing (45a) against (45b), a vector d1d2 ...dn of binary feature dimensions (e.g. [-pl, +2]) is taken to denote a number in a base 2 coding, with d1 being the least significant bit. A minus value corresponds to a 0 bit, a plus value to a 1 bit. The resulting binary number +2 (to take into account the list of dimensions stored in argument 1, [plu, two] in our example) denotes the argument number of the paradigm term. In our example, [-pl, +2] is 01 in reverse (the least significant bit is leftmost) binary notation, equivalent to 2 in the decimal number system, therefore adressing argument cell 4 after adding 2, which is indeed occupied by *warfst* [-pl, +2].

One aspect yet to be modelled is also evident from the forms under (45): the final analysis would need to incorporate the phonological regularities of German, as can be seen by the lack of schwa in *warfn*, [vaʁfən]. Subjunctive 1st person plural *würfen,* however, displays schwa in our representation (orthographic <e>, cf. Appendix 1). Schwa in this case can be seen as constituting a defining property of the subjunctive as a whole. Whereas the schwa in *warfn* can be thought of as being epenthetic, notice that subjunctive forms of a regular verb like *lachen* would require an excessive schwa to be elided *(lachteet* > *lachtet*, see Appendix 5). However, a more elegant solution could involve viewing the required presence of schwa as a constraint in both cases, along the lines of (Neef forthcoming). When couched within the framework of Declarative Phonologie (Scobbie 1991), this would be compatible with our present monotonic and monostratal conception of inflectional morphology, thus allowing the two domains of phonology and morphology to be easily integrated. However, the actual implementation of phonological matters goes beyond the scope of this paper.

6. A comparison with Network Morphology

Having described in some detail the implementation of MM, we would now like to compare our approach to that of *Network Morphology* (NM, Corbett and Fraser 1993, Fraser/Corbett (to appear)). NM is one of the few other approaches we know of that attempts to combine linguistically interesting analyses of inflectional morphology with formal rigour and a concrete computational model. They present analyses of Russian inflectional morphology and implement their model as a DATR theory (see Evans/Gazdar 1990). DATR is a lexical knowledge description language which uses non-monotonic default inheritance to concisely specify hierarchically structured lexical knowledge. A grammar written in DATR is also known as a DATR theory and in response to a DATR query the inferences to be drawn according to the formal rules of inference yield DATR theorems. A given query never results in more than one answer, hence DATR is said to be deterministic. Numerous computer implementations of the DATR formalism itself exist (e.g. Kilbury 1990b).

While having a computational model is highly desirable, the first problematic aspect of NM lies in the fact that apparently no further structuring is imposed on the set of possible NM grammars (apart from the trivial requirement to use only valid DATR expressions). In contrast, our approach requires the CUF grammar (CG) to conform to the format and content expected by the Paradigm Mechanism (PM), the latter being inalterable modulo some parameter settings. Thus, we would claim that a significant substantive restriction is imposed on possible morphological descriptions.

The most important difference, however, lies in the fact that MM actually incorporates a kind of automatic learning mechanism which is absent from NM. This is the case precisely because our paradigm tool automatically organises the unstructured set of its input (word forms) into paradigms that are structured according to the principles discussed in this paper. Crucially, these principles are a formalised and implemented part of the MM model. In contrast, NM seems to be more concerned with representing morphological forms in an insightful way, with any principles that guide an adequate and non-redundant representation residing in the theorist's head. Hence, testing and rigid falsification of the principles *themselves* in their potentially complex interaction is a practical possibility only in the sort of setting we have described here.

7. Conclusion

We have shown that the MM model described in Wunderlich/Fabri (1994) can be explicated to a sufficient degree so as to allow formalisation and implementation in a declarative framework (modulo the paradigm mechanism). The mechanism designed for the generation of inflectional forms has been implemented in a CUF grammar (CG) and is local and strictly monotonic in character. Due to the paradigm mechanism, however, the MM model as a whole has a global and non-monotonic flavour. This aspect is dealt with by viewing the paradigm mechanism as a sort of learning device that organises a fixed set of items once and for all, and by having this information transferred back to CG. We, thus, end up with an enriched but still monotonic and local CG´.

With respect to the questions formulated in section 2, the following brief remarks are in order. A possible reduction of various principles related to the notion of specificity in MM (Affix Order, Strict Monotonicity, Non-Redundancy) to scale ordering as implemented in CG was identified. Thus, a certain degree of *redundancy* present in the original conception has been removed. With regard to *explicitness* we might mention the need to further specify 3rd person singular *-t* as discussed in section 4.7. Another instance involves the status of MM principles with regard to their proper place of application, either in CG or in PM, which has been made fully explicit in the process of implementation. *Formalisation* within CUF *and* Prolog was found to be the most adequate solution to cover all the requirements of the model. Finally, turning to *consistency,* again the necessary enrichment of 3rd person singular *-t* by the additional [-2] feature in its output was found to be inconsistent with a literal interpretation of the principle of *Underspecification* (`The output contains only "+" features´, see (7bi) and the discussion in section 4.7).

A documentation of the computational side of our co-operative effort in terms of a formal CUF grammar, Paradigm Mechanism source code and example sessions using our tools can be found in full length in Appendix 3-5. Upon request it can also be obtained from the authors in electronic form.

References

Anderson, S.R. (1988) Morphology as a Parsing Problem. *Linguistics* 26, 521-544

Anderson, S.R. (1992) *A-Morphous Morphology*. Cambridge: Cambridge University Press

Barg, Petra (1994) Automatic Acquisition of DATR Theories from Observations. *Working Papers of SFB 282 "Theorie des Lexikons"*, 59. University of Duesseldorf

Bird, Steven (1991) Feature Structures and Indices. *Phonology* 8,137-144

Bybee, Joan (1985) *Morphology: A Study of the Relation between Meaning and Form*. Amsterdam: Benjamins

Cartwright, T.A. & M.R. Brent (1994) Segmenting speech without a lexicon: The roles of phonotactics and speech source. *Proceedings of the First Meeting of the ACL Special Interest Group in Computational Phonology.* Las Cruces, New Mexico, USA. 38-90

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

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

Chomsky, Noam (1994) A minimalist program for linguistic theory. *MIT Occasional Papers in Linguistics 1*. Cambridge, Mass.: MITWPL

Chomsky, Noam & Morris Halle (1968) *The Sound Pattern of English*. New York: Haper and Row

Corbett, Greville & Norman M. Fraser (1993) Network Morphology: A DATR account of Russian inflectional morphology. *Journal of Linguistics* 29, 113-142

Culy, C. (1985) The complexity of the vocabulary of Bambara. *Linguistics and Philosophy* 8, 345 -351

Di Sciullo, Anna-Maria & Edwin Williams (1987) *On the Definition of Word*. Cambridge, MA: MIT Press

Dörre, J. & M. Dorna (1993) CUF - A Formalism for Linguistic Knowledge Representation. In: J. Doerre (ed.) *Computational Aspects of Constraint-Based Linguistic Description I*, DYANA 2 deliverable, University of Stuttgart

Engelkamp, J., G. Erbach & H. Uszkoreit (1992) Handling Linear Precedence Constraints by Unification. *Proceedings of the 30th Annual Meeting of the ACL,* Newark/Delaware, 201-208

Evans,R. & G. Gazdar (1990): The DATR Papers: February (= Cognitive Science Research Paper 139), School of Cognitive and Computing Sciences, University of Sussex, Brighton, England

Fähnrich, Heinz (1987) *Kurze Grammatik der georgischen Sprache.* Leipzig: Enzyklopädie

Fraser, Norman M. & Greville Corbett (to appear) Gender, animacy, and declensional class assignment: a unified account for Russian. *Yearbook of Morphology* 1994

Gazdar, G. (1992): Ceteris Paribus. In: C. Rohrer & H. Kamp (eds.) *Aspects of Computational Linguistics: Syntax, Semantics, Phonetics*. Springer, Berlin

Gazdar, G., E. Klein, G. Pullum & I. Sag (1985) *Generalized Phrase Structure Grammar*. Oxford: Blackwell

Gibbon, D. (1992): ILEX: a linguistic approach to computational lexica. In: U. Klenk (ed.) *Computatio Linguae. Zeitschrift für Dialektologie & Linguistik Beiheft* 73:32-53. Franz Steiner, Stuttgart

Goodman, N. (1943) Simplicity of Ideas. *Journal of Symbolic Knowledge* 8, 107-127

Halle, Morris & Alec Marantz (1993) Distributed Morphology and the pieces of inflection. In: K. Hale & S.J. Bromberger (eds.) *The View from Building 20: Linguistics in Honor of Sylvain Bromberger*. Cambridge. MA: MIT Press, 111-176

Halle, Morris & Alec Marantz (1994) Paper delivered at the Symposium on Distributed Morphology. LAS Annual Meeting , Boston, January 1994

Haspelmath, Martin (1993) The diachronic externalisation of inflection. *Linguistics* 31, 279-309

Johnson, Mark (1988) Attribute-Value Logic and the Theory of Grammar. *CSLI Lecture Notes* 16, Center for the Study of Language and Information, Stanford, CA

Kilbury, James (1990a) QPATR and Constraint Threading. *Proceedings of COLING'90,* 382-384, Helsinki

Kilbury, James (1990b) QPATR implementation. Unpublished software program, Seminar f. Allgemeine Sprachwissenschaft, University of Düsseldorf

Kiparsky, Paul. (1982) From cyclic phonology to lexical phonology. In: H. van der Hulst & N. Smith (eds.) *The Structure of Phonological Representation (Part I)* Dordrecht: Foris

Kisseberth, C.W. (1970). On the functional unity of phonological rules. *Linguistic Inquiry* 1, 291-306

Klein, E. (1993) An HPSG Approach to Sierra Miwok Verb Stems. In: Ellison, T.M. & J.M. Scobbie (eds.) *Computational Phonology, Edinburgh Working Papers in Cognitive Science* 8:19-35.Centre for Cognitive Science, Edinburgh

König, E. (1994) A Study in Grammar Design. *Arbeitspapiere des Sonderforschungsbereichs 340 "Sprachtheoretische Grundlagen fuer die Computerlinguistik"* 54, Universität Stuttgart

Koskenniemi. K. (1983) Two-Level morphology: a general computational model for word form recognition and production. Ph.D. thesis, Dep. of General Linguistics, University of Helsinki, Publication No.11

Langer, H. (1994) Reverse Queries in DATR. *Proceedings of COLING '94*, Vol. 2, 1089-1095

Lieber, Rochelle (1992) *Deconstructing Morphology: Word Formation in Syntactic Theory.* Chicago: The University of Chicago Press

Moser, L. (1992) Simulating Turing Machines in DATR. *Cognitive Science Research Paper CSRP 241*, University of Sussex, Brighton

Neef, M. (forthcoming) *Wortdesign.* Ph.D. Thesis, University of Cologne, Germany

Pollard, Carl & Ivan Sag (1987) *Information-Based Syntax and Semantics. Vol. 1*. Stanford: CSLI

Pollard, Carl & Ivan Sag (1994) *Head-Driven Phrase Structure Grammar*. Chicago:The University of Chicago Press

Riehemann, Susanne (1993) Word Formation in Lexical Type Hierarchies - A Case Study of bar-Adjectives in German. *SfS-Report-02-93*, Seminar für Sprachwissenschaft, University of Tübingen

Scobbie, J. (1991): Towards Declarative Phonology. In: S. Bird (ed.) *Declarative Perspectives on Phonology, Working Papers in Cognitive Science* 7:1-26. Centre for Cognitive Science, Edinburgh

Shieber, Stuart. (1986): *An Introduction to Unification-Based Approaches to Grammar.* CSLI Lecture Notes 4, Stanford

Wiese, Richard (in press) *The Phonology of German*. Oxford University Press

Wolfart, H.C. (1973) *Plains Cree: A Grammatical Study*. Transactions of the American Philosophical Society, Vol. 63, Part 5. Philadelphia: American Philosophical Society

Wunderlich, Dieter (1992) A minimalist analysis of German verb morphology. *Working Papers of SFB 282 "Theorie des Lexikons"*, 21. University of Düsseldorf

Wunderlich, Dieter (1993) Funktionale Kategorien im Lexikon. In: F. Heckmann & G. Heyer (eds.) *Theorie und Praxis des Lexikons*. Berlin: de Gruyter, 54-73

Wunderlich, Dieter & Ray Fabri (1994) Minimalist Morphology: an approach to inflection. Ms., University of Düsseldorf

APPENDIX 1: Paradigm of German verb inflection (*werfen* 'throw')

+pret				
			+pl	**- pl**
+agr	**+subj**	**+2**	*wuerfet*	*wuerfest*
		-2	*wuerfen*	*wuerfe*
	- subj	**+2**	*warft*	*warfst*
		-2	*warfn*	*warf*
- agr	**+ subj**	**+2**	+pret -> + agr	+pret -> + agr
		- 2	+pret -> + agr	+pret -> + agr
	-subj	**+2**	+pret -> + agr	+pret -> + agr
		- 2	+pret -> + agr	+pret -> + agr

- pret								
			+pl		**- pl**			
+agr	**+subj**	**+2**	*werfet*		*werfet*			
		-2	*werfen*		*werfe*			
	- subj	**+2**				**+1**	**- 1**	
			+C	*werft*	**+C**	+2 -> -1	*wirf*	
			- C	*werft*	**- C**	+2 -> -1	*wirfst*	
		-2	*werfn*		**+1**	*werfe*		
					- 1	*wirf*		
-agr	**+ subj**	**+2**	+subj -> + agr		+subj -> + agr			
		- 2	+subj -> + agr		+subj -> + agr			
	- subj	**+2**	+plu -> + agr		+2 -> + agr			
		- 2	+plu -> + agr		**+ part**	*warfn*		
					- part	*werf*		

APPENDIX 2: Transfer of paradigm information into CG´

Recursive paradigms basically fulfill two purposes: on the one hand they organise an unstructured set of morphological forms in a hierarchical and - in our case - principle-guided way. On the other hand they precisely circumscribe the set of morpheme combinations and their linear precedence relations. It appears that we only need the second type of information in CG' to ensure that all and only the wellformed word forms are combinable by the MCO. Let us assume for the moment that the exclusion of inappropriate instances in the case of homophonous affixes (e.g. *-t*) is taken care of by independently needed morphological content which must be unifiable with root requirements, thus blocking the choice of an inappropriate morph. In that case it is sufficient to state *continuation classes* (akin to the terminology used in the Two-Level Morphology conception of lexicon structuring, cf. Koskenniemi 1983) as a special kind of phonological subcategorisation for roots, which denote the set of phonological strings that may follow. Should the reduction to phonologically defined continuation classes be shown to be somewhat deficient in the general case, we would propose refining the classes with whatever non-phonological information renders the affixes in question distinctive, such as morphological dimensions present in the paradigm.

An example for the phonological continuation information extractable from a paradigm is given in (1).

(1)

Root	may be followed by
warf	-n, -t, -st, 0
wirf	-st, 0
werf	-e, -n, -e-st, -e-t
wuerfe	-n, -t, -st, 0
worf	-n

(1) describes the combinatory potential of the roots of *werfen*, 'throw'. Note that the non-combination state of affairs is expressed by 0 (a purely descriptive device not to be misunderstood as an explicit zero affix), and that iterated combinations have to be accounted for as well (*-e-st, -e-t*).

The encoding of (1) that we envisage proceeds in terms of type declarations and appropriateness restrictions which constrain the admissible values of morpheme positions in the list of phonological material constituting a word, as shown in the CUF fragment in (2) below.

(2)

```
%%% general reflex of paradigm structure projected back as
%%% continuation class encodings anchored to roots
%%% N.B. hd = first element of a list, tl = the rest of a list

paradigm::  hd:root_phon_string,          % it's a root
            tl:may_follow_root.           % possible continuations

may_follow_root = {null}                  % empty list, that is 0, no
                                          % possible continuation,
                  | follows_root          % alternatively at least one affix



follows_root::    hd:affix_phon_string,   % its an affix
                  tl:may_follow_root.     % recurse on possibly more
                                          % affixes
```

```
%% specific instantiations linked to the above general type structure
%% by subtyping (<),
%% here for German werfen 'to throw'

worf_paradigm < paradigm.
worf_paradigm:: hd:worf, hd:follows_worf.

follows_worf < follows_root.         % no bare root!
follows_worf::    hd:n,              % worf-n is OK
                  tl:null.           % empty list =
                                     % no further affixation!

werf_paradigm < paradigm.
werf_paradigm:: hd:werf, tl:follows_werf.

follows_werf < follows_root.         % no bare root!
follows_werf::    hd:(n;t),          % werf-n OR werf-t possible (;)
                  tl:null.           % no further affixation!

follows_werf::    hd:e, tl:(null;follows_werfe).
follows_werfe::   hd:(st;t), tl:null.

wirf_paradigm < paradigm.
wirf_paradigm::   hd:wirf, tl:(null;follows_wirf).
follows_wirf::    hd:st, tl:null.

%%%%%%%% end of CUF fragment
```

Now, if one tried to evaluate the description `[werf,st]&werf_paradigm` the CUF system would be able to infer its inconsistency, given the type declarations in (ii). Yet the set of type declarations is to be thought of as the result of a mechanical conversion process of paradigms, yielding suitable types which can be added to the original CUF grammar CG. Of course, the type specifications have to be conjoined with the individual roots, e.g. `verbal_root(werf,'WERF')& phon:werf_paradigm.` Hence, once this enrichment of CG has been carried out, the modified grammar CG' possesses the same attractive computational properties as CG with respect to declarativity, reversibility etc., with the added value of incorporating paradigm-based combinatorial information that bans all overgeneration present in the original CG. To put it differently, by this move we have been able to convert the *explicit parallelism* embodied in the PM into an *implicit parallelism* present in the combinatorial restrictions imposed by continuation classes-as-types. Note finally that the context-dependent feature-filling that results from PM operation as well still needs to be addressed.

Appendix 3: The CUF grammar for German verb inflection (CG)

```
%%%
```

```
person_number_index::  per:person,
                       num:numb.

person::          one:value,        % 1st person
                  two:value.        % 2nd person

%% FCR: express negative  *[+1,+2] by stating appropriateness of positively defined  one/two combinations

person = first | second | third.
first::           one:plus, two:minus.
second::          one:minus, two:plus.
third::           one:minus, two:minus.
numb::            plu:value.        % plural

%% crossclassification of binary values {+,-} x origin of value, i.e. {underlying, derived}
%% Rests on values being used in a mutually exclusive fashion in affixes, i.e.,
%% no underlying value is converted to derived and vice versa. This is indeed a property
%% of MM analyses sofar.

value = underlying | derived.
value = plus | minus.
plus  = {underlying_plus, derived_plus}.
minus = {underlying_minus, derived_minus}.
underlying = {underlying_minus , underlying_plus}.
derived = {derived_minus , derived_plus}.

%%% non-linguistic technical markers not present in result structures

technical_markers < afs.
technical_markers = scale_position | combinator_state.

% scale positions encode the degree of morphological specificity in
% feature descriptions and are computed automagically by the paradigm tool.
%       |          |          |        |          |          |          |
%       V       ASPECT     TENSE   MOOD      NUMBER     PERSON      C
%                                                 /\
%                                              ONE TWO

scale_position = real_scale_position | {outside_scale}.
real_scale_position = '>=V'.

'>=V'          =      '>=ASPECT'|  {'V'}.
'>=ASPECT'     =      '>=TENSE'|  {'ASPECT'}.
'>=TENSE'      =      '>=MOOD'   |  {'TENSE'}.
'>=MOOD'       =      '>=NUMBER'|  {'MOOD'}.
'>=NUMBER'     =      '>=ONE'    |  {'NUMBER'}.
'>=ONE'               =      '>=TWO'    |  {'ONE'}.
'>=TWO'        =      '>=C'      |  {'TWO'}.
'>=C'          =      {'C'}.


%%% distinguish states of combinating machine


combinator_state = {start_with_a_root, incorporate_affix}.
```

```
%%%
% MACROS -- (CUF terminology: SORTS) -- PATR-II terminology: TEMPLATES
%
% N.B. macros are not *exactly* the same as sorts, but for present purposes
% the difference can be ignored.
%
% format: name(type of arg.1,  ..., type of arg. n) -> type of arg.0.
%       name(arg1, ..., arg n) := <Defining Body Of Macro, i.e.,
%       feature descriptions of arg. 0 >.
%       all arg i must contain valid feature descriptions.
%%%

%%% generic root and affix templates

verbal_root(root_phon_string, non_empty_semantics) -> root.
verbal_root(PhonString, Semantics) :=
      phon:[PhonString|_PlaceHolderForMorePhonologicalMaterial] &
      sem:Semantics &
      verb.

verb -> base_entry.
verb := syn:maj:(n:underlying_minus &
            v:underlying_plus).

affix(affix_phon_string) -> affix.
affix(PhonString) :=
      phon:[PhonString] &
      verb.

%%% abbreviations for common (morphological) feature specifications

minor(minor_features) -> base_entry.
minor(Features) := syn:min:Features.

not_agr(Type)           := minor(part:(minus & Type)).
part(Type)              := minor(part:(plus & Type)).
pret(Type)              := minor(pret:(plus & Type)).
not_pret(Type)          := minor(pret:(minus & Type)).
subj(Type)              := minor(subj:(plus & Type)).
not_subj(Type)          := minor(subj:(minus & Type)).
c(Type)                 := minor(c:(plus & Type)).
not_first_per(Type)     := minor(inx:per:one:(minus & Type)).
not_sec_per(Type)       := minor(inx:per:two:(minus & Type)).
first_per(Type)         := minor(inx:per:one:(plus & Type)).
sec_per(Type)           := minor(inx:per:two:(plus & Type)).
plu(Type)               := minor(inx:num:plu:(plus & Type)).
not_plu(Type)           := minor(inx:num:plu:(minus & Type)).
agr(Type)               := minor(agr:(plus & Type)).
not_agr(Type)           := minor(agr:(minus & Type)).

output_minus            := derived_minus.
output_plus             = derived_plus.
output                  := derived_plus.
input_plus              := underlying_plus.
input_minus             := underlying_minus.
input                   := value.
```

```
%%%
% SUPPORT DEFINITIONS
%%%

true(_) := _.   % evaluate consistency of arg.1, but insert empty feature
            % structure via arg.0, i.e., ... := _Empty.

%% tabulate (SP-------
%%              I
%% facts, where SP is a point on the morphological scale and
%% I is the interval of points strictly greater than SP
%% Used to implement ranking/affix order in the morphological combination operation

next_position(scale_position) -> scale_position.

next_position('V')  := '>=ASPECT'.
next_position('ASPECT') := '>=TENSE'.
next_position('TENSE') := '>=MOOD'.
next_position('MOOD') := '>=NUMBER'.
next_position('NUMBER') := '>=ONE'.
next_position('ONE') := '>=TWO'.
next_position('TWO') := '>=C'.
next_position('C') := outside_scale.

%%%
% LEXICON
%%%

%%% REGULAR WEAK GERMAN VERB lachen 'to laugh'

lex('V') := verbal_root(lach, 'LACH').

%%% IRREGULAR STRONG  GERMAN VERB werfen 'to throw'

lex('V')  := verbal_root(werf,  'WERF').
lex('ONE') := verbal_root(wirf,  'WERF')          & not_first_per(input_minus)
                                                  & agr(value)
                                                  & not_pret(value)
                                                  & not_subj(value)
                                                  & not_plu(value).

lex('C') := verbal_root(wirf,  'WERF')            & not_first_per(input_minus)
                                                  & c(input_plus) & sec_per(input_plus)
                                                  & not_pret(value)
                                                  & not_subj(value)
                                                  & not_plu(value)
                                                  & agr(value).

lex('TENSE') := verbal_root(warf,  'WERF')        & pret(input_plus)
                                                  & agr(value).

lex('ASPECT') := verbal_root(worfn, 'WERF')       & part(input_plus)
                                                  & not_agr(input_minus).

lex('MOOD') := verbal_root(wuerfe, 'WERF')        & pret(input_plus)
                                                  & subj(input_plus)
                                                  & agr(value).

%%% GERMAN INFLECTIONAL SUFFIXES

lex('C') := affix(e)                              & c(output)       % IMPERATIVE SG
                                                  & not_pret(input)
```

```
                                & not_subj(input)
                                & sec_per(output)
                                & agr(value).

lex('TENSE') := affix(te)       & pret(output)     % PRETERITE
                                & agr(value).

lex('MOOD') := affix(e)         & subj(output)     % SUBJUNCTIVE
                                & agr(value).

lex('TWO') := affix(t)          & plu(output)      % 2PL
                                & sec_per(output)
                                & agr(value).

lex('NUMBER') := affix(n)       & plu(output)      % 1PL OR 3PL
                                & agr(value).

lex('TWO') := affix(st)         & sec_per(output)  % 2SG
                                & agr(value).

lex('ONE') := affix(e)          & not_pret(input)  % 1SG
                                & not_subj(input)
                                & not_plu(input)
                                & first_per(output)
                                & agr(value).

lex('C') := affix(t)            & c(output)        % IMPERATIVE PL
                                & plu(output)
                                & not_pret(input)
                                & not_subj(input)
                                & sec_per(output)
                                & agr(value).

lex('ASPECT') := affix(t)       & part(output)     % PARTICIPLE
                                & not_agr(output_minus).


lex('TWO') := affix(t)          & not_pret(input)  % 3SG
                                & not_sec_per(output_minus)
                                & not_first_per(minus)
                                & not_subj(input)
                                & not_plu(input)
                                & agr(value).

 lex('ASPECT') := affix(n)      &  not_agr(output_minus).      %  INFINITIVE
```

```
%%%
% MORPHOLOGICAL COMBINATION 'OPERATION'
%%%

mc := morph_combine(start_with_a_root, _PhonologyIrrelevantHere, _ScalePositionIrrelevantInThisCase).

morph_combine(combinator_state, list, scale_position) -> base_entry.          % list is a predefined type

morph_combine(start_with_a_root, _PhonologyIrrelevantHere, _ScalePositionIrrelevantInThisCase) :=
      is_a_root(MinScalePosForFirstAffix, Affixes) &
      morph_combine(incorporate_affix, Affixes, MinScalePosForFirstAffix).

morph_combine(incorporate_affix, [Affix|MoreAffixes], ScalePosition) :=      % yes, we have at least one
                                                                             % affix
      is_an_affix(Affix, ScalePosition, NextPosition) &
      morph_combine(incorporate_affix, MoreAffixes, NextPosition).

morph_combine(incorporate_affix, [], _ScalePositionIrrelevantInThisCase) := _.  % no more affixes, hence
                                                                                % empty list []

%%% support templates for root and suffix 'access'

is_a_root(scale_position, list) -> root.

is_a_root(next_position(ScalePosition), Suffixes) :=
      lex(ScalePosition) &                                  % it's a lexical entry
      phon:[root_phon_string | Suffixes].                   % with a root phonological string which
                                                            % gets extended by suffixes to be unified in
is_an_affix(affix_phon_string, scale_position, scale_position) -> root.

is_an_affix(AffixPhon, OldPosition, next_position(OldPosition)) :=
      true(                                                 % it is true, .i.e. consistent, that
          lex(OldPosition) &                                        % there is a lexical entry
          affix &                                           % which has the affix property
          phon:[AffixPhon] &                                % with AffixPhon segmental string
          syn:GramFeats) &                                          % and whose syntactical features are all
that is
      syn:GramFeats.                                        % structure-shared with the word feature structure
                                                            % that we are building up right now
%%%
% MECHANICAL TEST OF THE MM MODEL:
% ================================
%
% Use ?- run(werfen).
%
% or e.g.
%
% ?- cuf(mc & phon:[wirf,st]).
%
% to experiment with the grammar alone.
%
% Use ?- cuf2paradigm(test(werfen)), show.
%
% to run grammar+paradigm tool and show resulting forms.
%%%

test(werfen) := mc & sem:'WERF'.
test(lachen) := mc & sem:'LACH'.
```

```
%%%
```

APPENDIX 4: The source code of the Paradigm Mechanism (PM)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                                                                         %%%
%%%    File: pm.pl                                                          %%%
%%%    Author: Markus Walther                                               %%%
%%%    Purpose: compute paradigms from morphological                        %%%
%%%                specifications (PM)                                      %%%
%%%    Created: 8.Maerz 1994                                                %%%
%%%    Modified: 08.Februar 1995                                            %%%
%%%    Copyright: Seminar fuer Allgemeine Sprachwissenschaft                %%%
%%%    Universitaet Duesseldorf,1995                                        %%%
%%%                                                                         %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%% INITIALIZATION

:- initialization(info).

info :-

        write('
Paradigm Mechanism (PM) Services
--------------------------------
?- cuf2paradigm(CUFExpression /* e.g. test(werfen) */)
?- pmflags. /* shows user-accesible PM flags, i.e. parameters */
?- pmflag(FlagName,FlagValue). /* Set flag to value */
?- info. /* this info */

'),
        pmflags.

:- ensure_loaded(library(basics)).

%%% SUPPORT DEFINITIONS

%% OPERATOR DECLARATIONS

:-op(800,xfx,if_error).
:-op(800,xfx,if_fatal_error).
:-op(800,xfx,if_warning).
:-op(800,xfx,if_info).

%% OTHER DECLARATIONS

:- dynamic '$pmflag'/2, check_FCRs_at/2.

%% MORE STUFF...

always.

%% ERROR MESSAGES

% adapted from code by ALE, (Bob Carpenter/Gerald Penn)

% if condition Cond holds, provides Msg as error message;  always succeeds
% ---------------------------------------------------------------------------
```

```
if_message_generic(MsgType, Msg, Cond) :-
  ( call(Cond) -> ttynl,
      write_list([MsgType|Msg]),
      ttyflush,
      ( ( MsgType = '!',  clause('$pmflag'(stop_on_error,yes),true)
        ; MsgType = '!!'
        ) -> /* put finalizing actions here */
             raise_exception('Paradigm tool cannot recover from errors.
! Please try to fix them before continuing.')
      ; true
      )
  ; true
  ),
  !.

% if_warning(Msg,Cond)
% ------------------------------------------------------------------------
% if condition Cond holds, prints out Msg;  always succeeds
% ------------------------------------------------------------------------
if_warning(Msg, Cond) :-
        if_message_generic('*', Msg,Cond).
if_error(Msg, Cond) :-
        if_message_generic('!', Msg, Cond).
if_fatal_error(Msg, Cond) :-
        if_message_generic('!!', Msg, Cond).
if_info(Msg, Cond) :-
        if_message_generic('%', Msg, Cond).

% write_list(Xs)
% ------------------------------------------------------------------------
% writes out elements of Xs with spaces between elements
% ------------------------------------------------------------------------

write_list([]).
write_list([X|Xs]) :-
        ( nonvar(X), functor(X, ',', 2) -> call(X)
        ; write(X), write(' ')
        ),
        write_list(Xs),
        !.

% PMFLAGS
% ------------------------------------------------------------------------

%%%  pmflag(+Name, +Value)
%%%  pmflag value setting

pmflag(Name, Value) :-
   ( atom(Name) ->
         ( nonvar(Value) ->
               ( valid_pmflag_value(Name, Value) ->
                     set_pmflag(Name, Value)
               ; [undefined,pmflag,Name,or,invalid,value,Value,(nl,nl)]
                 if_error  always
               )
         ;  [variable,value,for,Name,not,allowed, (nl,nl)] if_error always
         )
   ;  [pmflag,name,must,not,be,a,variable, (nl,nl)] if_error always
   ), !.
```

```
%%% pmflags
%%% shows pmflag settings
pmflags :-
        ['  Use ?- pmflag(Name, NewValue). to alter pmflags!',(nl,true)]
                                        if_info always,
        [' ',pmflag,name,' :',pmflag,value]      if_info always,
        ['  ----------------------']    if_info always,
        repeat,
        ( clause('$pmflag'(Name, Value),true),
          [' ',Name,':',Value] if_info always,
          fail
        ; nl, !
        ).

set_pmflag(Name, Value) :-
        retract('$pmflag'(Name,_)),
        assert('$pmflag'(Name,Value)),
        !.

%%% pmflag values
binary(yes).
binary(no).

%%% pmflags
'$pmflag'(stop_on_error, no).
'$pmflag'(block_on_fcr_hit, yes).
'$pmflag'(show_order, yes).
'$pmflag'(number_of_retries, 50).
'$pmflag'(violatable_insertion_behaviour, yes).
'$pmflag'(scale_ordering, yes).
'$pmflag'(compute_scale_position, no).
'$pmflag'(verbose,no).
valid_pmflag_value(stop_on_error, Val) :- binary(Val).
valid_pmflag_value(block_on_fcr_hit, Val) :- binary(Val).
valid_pmflag_value(show_order, Val) :- binary(Val).
valid_pmflag_value(number_of_retries, Val) :- integer(Val).
valid_pmflag_value(violatable_insertion_behaviour, Val) :- binary(Val).
valid_pmflag_value(scale_ordering, Val) :- binary(Val).
valid_pmflag_value(compute_scale_position, Val) :- binary(Val).
valid_pmflag_value(verbose, Val) :- binary(Val).

%%%
% CUSTOM SET OPEERATIONS
%%

my_intersect(Set1, Set2) :-
        member(Element, Set1),              %  generates Elements from Set1
        map_extended_boolean(Element,       %  weaken element compatibility
                        Element1),          %  (see principleVI below)
        memberchk(Element1, Set2),          %  tests them against Set2
        !.                                  %  if it succeeds once, is enough.

%%%%%%
% REAL PARADIGM STUFF STARTS HERE
%%%%%%

boolean(+, 1).
boolean(-, 0).

clear_all_indices :-
        abolish(index/2).
```

```
% Assert index(Dimension,I) facts from Dimensions list in ascending numerical order starting from I.
% names_to_indices(+DimensionNamesWithoutValues,+StartIndex)
% Typically StartIndex = 1

names_to_indices([], _).
names_to_indices([Dimension|Dimensions], Position) :-
        assert(index(Dimension,Position)),
        Position0 is Position << 1,
        names_to_indices(Dimensions, Position0).

% compute index of paradigm term argument position from boolean-valued dimension list, adding offset
IndexIn.
% dimensions_to_index(+BooleanValuedDimensions, +IndexIn, -IndexOut)
% Typically IndexIn = 2

dimensions_to_index([], Index, Index).
dimensions_to_index([Dimension|Dimensions], Index0, Index) :-
        'dimension to index'(Dimension, Index1),
        Index2 is Index0 + Index1,
        dimensions_to_index(Dimensions, Index2, Index).

'dimension to index'(Dimension, Index) :-
        functor(Dimension, Name, 1),
        arg(1, Dimension, BooleanValue),
        index(Name, Index0),
        boolean(BooleanValue, Offset),
        Index is Index0*Offset.

% create new paradigm from list of its dimensions.
% new(+ParadigmDimensions, -ParadigmRepresentation)

new(Dimensions, Paradigm) :-
        sort(Dimensions, Dimensions0),      % remove duplicates and order
        length(Dimensions0, NumberOfDims),
        names_to_indices(Dimensions0, 1),
        Dimensionality is (1 << NumberOfDims) + 1, % Dim's are boolean-valued
        functor(Paradigm, paradigm, Dimensionality),
        arg(1, Paradigm, Dimensions0),
        !.

% attempt to insert datum into paradigm cell characterised by dimensions, verbose comments on
% sequence of internal operations returned as term returnCode.
% insert(+SortdBooleanValuedDimensions:DatumToBeInserted,+Paradigm,
%        -ReturnCode)

insert(Dimensions:Datum, Paradigm, GlobalParadigm, ReturnCode) :-
        nonvar(Datum),
        nonvar(Dimensions),
        is_list(Dimensions),
        ( var(Paradigm) ->                                  % empty cell -> create new paradigm
            convert_to_boolean_valued(Dimensions0, Dimensions),
            new(Dimensions0, Paradigm),
            ReturnCode = '
new_paradigm_created'(ReturnCode0),
            apply_FCRs(GlobalParadigm, []),
            insert(Dimensions:Datum, Paradigm, GlobalParadigm, ReturnCode0)
        ; nonvar(Paradigm),                                 % its a subparadigm
          arg(1, Paradigm, ParaDimensions),
          convert_to_boolean_valued(ParaDimensions, ParaDimensions0),
          set_difference(Dimensions, ParaDimensions0, InDimsNotInPara),
          set_difference(ParaDimensions0, Dimensions, InParaNotInDims),
          ( InParaNotInDims \== [] ->                       % need to guess missing values nondeterm.
```

```
            principleV(InParaNotInDims, InParaNotInDims1),          % suppress +, so it's deterministic again!
            ReturnCode = '
guessing_values_and_continuing'(InParaNotInDims1,
                                            ReturnCode0),
            append(Dimensions, InParaNotInDims1, Dimensions0),
            sort(Dimensions0, Dimensions1),
            insert(Dimensions1:Datum, Paradigm, GlobalParadigm,ReturnCode0)
        ; InParaNotInDims == [],                                % all paradigm-required dimensions supplied
                                                % now find cell with full paradigm dimensions specified
          set_difference(Dimensions, InDimsNotInPara, FullDimensions),
          dimensions_to_index(FullDimensions, 2, Index),
          arg(Index, Paradigm, Cell),
          ( var(Cell) ->
             ( InDimsNotInPara == [] ->                         % no extra dimensions
               Cell = Datum,
               ReturnCode = '
free_cell_now_occupied_with'(Datum,located_at,FullDimensions)
            ; InDimsNotInPara \== [],                           % excessive dimensions -> open subparadigm with
these
               ReturnCode = '
creating_subparadigm_at'(FullDimensions, ReturnCode0),
              insert(InDimsNotInPara:Datum, Cell, GlobalParadigm,
                   ReturnCode0)
             )
          ; nonvar(Cell) ->                                     % it's a subparadigm we're about to navigate into, so ...


            ( functor(Cell, paradigm, _DimPlusOne) ->
               ReturnCode = '
inserting_in_subparadigm_at'(FullDimensions, ReturnCode0),
               insert(InDimsNotInPara:Datum, Cell, GlobalParadigm,% ... recurse
                    ReturnCode0)
            ; \+ functor(Cell, paradigm, _),                    % it's an ordinary paradigm cell
              ( Cell = Datum ->                                 % and its contents are unifiable with Datum
                   ReturnCode = '
set_compatible_cell_at'(FullDimensions)
              ; \+ Cell = Datum,                                % and its contents are inconsistent with Datum
               ( '$pmflag'(verbose,yes) ->
                   write('
%% paradigm cell at'(FullDimensions,already_occupied_by,' '(Cell),'
%% which_is_incompatible_with',' '(Datum))),
                  write('
%% Therefore I did not insert '(Datum)),
                   nl
                ; !
                ),
               ( '$pmflag'(violatable_insertion_behaviour, yes) ->
                  ReturnCode = '
      here '(Datum,'was blocked by '(Cell,' at '(FullDimensions)))
                ; fail
                )
              )
             )
            )
         )
        ).

% convert list of atomic dimensions d_i to list of d_i/1 terms

convert_to_boolean_valued([], []).
convert_to_boolean_valued([In|Ins], [Out|Outs]) :-
      functor(Out, In, 1),
```

```
        convert_to_boolean_valued(Ins, Outs).

% the elements of Set2 that are not in Set1 appear as Diff
% set_difference(+Set1, +Set2, -Diff)

set_difference([], _, []).
set_difference([Ele|Elems], Set2, Diff) :-
        member(Ele, Set2),
        !, set_difference(Elems, Set2, Diff).
set_difference([Ele|Elems], Set2, [Ele|Diff]) :-
        set_difference(Elems, Set2, Diff).

% simple test for listhood

is_list([]).
is_list([_|_]).

% map In-list via one-place Predicate to Out-list
% maplist(+Predicate, +In, -Out)

maplist(_, [], []).
maplist(Predicate, [FirstIn|RestIns], [FirstOut|RestOuts]) :-
        Mapping =.. [Predicate, FirstIn, FirstOut],
        call(Mapping),
        maplist(Predicate, RestIns, RestOuts).


% map In-list via N-place [Predicate,Arg1,...,ArgN] to Out-list
% maplist(+PredicateFollowedByArgsAsList, +In, -Out)

maplist1(_, [], []).
maplist1([Predicate|Args], [FirstIn|RestIns], [FirstOut|RestOuts]) :-
        Mapping =.. [Predicate, FirstIn, FirstOut|Args],
        call(Mapping),
        maplist1([Predicate|Args], RestIns, RestOuts).

% assert a list of Terms

assert_list([]).
assert_list([Term|More]) :-
        assert(Term),
        assert_list(More).

%% ORDERING PRINCIPLES FOR PARADIGM CONSTRUCTION FROM ELEMENTS
%%
%% PRINCIPLE I. More dimensions >I less dimensions.
%% PRINCIPLE II. Higher dimension w.r.t. morphological scale >II lower dimensions.
%%             Can be turned off.
%% PRINCIPLE III. underlying >III derived values
%% PRINCIPLE IV. shorter phonological strings >IV to longer ones
%%
%% This is not really an ordering principle, but anyhow ...
%% PRINCIPLE V. plus (+) values must not be freely introduced.
%%
%% CENTRAL DATA STRUCTURES:
%% Elements = [Element|Elements]
%% Element  = BooleanValuedDimensionsList:ParadigmCellContent


%% principleI(+Element, -OrderedElement)
%% longer dimensions lists first - via list length/2
```

```
principleI(BVDList:ParadigmCell, Length-(BVDList:ParadigmCell)) :-
        length(BVDList, Length0),
        Length is -Length0. % reverse sign (-) for correct sorting order

%% principleII(+Element, -OrderedElement)
%% Highest scale positions first.
%% Implemented by A. scale definition facility define_scale/1 including a mapping
%% from scale dimensions to natural numbers. Higher positions carry lower numbers.
%% B. Making use of this mapping to augment Dimensions within Elements by their corresp.
%% nat. number scale position. Taking care of multiple dimensions is accomodated by sorting
%% the resulting vector of scale positions.

principleII(Length-(Dimensions:Cell),
             ScalePositionsVector-Length-(Dimensions:Cell)) :-
        ( '$pmflag'(scale_ordering,yes) ->
           ScalePositionsVector = ScalePositionsVector0
        ;  ScalePositionsVector = the_same_everywhere
        ),
        map_dimensions_to_scalepositions_vector(Dimensions, ScalePositionsVector0),
        ( '$pmflag'(compute_scale_position,yes) ->
           min(ScalePositionsVector0, MaxScalePosition),
           write('
%% scale position for '(Dimensions:Cell,'   is   ',MaxScalePosition))
        ; !
        ).


%% the minimum of a list of numbers is Minimum
%% min(+NonEmptyListOfNumbers, - Minimum)

min([Number|RestNumbers], Minimum) :-
        min1(RestNumbers, Number, Minimum).
min1([], Minimum, Minimum).
min1([Number|RestNumbers], PutativeMinimum, Minimum) :-
        ( number(Number), PutativeMinimum > Number -> min1(RestNumbers, Number, Minimum)
        ; min1(RestNumbers, PutativeMinimum, Minimum)
        ).

%% principleIII(+Element,-OrderedElement)
%% lexical before derived values.
%% Implemented by defining a mapping of
%% the crossproduct of {lexical,derived} x {+,-}
%% to integers (extended_boolean/2) and substituting
%% the conveniently defined shortnames for crossproduct elements
%% by their Integer x {+,-} counterparts. The mapping is defined so that
%% sort/2 sorts lexical (more negative) values in front of derived
%% values with neutral/underspecified w.r.t. lex./der. ones in between.

% map In version of Functor/1 to Out version differing in Argument supplied (ArgIn vs. ArgOut)

get_args(In, Out, ArgIn, ArgOut) :-
        functor(In,Functor,Arity),
        functor(Out,Functor,Arity),
        arg(1, In, ArgIn),
        arg(1, Out, ArgOut).

% map CUF version of binary value to PM internal representation which is more
% appropriate to sorting (-1 < 0 < 1)

extended_boolean(In, Out) :-
        get_args(In, Out, ArgIn, ArgOut),
        extended_boolean1(ArgIn, ArgOut).
```

```
extended_boolean1('(minus & underlying)', (-1,'-')).
extended_boolean1(underlying_minus, (-1,'-')).
extended_boolean1(derived_minus, (1, '-')).
extended_boolean1('(plus & underlying)', (-1,'+')).
extended_boolean1(underlying_plus, (-1,'+')).
extended_boolean1(derived_plus, (1, '+')).
extended_boolean1(minus,  (0, '-')).
extended_boolean1(plus,  (0, '+')).

map_extended_boolean(In, Out) :-
        get_args(In, Out, ArgIn, ArgOut),
        map_extended_boolean1(ArgIn, ArgOut).

map_extended_boolean1('(minus & underlying)', Result) :- map_extended_boolean1(underlying_minus,
Result).
map_extended_boolean1('(plus & underlying)', Result) :- map_extended_boolean1(underlying_minus, Result).
map_extended_boolean1(underlying_minus, underlying_minus).
map_extended_boolean1(underlying_minus, derived_minus).
map_extended_boolean1(underlying_minus, minus).
map_extended_boolean1(derived_minus, derived_minus).
map_extended_boolean1(derived_minus, underlying_minus).
map_extended_boolean1(derived_minus, minus).
map_extended_boolean1(minus, minus).
map_extended_boolean1(minus, derived_minus).
map_extended_boolean1(minus, underlying_minus).
map_extended_boolean1(underlying_plus, underlying_plus).
map_extended_boolean1(underlying_plus, derived_plus).
map_extended_boolean1(underlying_plus, plus).
map_extended_boolean1(derived_plus, derived_plus).
map_extended_boolean1(derived_plus, underlying_plus).
map_extended_boolean1(derived_plus, plus).
map_extended_boolean1(plus, plus).
map_extended_boolean1(plus, derived_plus).
map_extended_boolean1(plus, underlying_plus).

principleIII(OtherPrinciplesInfo-(DimensionsIn:Cell),
            OtherPrinciplesInfo-(DimensionsOut:Cell)) :-
        mapping(DimensionsIn, DimensionsOut).

mapping(DIn, DOut) :-
        maplist(extended_boolean, DIn, DOut).

%%% PRINCIPLE IV : SIMPLEX
%%% shorter phonological strings are preferred to longer ones
%%% Implemented by converting atomic phonstring values to
%%% PROLOG strings, counting their length (>= 0) and attaching
%%% this to the ordering info already present in the ordering term

principleIV(OrderingInfo-(Dimensions:PhonStringAtom),
             OrderingInfo-((Dimensions-PhonologicalLength):PhonStringAtom)) :-
        name(PhonStringAtom, PhonString),
        length(PhonString, PhonologicalLength).

remove_underlying_derived_distinction(DimensionsListIn, DimensionsListOut) :-
        maplist('remove lexical derived distinction', DimensionsListIn,
                                        DimensionsListOut).

'remove lexical derived distinction'(DimensionIn, DimensionOut) :-
        get_args(DimensionIn, DimensionOut, (_IgnoreNumericPart,PlusOrMinus),
              PlusOrMinus).
```

```
%% principleV(+BooleanValuedDimensionsIn, -MinusValuedDimensionsOut)
%% No plus values may be freely introduced.
%% Simple mapping to 1-arity terms containing - as value

principleV(BooleanValuedDimensionsIn, MinusValuedDimensionsOut) :-
        maplist(boolean_to_minus,
                BooleanValuedDimensionsIn, MinusValuedDimensionsOut), !.

% replace one-place Property by one-place minus-valued Property

boolean_to_minus(Property, MinusProperty) :-
        functor(Property, PropName, 1),
        functor(MinusProperty, PropName, 1),
        arg(1, MinusProperty, -).

order_info(_OI1-_OI2-((ListIn-_OI3):Cell), ListOut:Cell) :-
        remove_underlying_derived_distinction(ListIn, ListOut).




% filter out features containing a certain value specified by throw_out/1 facts

filter([], _, []).
filter([Feature|More], Phonology, Dimensions) :-
        arg(1, Feature, Value),
        ( throw_out(Value) ->
           ('$pmflag'(verbose,yes) ->
               write('
%% Dimension '(Feature,' has been removed from ',Phonology,' specifications!'))
             ; !
             ),
           Dimensions = Dimensions0
        ;   Dimensions = [Feature|Dimensions0]
        ), filter(More, Phonology, Dimensions0).

:- dynamic throw_out/1.  % actual throw_out facts are ususally supplied in CUF control file

put_dimensions_in_canonical_order(DimensionsIn:Phonology, DimensionsOut:Phonology) :-
        filter(DimensionsIn, Phonology, DimensionsIntermediate),
        sort(DimensionsIntermediate, DimensionsOut).

%% order the list of elements (i.e. wordform candidates) according to the principles,
%% doing some filtering before and throwing away the now superfluous ordering by
%% mapping through order_info afterwards. Ordering amounts to simple PROLOG sorting here.

order(ElementsIn, Elements) :-
        maplist(put_dimensions_in_canonical_order,ElementsIn, Elements1),
        maplist(principleI, Elements1, Elements2),
        maplist(principleII, Elements2, Elements3),
        maplist(principleIII, Elements3, Elements4),
        maplist(principleIV, Elements4, Elements5),
        sort(Elements5, Elements6),
        maplist(order_info, Elements6, Elements).

%% increment counter Countername by one

inc(CounterName) :-
        functor(Counter, CounterName, 1),
        ( clause(Counter, true) ->              % Counter exists,
```

```
          arg(1,Counter, Current),          % so increment it
          retract(Counter),
          New is Current + 1,
          functor(NewCounter, CounterName, 1),
          arg(1, NewCounter, New),
          assert(NewCounter)
        ; arg(1, Counter, 0),               % initialise non-ex.
          assert(Counter)                   % Counter to zero
        ), !.

%% current value of counter CounterName is greater than Value

greater_than(Value, CounterName) :-
        functor(Counter, CounterName, 1),
        clause(Counter, true) ->
                arg(1, Counter, Current),
                Current >= Value.




%% fail MaxNumberOfFail times, doing the counting via FailCounterName

fail(FailCounterName, MaxNumberOfFail) :-
        inc(FailCounterName),
        (greater_than(MaxNumberOfFail, FailCounterName) -> true
        ; fail
        ).

%% record all possible solutions of inserting Elements into a paradigm
%% as solution/2 facts in the PROLOG database, upper bound for this
%% being MaxAttempts, however.

all_solutions(Elements) :-
        abolish(solution/2),
        abolish(number_of_retries/1),
        '$pmflag'(number_of_retries, MaxAttempts),
        ( paradigm(Elements, Paradigm),
          record_if_different(Paradigm, IsDifferent),
          ( IsDifferent == true -> write(Paradigm)
          ; true
          ),
          fail(number_of_retries, MaxAttempts)
        ; true
        ),
        !.

% paradigm(+Elements, -Paradigm)
% make paradigm applying principles and appropriate insertions

paradigm(Elements, Paradigm*Histories) :-
        clear_all_indices,
        order(Elements, OrderedElements),
        ('$pmflag'(show_order,yes) ->
          write('

%% ordered paradigm cell candidates: '),
          write_ordered(OrderedElements)
        ; true
        ),
```

```
        do_inserts(OrderedElements, Paradigm, '', Histories).

% write ordered candidates with >> sign at end of line

write_ordered([H|T]) :-
        nl,
        write(H),
        ( T \== [] -> write(' >>'), write_ordered(T)
        ; nl
        ),
        !.

%% do insertion of ordered candidate list into a paradigm, connecting individual
%% (return code term-represented) histories of the insertions to a global History

do_inserts([], _Paradigm, Histories, Histories).
do_inserts([Element|Elements], Paradigm, Histories0, Histories) :-
        insert(Element, Paradigm, Paradigm, History),
        Histories1 = ''(Histories0, History),
        do_inserts(Elements, Paradigm, Histories1, Histories).


%%
%% SUPPORT FOR SCALE-DEPENDANT PARADIGM INSERTION ORDERING
%%

% define a scale of ordered (morphological) features.
% define_scale(+ListOfEquivalenceClassesOfDimensions)
% EquivalenceClassOfDimensions = [Dimension|(EquivalenceClassOfDimensions ; [])]

define_scale(ListOfEquivalenceClassesOfDimensions) :-
        abolish(scale/2),
        'define scale'(ListOfEquivalenceClassesOfDimensions, 0).

scale_info(Dimension, scale(Dimension, NatNumberAsPosition), NatNumberAsPosition).

'define scale'([], _).
'define scale'([EquivalenceClassOfDimensions|More], I) :-
        convert_to_boolean_valued(EquivalenceClassOfDimensions, BVDList),
        maplist1([scale_info,I], BVDList, ScaleList),
        assert_list(ScaleList),
        J is I+1,
        'define scale'(More, J).

% map morphological features/dimensions to positions on a scale
% map_dimensions_to_scaleposes(+BooleanValuedDimensionsList, -
ListOfNaturalNumbersDenotingScalePositions)

map_dimensions_to_scaleposes([], [undefined]).
map_dimensions_to_scaleposes([Dimension|MoreDim], [Position|MorePos]) :-
                        ( scale(Dimension, Position) -> true
                        ; write('
!! paradigm dimension '(Dimension,'not defined in current scale.')), abort
                        ),
                        map_dimensions_to_scaleposes(MoreDim, MorePos),
                        !. % no backtracking from outside

% map_dimensions_to_scalepositions_vector(+BooleanValuedDimensionsList, -ListOfSortedNaturalNumbers)

map_dimensions_to_scalepositions_vector(BooleanValuedDimensionsList, ScalePositionsVector) :-
        map_dimensions_to_scaleposes(BooleanValuedDimensionsList, UnsortedNatNumbersList),
        sort(UnsortedNatNumbersList, ScalePositionsVector).
```

```
%%%
% Feature Cooccurrence Restrictions HANDLING in Paradigms
%%%

% convert a numerically given address within a N-dimensional paradigm with given Dimensions to
% the corresponding +/- valued list of Dimensions.
% paradigm_address_to_binary_values(+NoOfDimensions, +DimensionList, +Address,
%                                   -BooleanValuedDimensionsOfAddress)

paradigm_address_to_binary_values(0, _, _, []).
paradigm_address_to_binary_values(BitPosition, [Dim|Dims], Address,
                                  [BoolValuedDim|BoolValuedDims]) :-
        NewAddress is Address div 2,
        Value is Address mod 2,
        NewBitPosition is BitPosition - 1,
        functor(BoolValuedDim, Dim, 1),
        arg(1, BoolValuedDim, BooleanValue),
        ( Value == 1 -> BooleanValue = '+'
        ; BooleanValue = '-'
        ),   paradigm_address_to_binary_values(NewBitPosition, Dims,  NewAddress, BoolValuedDims),  !.


% apply FCRs within a (sub)paradigm taking inherited dimensions from
% 'outer' paradigms into account when calculating whether to impose a
% FCR constraint at a particular (embedded) cell.
% apply_FCRs(?Paradigm, +InheritedBooleanValuedDimensions)

apply_FCRs(Paradigm, InheritedBooleanValuedDimensions) :-
        functor(Paradigm,paradigm,SizePlusOne),
        Size is SizePlusOne - 2, % account for extra Dim. slot in paradigm &
                                 % counting down to 0
        arg(1, Paradigm, Dimensions),
        length(Dimensions, NoOfDimensions),
        'apply FCRs'(Size, NoOfDimensions, Paradigm,
                     InheritedBooleanValuedDimensions),
        !.

'apply FCRs'(-1, _NoOfDimensions, _Paradigm,
             _InheritedBooleanValuedDimensions) :- !.

'apply FCRs'(CellAddress, NoOfDimensions, Paradigm,
             InheritedBooleanValuedDimensions) :-
        arg(1, Paradigm, Dimensions),
        paradigm_address_to_binary_values(NoOfDimensions, Dimensions,
                                          CellAddress, BoolValuedAddress),
        append(InheritedBooleanValuedDimensions, BoolValuedAddress,
               RealBooleanValuedAddress),
        RealCellAddress is CellAddress + 2,
        arg(RealCellAddress, Paradigm, Cell),
        !,
        ( nonvar(Cell), functor(Cell,paradigm,_) ->
                apply_FCRs(Cell, RealBooleanValuedAddress)
        ; check_FCRs_at(RealBooleanValuedAddress, Cell)
        ),
        NextCellAddress is CellAddress - 1,
        'apply FCRs'(NextCellAddress, NoOfDimensions, Paradigm,
                     InheritedBooleanValuedDimensions).

check_FCRs_at(_, _).  %% NO FCR's; this is the default if no explicit FCRs are specified.

%% install a list of FCRs
```

```
fcrs([FCR|FCRs]) :-
        abolish(check_FCRs_at/2),
        build_fcr_code([FCR|FCRs], BooleanValuedAddress, Cell, Body),
        assert((check_FCRs_at(BooleanValuedAddress, Cell) :- Body)).

%% build complex PROLOG body to be activated in later FCR application

build_fcr_code([], _, _, true).
build_fcr_code([Antecedent >> Consequent|MoreFCRs], BooleanValuedAddress,
                Cell, Goal) :-
        pnegate(Consequent, NegatedConsequent),
        Goal = ( member(Antecedent, BooleanValuedAddress),
                 member(NegatedConsequent, BooleanValuedAddress) ->
                      ( '$pmflag'(verbose,yes) ->
                           nl,
                           write('%% '),
                           write(BooleanValuedAddress),
                           write(' is blocked by FCR '),
                           write(Antecedent >> Consequent),
                           nl  ; !  ),
                      ( '$pmflag'(block_on_fcr_hit, yes) ->
                             Cell = '*'(Antecedent >> Consequent)
                       ; !
                       )
                    ; MoreGoals),
        build_fcr_code(MoreFCRs, BooleanValuedAddress, Cell, MoreGoals).

%% paradigm-related negation of a boolean-valued feature/dimension

pnegate(BooleanValuedDimension, NegatedDimension) :-
        BooleanValuedDimension =.. [Dimension, Value],
        ( Value == '+' -> NegatedValue = '-'
        ; NegatedValue = '+'
        ),
        NegatedDimension =.. [Dimension,NegatedValue].

%% record a solution of not already recorded up to a (neglectable) different history

record_if_different(Paradigm*Annotations, IsDifferent) :-
        ( clause(solution(Paradigm,_),true) -> IsDifferent = false
        ; assert(solution(Paradigm,Annotations)),
          IsDifferent = true
        ), !.


%%% cuf2paradigm(+TestSort,-Paradigm)
%%% Compute all solutions of TestSort, convert them into
%%% a flat representation suitable for the Paradigm Tool
%%% and let the Tool return the Paradigm arising from the solutions

cuf2paradigm(TestSort) :-
        abolish(solution/1),
        cuf_extern2intern(TestSort, Intern, Constraints, Result),
        cuf_prove(all, Intern, [], Constraints, ConstraintsOut, _),
        cuf_intern2extern(Result, ConstraintsOut, [], CUFSolutionRepresentation),
        cuf2paradigm_element(CUFSolutionRepresentation, ParadigmElement),
        assert(solution(ParadigmElement)),
        fail.
cuf2paradigm(_) :-
        collect_solutions(Input),
        all_solutions(Input).       % print out all solutions to paradigm insertion task based on Input
```

```
collect_solutions([Element|Tail]) :-
        retract(solution(Element)),
        collect_solutions(Tail).
collect_solutions([]).

%%% convert a CUF feature structure describing a word form candidate into a PM-insertable
%%% representation.
%%% cuf2paradigm_element(+CUFResult, -ParadigmElement)
%%% N.B. There's a grammar dependency here that needs to be removed in the future. Should be
%%% pretty easy, though.

cuf2paradigm_element(CUFResult, Dimensions:PhonemeSequence) :-
        searchCUFexpr(CUFResult, phon:PhonStringAsAtomList, found),
        atomlist2atom(PhonStringAsAtomList, [], PhonemeSequence),
        searchCUFexpr(CUFResult, c:CVal, CValFound),
        searchCUFexpr(CUFResult, plu:PluVal, PluValFound),
        searchCUFexpr(CUFResult, one:OneVal, OneValFound),
        searchCUFexpr(CUFResult, two:TwoVal, TwoValFound),
        searchCUFexpr(CUFResult, pret:PretVal, PretValFound),
        searchCUFexpr(CUFResult, subj:SubjVal, SubjValFound),
        searchCUFexpr(CUFResult, part:PartVal, PartValFound),
        searchCUFexpr(CUFResult, agr:FinVal, FinValFound),
        construct_dimensions([c:CVal*CValFound, plu:PluVal*PluValFound, one:OneVal*OneValFound,
two:TwoVal*TwoValFound,
                        pret:PretVal*PretValFound, subj:SubjVal*SubjValFound, part:PartVal*PartValFound,
agr:FinVal*FinValFound],
                        Dimensions),
        !.                                      % no backtracking from outside !

%% collapse a list of PROLOG atoms into a single atom by way of order-preserving concatenation

atomlist2atom([], FinalString, Atom) :-
        name(Atom, FinalString).
atomlist2atom([Atom0|Atoms], IncomingString, Atom) :-
        name(Atom0,Atom0String),
        append(IncomingString, Atom0String, IntermediateString),
        atomlist2atom(Atoms, IntermediateString, Atom).

construct_dimensions([], []).
construct_dimensions([Feature:Value*Status| More], Dimensions) :-
        ( Status == found ->
           functor(Dimension,Feature,1),
           arg(1, Dimension, Value),
           Dimensions = [Dimension|Dimensions0]
        ;   Dimensions = Dimensions0
        ), construct_dimensions(More, Dimensions0).

%%% search an expression in CUF surface syntax for a feature to be found,
%%% reporting its (non)success.
%%% searchCUFexpr(+Expression, ?ElementToBeFound, -SuccessStatus)

searchCUFexpr(Left & Right, ElementToBeFound, SuccessStatus) :-
        searchCUFexpr(Left, ElementToBeFound, SuccessStatusOfLeftConjunct),
        ( SuccessStatusOfLeftConjunct == not_found ->
           searchCUFexpr(Right, ElementToBeFound, SuccessStatus)
        ; SuccessStatus = SuccessStatusOfLeftConjunct
        ),
        !.
searchCUFexpr(Feature:Value, Feature:Value, found) :- !.
searchCUFexpr(_Feature:Value, ElementToBeFound, SuccessStatus) :-
        !, searchCUFexpr(Value, ElementToBeFound, SuccessStatus).
```

```
searchCUFexpr(_VarOrAtomicOrList, _ElementToBeFound, not_found).


%% apply a 2-place predicate P(+CellContent,+CellAddress) to every cell
%% of a paradigm. To be useful, P should have side-effects, such as printing
%% something.
%% apply_function_to_paradigm_cell(+FunctionFunctor, ?Paradigm,
%% +InheritedBooleanValuedDimensions)

apply_function_to_paradigm_cell(Function, Paradigm,
                        InheritedBooleanValuedDimensions) :-
        functor(Paradigm,paradigm,SizePlusOne),
        Size is SizePlusOne - 2, % account for extra Dim. slot in paradigm &
                         % counting down to 0
        arg(1, Paradigm, Dimensions),
        length(Dimensions, NoOfDimensions),
        'apply function to paradigm cell'(Size, NoOfDimensions, Paradigm,
                InheritedBooleanValuedDimensions, Function),
        !.

'apply function to paradigm cell'(-1, _NoOfDimensions, _Paradigm,
            _InheritedBooleanValuedDimensions, _Function) :- !.

'apply function to paradigm cell'(CellAddress, NoOfDimensions, Paradigm,
            InheritedBooleanValuedDimensions, Function) :-
        arg(1, Paradigm, Dimensions),
        paradigm_address_to_binary_values(NoOfDimensions, Dimensions,
                                CellAddress, BoolValuedAddress),
        append(InheritedBooleanValuedDimensions, BoolValuedAddress,
             RealBooleanValuedAddress),
        RealCellAddress is CellAddress + 2,
        arg(RealCellAddress, Paradigm, Cell),
        !,
        ( nonvar(Cell), functor(Cell,paradigm,_) ->
              apply_function_to_paradigm_cell(Function, Cell,
                                    RealBooleanValuedAddress)
        ; Goal =.. [Function, Cell, RealBooleanValuedAddress],
          call(Goal)
        ),
        NextCellAddress is CellAddress - 1,
        'apply function to paradigm cell'(NextCellAddress, NoOfDimensions,
            Paradigm, InheritedBooleanValuedDimensions, Function).


display_cell('*'(_FCRBlockedCell), _Address) :- !.
display_cell(Content, Address) :-
        write(Content:Address), nl.

%% show an entire paradigm in compact form

show :-
        solution(Paradigm,_GenerationHistory),
        write('
Current paradigm:
=================
'),
        apply_function_to_paradigm_cell(display_cell, Paradigm, []).

end_of_file.
```

APPENDIX 5: Sample session using CG and PM

```
% loading file /home/walther/bin/cuf
%  foreign file /usr/local/quintus/generic/qplib3.1.4/library/sun4-5/libpl.so loaded
%  foreign file /home/walther/cuf2.30/kern/foreign/counter.o loaded
%  foreign file /home/walther/cuf2.30/kern/foreign/maketemp.o loaded
%  foreign file /home/walther/cuf2.30/kern/quintus/obj/atomsat-pl.o loaded


                    CUF - Version 2.30

        Institut fuer maschinelle Sprachverarbeitung (IMS),
              Universitaet Stuttgart, Germany

           Michael Dorna, Jochen Doerre & Joerg Junger

                        July 1994


%%% global initialization: .....
% cuf loaded in module user, 1.350 sec 349,828 bytes
Quintus Prolog Release 3.1.4 (Sun-4, SunOS 5.3)
Copyright (C) 1993, Quintus Corporation.  All rights reserved.
2100 Geng Road, Palo Alto, California U.S.A. (415) 813-3800

| ?- dload(mm).



%%% CUF WARNING: Do not interrupt during file handling!
%%% reading file /home/walther/b3/konsim/cuf/mm.cuf:
.....................................................................................................
%%% checking compilation flags: .
%%% building rest of tables: .....................................................................
%%% compiling type axioms: .........................
%%% compiling declarations: .....................
%%% classifying features: ..
%%% polyfeature found: agr
.......
%%% polyfeature found: one
.........
%%% polyfeature found: two
..
%%% testing polyfeature declarations:  ......................
%%% testing monofeature declarations:  ..................
%%% checking denotations of types and constants:

%%% CUF WARNING: these types cause indeterminate atoms (at least):
%%% root_phon_string

%%% compiling clauses: ..........................................................
%%% compiling control file(s):
% compiling file /home/walther/b3/konsim/cuf/mm.ctrl
%  compiling file /home/walther/b3/konsim/cuf/pm.pl



Paradigm Mechanism (PM) Services
--------------------------------
```

```
?- cuf2paradigm(CUFExpression /* e.g. test(werfen) */)
?- pmflags. /* shows user-accesible PM flags, i.e. parameters */
?- pmflag(FlagName,FlagValue). /* Set flag to value */
?- info. /* this info */

%   Use ?- pmflag(Name, NewValue). to alter pmflags!

%   pmflag name  : pmflag value
%   ----------------------
%   stop_on_error : no
%   block_on_fcr_hit : yes
%   show_order : yes
%   number_of_retries : 50
%   violatable_insertion_behaviour : yes
%   scale_ordering : yes
%   compute_scale_position : no
%   verbose : no
%  pm.pl compiled in module user, 2.317 sec 19,688 bytes
% mm.ctrl compiled in module user, 2.434 sec 23,140 bytes

%%% deleting old interpreter code: ...
%%% building new interpreter code: .....................................................................
%%% expansion of deterministic goals: .....................................................................
%%% compilation finished:
%%% errors:   0
%%% warnings: 1

yes
| ?- cuf2paradigm(test(werfen)),show.

%% ordered paradigm cell candidates:
[agr(+),plu(+),pret(+),subj(+),two(+)]:wuerfet >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:wuerfent >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:wuerfenst >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:warfet >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:warfent >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:warfenst >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:werfteet >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:werfteent >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:werfteenst >>
[agr(+),c(+),one(-),two(+)]:wirf >>
[agr(+),c(+),one(-),two(+)]:wirfe >>
[agr(+),c(+),one(-),two(+)]:wirfste >>
[agr(+),c(+),plu(+),two(+)]:werft >>
[agr(+),c(+),plu(+),two(+)]:werfne >>
[agr(+),c(+),plu(+),two(+)]:werfnt >>
[agr(+),c(+),plu(+),two(+)]:werfte >>
[agr(+),c(+),plu(+),two(+)]:werftt >>
[agr(+),c(+),plu(+),two(+)]:werfnte >>
[agr(+),c(+),plu(+),two(+)]:werfntt >>
[agr(+),c(+),plu(+),two(+)]:werfstt >>
[agr(+),c(+),plu(+),two(+)]:werfnste >>
[agr(+),c(+),plu(+),two(+)]:werfnstt >>
[agr(+),plu(+),pret(+),subj(+)]:wuerfen >>
[agr(+),plu(+),pret(+),subj(+)]:warfen >>
[agr(+),plu(+),pret(+),two(+)]:warft >>
[agr(+),plu(+),pret(+),two(+)]:warfnt >>
[agr(+),plu(+),pret(+),two(+)]:warfnst >>
[agr(+),plu(+),pret(+),subj(+)]:werfteen >>
[agr(+),plu(+),pret(+),two(+)]:werftet >>
[agr(+),plu(+),pret(+),two(+)]:werftent >>
[agr(+),plu(+),pret(+),two(+)]:werftenst >>
```

[agr(+),plu(+),subj(+),two(+)]:werfet >>
[agr(+),plu(+),subj(+),two(+)]:werfent >>
[agr(+),plu(+),subj(+),two(+)]:werfenst >>
[agr(+),pret(+),subj(+),two(+)]:wuerfest >>
[agr(+),pret(+),subj(+),two(+)]:warfest >>
[agr(+),pret(+),subj(+),two(+)]:werfteest >>
[agr(+),c(+),two(+)]:werfe >>
[agr(+),c(+),two(+)]:werfte >>
[agr(+),c(+),two(+)]:werfste >>
[agr(+),c(+),two(+)]:werftste >>
[agr(+),one(-),two(+)]:wirfst >>
[agr(+),one(-),two(-)]:wirft >>
[agr(+),plu(+),pret(+)]:warfn >>
[agr(+),plu(+),pret(+)]:werften >>
[agr(+),plu(+),subj(+)]:werfen >>
[agr(+),plu(+),two(+)]:werft >>
[agr(+),plu(+),two(+)]:werfnt >>
[agr(+),plu(+),two(+)]:werfnst >>
[agr(+),pret(+),subj(+)]:wuerfe >>
[agr(+),pret(+),subj(+)]:warfe >>
[agr(+),pret(+),two(+)]:warfst >>
[agr(+),pret(+),subj(+)]:werftee >>
[agr(+),pret(+),two(+)]:werftest >>
[agr(+),subj(+),two(+)]:werfest >>
[agr(-),part(+)]:worfn >>
[agr(+),one(-)]:wirf >>
[agr(+),one(+)]:werfe >>
[agr(+),one(+)]:werfte >>
[agr(+),plu(+)]:werfn >>
[agr(+),pret(+)]:warf >>
[agr(+),pret(+)]:werfte >>
[agr(+),subj(+)]:werfe >>
[agr(+),two(+)]:werfst >>
[agr(+),two(+)]:werftst >>
[agr(+),two(-)]:werft >>
[agr(+),two(-)]:werftt >>
[agr(-),part(+)]:werft >>
[agr(+)]:werft >>
[agr(-)]:werfn >>
[part(-)]:werfn >>
[]:werf
paradigm([agr,plu,pret,subj,two],paradigm([part],werfn,worfn),paradigm([one],wirft,werfe),*(plu(+)>>agr(+)),
werfn,*(pret(+)>>agr(+)),warf,*(pret(+)>>agr(+)),warfn,*(subj(+)>>agr(+)),werfe,*(subj(+)>>agr(+)),werfen,
(pret(+)>>agr(+)),wuerfe,(pret(+)>>agr(+)),wuerfen,*(two(+)>>agr(+)),paradigm([c,one],wirfst,wirf,*(two(
+)>>one(-)),*(two(+)>>one(-
))),*(plu(+)>>agr(+)),paradigm([c],werft,werft),*(pret(+)>>agr(+)),warfst,*(pret(+)>>agr(+)),warft,*(subj(+)>
>agr(+)),werfest,*(subj(+)>>agr(+)),werfet,*(pret(+)>>agr(+)),wuerfest,*(pret(+)>>agr(+)),wuerfet)*
((((((

guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
here (wirfe,was blocked by (wirf, at ([c(+),one(-)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
here (wirfste,was blocked by (wirf, at ([c(+),one(-)])))),
guessing_values_and_continuing([pret(-),subj(-)],
creating_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
new_paradigm_created(
free_cell_now_occupied_with(werft,located_at,[c(+)]))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfne,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfnt,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfte,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werftt,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfnte,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfntt,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfstt,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfnste,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (werfnstt,was blocked by (werft, at ([c(+)])))),
guessing_values_and_continuing([two(-)],
free_cell_now_occupied_with(wuerfen,located_at,[agr(+),plu(+),pret(+),subj(+),two(-)])),
guessing_values_and_continuing([two(-)],
here (warfen,was blocked by (wuerfen, at ([agr(+),plu(+),pret(+),subj(+),two(-)])))),
guessing_values_and_continuing([subj(-)],
free_cell_now_occupied_with(warft,located_at,[agr(+),plu(+),pret(+),subj(-),two(+)])),
guessing_values_and_continuing([subj(-)],
here (warfnt,was blocked by (warft, at ([agr(+),plu(+),pret(+),subj(-),two(+)])))),
guessing_values_and_continuing([subj(-)],
here (warfnst,was blocked by (warft, at ([agr(+),plu(+),pret(+),subj(-),two(+)])))),
guessing_values_and_continuing([two(-)],
here (werfteen,was blocked by (wuerfen, at ([agr(+),plu(+),pret(+),subj(+),two(-)])))),
guessing_values_and_continuing([subj(-)],
here (werftet,was blocked by (warft, at ([agr(+),plu(+),pret(+),subj(-),two(+)])))),
guessing_values_and_continuing([subj(-)],
here (werftent,was blocked by (warft, at ([agr(+),plu(+),pret(+),subj(-),two(+)])))),
guessing_values_and_continuing([subj(-)],
here (werftenst,was blocked by (warft, at ([agr(+),plu(+),pret(+),subj(-),two(+)])))),
guessing_values_and_continuing([pret(-)],
free_cell_now_occupied_with(werfet,located_at,[agr(+),plu(+),pret(-),subj(+),two(+)])),
guessing_values_and_continuing([pret(-)],
here (werfent,was blocked by (werfet, at ([agr(+),plu(+),pret(-),subj(+),two(+)])))),
guessing_values_and_continuing([pret(-)],
here (werfenst,was blocked by (werfet, at ([agr(+),plu(+),pret(-),subj(+),two(+)])))),
guessing_values_and_continuing([plu(-)],

free_cell_now_occupied_with(wuerfest,located_at,[agr(+),plu(-),pret(+),subj(+),two(+)])),
guessing_values_and_continuing([plu(-)],
    here (warfest,was blocked by (wuerfest, at ([agr(+),plu(-),pret(+),subj(+),two(+)])))),
guessing_values_and_continuing([plu(-)],
    here (werfteest,was blocked by (wuerfest, at ([agr(+),plu(-),pret(+),subj(+),two(+)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([one(-)],
    here (werfe,was blocked by (wirf, at ([c(+),one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([one(-)],
    here (werfte,was blocked by (wirf, at ([c(+),one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([one(-)],
    here (werfste,was blocked by (wirf, at ([c(+),one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([one(-)],
    here (werftste,was blocked by (wirf, at ([c(+),one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
free_cell_now_occupied_with(wirfst,located_at,[c(-),one(-)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
creating_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
new_paradigm_created(
free_cell_now_occupied_with(wirft,located_at,[one(-)])))),
guessing_values_and_continuing([subj(-),two(-)],
free_cell_now_occupied_with(warfn,located_at,[agr(+),plu(+),pret(+),subj(-),two(-)]))),
guessing_values_and_continuing([subj(-),two(-)],
    here (werften,was blocked by (warfn, at ([agr(+),plu(+),pret(+),subj(-),two(-)])))),
guessing_values_and_continuing([pret(-),two(-)],
free_cell_now_occupied_with(werfen,located_at,[agr(+),plu(+),pret(-),subj(+),two(-)]))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
free_cell_now_occupied_with(werft,located_at,[c(-)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
    here (werfnt,was blocked by (werft, at ([c(-)])))))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
    here (werfnst,was blocked by (werft, at ([c(-)])))))),
guessing_values_and_continuing([plu(-),two(-)],
free_cell_now_occupied_with(wuerfe,located_at,[agr(+),plu(-),pret(+),subj(+),two(-)]))),
guessing_values_and_continuing([plu(-),two(-)],
    here (warfe,was blocked by (wuerfe, at ([agr(+),plu(-),pret(+),subj(+),two(-)])))),
guessing_values_and_continuing([plu(-),subj(-)],
free_cell_now_occupied_with(warfst,located_at,[agr(+),plu(-),pret(+),subj(-),two(+)]))),
guessing_values_and_continuing([plu(-),two(-)],
    here (werftee,was blocked by (wuerfe, at ([agr(+),plu(-),pret(+),subj(+),two(-)])))),
guessing_values_and_continuing([plu(-),subj(-)],
    here (werftest,was blocked by (warfst, at ([agr(+),plu(-),pret(+),subj(-),two(+)])))),
guessing_values_and_continuing([plu(-),pret(-)],
free_cell_now_occupied_with(werfest,located_at,[agr(+),plu(-),pret(-),subj(+),two(+)]))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
creating_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
new_paradigm_created(

free_cell_now_occupied_with(worfn,located_at,[part(+)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
here (wirf,was blocked by (wirft, at ([one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
free_cell_now_occupied_with(werfe,located_at,[one(+)]))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
here (werfte,was blocked by (werfe, at ([one(+)])))))),
guessing_values_and_continuing([pret(-),subj(-),two(-)],
free_cell_now_occupied_with(werfn,located_at,[agr(+),plu(+),pret(-),subj(-),two(-)]))),
guessing_values_and_continuing([plu(-),subj(-),two(-)],
free_cell_now_occupied_with(warf,located_at,[agr(+),plu(-),pret(+),subj(-),two(-)]))),
guessing_values_and_continuing([plu(-),subj(-),two(-)],
here (werfte,was blocked by (warf, at ([agr(+),plu(-),pret(+),subj(-),two(-)])))),
guessing_values_and_continuing([plu(-),pret(-),two(-)],
free_cell_now_occupied_with(werfe,located_at,[agr(+),plu(-),pret(-),subj(+),two(-)]))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-),one(-)],
here (werfst,was blocked by (wirfst, at ([c(-),one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-),one(-)],
here (werftst,was blocked by (wirfst, at ([c(-),one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([one(-)],
here (werft,was blocked by (wirft, at ([one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([one(-)],
here (werftt,was blocked by (wirft, at ([one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
here (werft,was blocked by (worfn, at ([part(+)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([one(-)],
here (werft,was blocked by (wirft, at ([one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([part(-)],
free_cell_now_occupied_with(werfn,located_at,[part(-)])))),
guessing_values_and_continuing([agr(-),plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
set_compatible_cell_at([part(-)]))),
guessing_values_and_continuing([agr(-),plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([part(-)],
here (werf,was blocked by (werfn, at ([part(-)])))))))
Current paradigm:
==========
wuerfet:[agr(+),plu(+),pret(+),subj(+),two(+)]
wuerfest:[agr(+),plu(-),pret(+),subj(+),two(+)]
werfet:[agr(+),plu(+),pret(-),subj(+),two(+)]
werfest:[agr(+),plu(-),pret(-),subj(+),two(+)]
warft:[agr(+),plu(+),pret(+),subj(-),two(+)]
warfst:[agr(+),plu(-),pret(+),subj(-),two(+)]
werft:[agr(+),plu(+),pret(-),subj(-),two(+),c(+)]
werft:[agr(+),plu(+),pret(-),subj(-),two(+),c(-)]

wirf:[agr(+),plu(-),pret(-),subj(-),two(+),c(+),one(-)]
wirfst:[agr(+),plu(-),pret(-),subj(-),two(+),c(-),one(-)]
wuerfen:[agr(+),plu(+),pret(+),subj(+),two(-)]
wuerfe:[agr(+),plu(-),pret(+),subj(+),two(-)]
werfen:[agr(+),plu(+),pret(-),subj(+),two(-)]
werfe:[agr(+),plu(-),pret(-),subj(+),two(-)]
warfn:[agr(+),plu(+),pret(+),subj(-),two(-)]
warf:[agr(+),plu(-),pret(+),subj(-),two(-)]
werfn:[agr(+),plu(+),pret(-),subj(-),two(-)]
werfe:[agr(+),plu(-),pret(-),subj(-),two(-),one(+)]
wirft:[agr(+),plu(-),pret(-),subj(-),two(-),one(-)]
worfn:[agr(-),plu(-),pret(-),subj(-),two(-),part(+)]
werfn:[agr(-),plu(-),pret(-),subj(-),two(-),part(-)]

yes
| ?- cuf2paradigm(test(lachen)),show.

%% ordered paradigm cell candidates:
[agr(+),plu(+),pret(+),subj(+),two(+)]:lachteet >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:lachteent >>
[agr(+),plu(+),pret(+),subj(+),two(+)]:lachteenst >>
[agr(+),c(+),plu(+),two(+)]:lacht >>
[agr(+),c(+),plu(+),two(+)]:lachne >>
[agr(+),c(+),plu(+),two(+)]:lachnt >>
[agr(+),c(+),plu(+),two(+)]:lachte >>
[agr(+),c(+),plu(+),two(+)]:lachtt >>
[agr(+),c(+),plu(+),two(+)]:lachnte >>
[agr(+),c(+),plu(+),two(+)]:lachntt >>
[agr(+),c(+),plu(+),two(+)]:lachstt >>
[agr(+),c(+),plu(+),two(+)]:lachnste >>
[agr(+),c(+),plu(+),two(+)]:lachnstt >>
[agr(+),plu(+),pret(+),subj(+)]:lachteen >>
[agr(+),plu(+),pret(+),two(+)]:lachtet >>
[agr(+),plu(+),pret(+),two(+)]:lachtent >>
[agr(+),plu(+),pret(+),two(+)]:lachtenst >>
[agr(+),plu(+),subj(+),two(+)]:lachet >>
[agr(+),plu(+),subj(+),two(+)]:lachent >>
[agr(+),plu(+),subj(+),two(+)]:lachenst >>
[agr(+),pret(+),subj(+),two(+)]:lachteest >>
[agr(+),c(+),two(+)]:lache >>
[agr(+),c(+),two(+)]:lachte >>
[agr(+),c(+),two(+)]:lachste >>
[agr(+),c(+),two(+)]:lachtste >>
[agr(+),plu(+),pret(+)]:lachten >>
[agr(+),plu(+),subj(+)]:lachen >>
[agr(+),plu(+),two(+)]:lacht >>
[agr(+),plu(+),two(+)]:lachnt >>
[agr(+),plu(+),two(+)]:lachnst >>
[agr(+),pret(+),subj(+)]:lachtee >>
[agr(+),pret(+),two(+)]:lachtest >>
[agr(+),subj(+),two(+)]:lachest >>
[agr(+),one(+)]:lache >>
[agr(+),one(+)]:lachte >>
[agr(+),plu(+)]:lachn >>
[agr(+),pret(+)]:lachte >>
[agr(+),subj(+)]:lache >>
[agr(+),two(+)]:lachst >>
[agr(+),two(+)]:lachtst >>
[agr(+),two(-)]:lacht >>
[agr(+),two(-)]:lachtt >>
[agr(-),part(+)]:lacht >>

[agr(+)]:lacht >>
[agr(-)]:lachn >>
[part(-)]:lachn >>
[]:lach
paradigm([agr,plu,pret,subj,two],paradigm([part],lachn,lacht),paradigm([one],lacht,lache),*(plu(+)>>agr(+)),la
chn,*(pret(+)>>agr(+)),lachte,*(pret(+)>>agr(+)),lachten,*(subj(+)>>agr(+)),lache,*(subj(+)>>agr(+)),lachen,
(pret(+)>>agr(+)),lachtee,(pret(+)>>agr(+)),lachteen,*(two(+)>>agr(+)),paradigm([c],lachst,lache),*(plu(+)
>>agr(+)),paradigm([c],lacht,lacht),*(pret(+)>>agr(+)),lachtest,*(pret(+)>>agr(+)),lachtet,*(subj(+)>>agr(+)),
lachest,*(subj(+)>>agr(+)),lachet,*(pret(+)>>agr(+)),lachteest,*(pret(+)>>agr(+)),lachteet)*(((((((((((((((((((((
((((((((((((,
new_paradigm_created(
free_cell_now_occupied_with(lachteet,located_at,[agr(+),plu(+),pret(+),subj(+),two(+)])),
here (lachteent,was blocked by (lachteet, at ([agr(+),plu(+),pret(+),subj(+),two(+)]))),
here (lachteenst,was blocked by (lachteet, at ([agr(+),plu(+),pret(+),subj(+),two(+)]))),
guessing_values_and_continuing([pret(-),subj(-)],
creating_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
new_paradigm_created(
free_cell_now_occupied_with(lacht,located_at,[c(+)]))))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachne,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachnt,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachte,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachtt,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachnte,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachntt,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachstt,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachnste,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
here (lachnstt,was blocked by (lacht, at ([c(+)])))),
guessing_values_and_continuing([two(-)],
free_cell_now_occupied_with(lachteen,located_at,[agr(+),plu(+),pret(+),subj(+),two(-)])),
guessing_values_and_continuing([subj(-)],
free_cell_now_occupied_with(lachtet,located_at,[agr(+),plu(+),pret(+),subj(-),two(+)])),
guessing_values_and_continuing([subj(-)],
here (lachtent,was blocked by (lachtet, at ([agr(+),plu(+),pret(+),subj(-),two(+)]))),
guessing_values_and_continuing([subj(-)],
here (lachtenst,was blocked by (lachtet, at ([agr(+),plu(+),pret(+),subj(-),two(+)]))),
guessing_values_and_continuing([pret(-)],
free_cell_now_occupied_with(lachet,located_at,[agr(+),plu(+),pret(-),subj(+),two(+)])),
guessing_values_and_continuing([pret(-)],
here (lachent,was blocked by (lachet, at ([agr(+),plu(+),pret(-),subj(+),two(+)]))),
guessing_values_and_continuing([pret(-)],
here (lachenst,was blocked by (lachet, at ([agr(+),plu(+),pret(-),subj(+),two(+)]))),
guessing_values_and_continuing([plu(-)],
free_cell_now_occupied_with(lachteest,located_at,[agr(+),plu(-),pret(+),subj(+),two(+)])),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],

creating_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
new_paradigm_created(
free_cell_now_occupied_with(lache,located_at,[c(+)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
here (lachte,was blocked by (lache, at ([c(+)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
here (lachste,was blocked by (lache, at ([c(+)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
here (lachtste,was blocked by (lache, at ([c(+)])))))),
guessing_values_and_continuing([subj(-),two(-)],
free_cell_now_occupied_with(lachten,located_at,[agr(+),plu(+),pret(+),subj(-),two(-)]))),
guessing_values_and_continuing([pret(-),two(-)],
free_cell_now_occupied_with(lachen,located_at,[agr(+),plu(+),pret(-),subj(+),two(-)]))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
free_cell_now_occupied_with(lacht,located_at,[c(-)])))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
here (lachnt,was blocked by (lacht, at ([c(-)])))))),
guessing_values_and_continuing([pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(+),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
here (lachnst,was blocked by (lacht, at ([c(-)])))))),
guessing_values_and_continuing([plu(-),two(-)],
free_cell_now_occupied_with(lachtee,located_at,[agr(+),plu(-),pret(+),subj(+),two(-)]))),
guessing_values_and_continuing([plu(-),subj(-)],
free_cell_now_occupied_with(lachtest,located_at,[agr(+),plu(-),pret(+),subj(-),two(+)]))),
guessing_values_and_continuing([plu(-),pret(-)],
free_cell_now_occupied_with(lachest,located_at,[agr(+),plu(-),pret(-),subj(+),two(+)]))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
creating_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
new_paradigm_created(
free_cell_now_occupied_with(lache,located_at,[one(+)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
here (lachte,was blocked by (lache, at ([one(+)])))))),
guessing_values_and_continuing([pret(-),subj(-),two(-)],
free_cell_now_occupied_with(lachn,located_at,[agr(+),plu(+),pret(-),subj(-),two(-)]))),
guessing_values_and_continuing([plu(-),subj(-),two(-)],
free_cell_now_occupied_with(lachte,located_at,[agr(+),plu(-),pret(+),subj(-),two(-)]))),
guessing_values_and_continuing([plu(-),pret(-),two(-)],
free_cell_now_occupied_with(lache,located_at,[agr(+),plu(-),pret(-),subj(+),two(-)]))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
free_cell_now_occupied_with(lachst,located_at,[c(-)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(+)],
guessing_values_and_continuing([c(-)],
here (lachtst,was blocked by (lachst, at ([c(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([one(-)],
free_cell_now_occupied_with(lacht,located_at,[one(-)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([one(-)],

```
      here (lachtt,was blocked by (lacht, at ([one(-)])))))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
creating_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
new_paradigm_created(
free_cell_now_occupied_with(lacht,located_at,[part(+)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(+),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([one(-)],
set_compatible_cell_at([one(-)])))),
guessing_values_and_continuing([plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([part(-)],
free_cell_now_occupied_with(lachn,located_at,[part(-)])))),
guessing_values_and_continuing([agr(-),plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
set_compatible_cell_at([part(-)]))),
guessing_values_and_continuing([agr(-),plu(-),pret(-),subj(-),two(-)],
inserting_in_subparadigm_at([agr(-),plu(-),pret(-),subj(-),two(-)],
guessing_values_and_continuing([part(-)],
      here (lach,was blocked by (lachn, at ([part(-)])))))))


Current paradigm:
=================
lachteet:[agr(+),plu(+),pret(+),subj(+),two(+)]
lachteest:[agr(+),plu(-),pret(+),subj(+),two(+)]
lachet:[agr(+),plu(+),pret(-),subj(+),two(+)]
lachest:[agr(+),plu(-),pret(-),subj(+),two(+)]
lachtet:[agr(+),plu(+),pret(+),subj(-),two(+)]
lachtest:[agr(+),plu(-),pret(+),subj(-),two(+)]
lacht:[agr(+),plu(+),pret(-),subj(-),two(+),c(+)]
lacht:[agr(+),plu(+),pret(-),subj(-),two(+),c(-)]
lache:[agr(+),plu(-),pret(-),subj(-),two(+),c(+)]
lachst:[agr(+),plu(-),pret(-),subj(-),two(+),c(-)]
lachteen:[agr(+),plu(+),pret(+),subj(+),two(-)]
lachtee:[agr(+),plu(-),pret(+),subj(+),two(-)]
lachen:[agr(+),plu(+),pret(-),subj(+),two(-)]
lache:[agr(+),plu(-),pret(-),subj(+),two(-)]
lachten:[agr(+),plu(+),pret(+),subj(-),two(-)]
lachte:[agr(+),plu(-),pret(+),subj(-),two(-)]
lachn:[agr(+),plu(+),pret(-),subj(-),two(-)]
lache:[agr(+),plu(-),pret(-),subj(-),two(-),one(+)]
lacht:[agr(+),plu(-),pret(-),subj(-),two(-),one(-)]
lacht:[agr(-),plu(-),pret(-),subj(-),two(-),part(+)]
lachn:[agr(-),plu(-),pret(-),subj(-),two(-),part(-)]

yes
| ?-
```

Liste der bisher erschienenen Arbeitspapiere des SFB 282

*1 Joachim Jacobs: Bewegung als Valenztransfer (Juni 91)
*2 LEXLERN: Studies on the Acquisition of Agreement and Case Marking (Juni 91)
*3 Tracy A. Hall: On the Representation of Geminate Consonants in Korean (Juli 91)
*4 LEXLERN: Universal Grammar in Second Language Acquisition (Aug. 91)
*5 Gisa Rauh: Zur Grammatik Lexikalischer und nicht-lexikalischer Präpositionen im Englischen (Juni 91)
6 Gisbert Fanselow: Ein modulares Konzept der Lexikonerweiterung
Susan Olsen: Zur Grammatik des Wortes -Argumente zur Argumentvererbung (Sept. 91)
7 Elisabeth Löbel: Zur kategorialen Bestimmung der Possessiva in der NP/DP: Possessiva als Adjektive (Sept. 91)
8 Ingrid Kaufmann: Semantik der wegbezogenen Präpositionen des Deutschen (Aug. 91)
9 Barbara Lenz: Un-Affigierung im Lexikon (Okt. 91)
10 Daniel Büring/Katharina Hartmann: Regelhaft und regellos. Zum Verhältnis von Syntax und Lexikon (Nov. 91)
*11 Annette Killmer: Logische Räume (Jan. 92)
*12 Gisa Rauh: Präpositionale Formen im Lexikon (Jan. 92)
*13 Joachim Jacobs: Integration (Jan. 92)
14 Si-Taek Yu: Probleme der Unterspezifikation in der koreanischen Phonologie (Jan. 92)
*15 Dagmar Haumann: Lexikalische Argumentstuktur und θ-Rollen englischer Lokalpräpositionen (Feb. 92)
*16 Harald Clahsen: Overregularization in the acquisition of inflectional morphology (März 92)
*17 Angelika Huppertz: 'A Phrase for Case' - Zum syntaktischen Anspruch einer KP-Analyse (Mai 92)
*18 Wolfgang Kehrein: Lexikalische Phonologie des Finnischen (Juni 92)
*19 Barbara Lenz: Modalverben und Dualität im Deutschen (April 92)
*20 Dieter Wunderlich: Towards a lexicon-based theory of agreement (April 92)
*21 Dieter Wunderlich: A minimalist analysis of German verb morphology (Mai 92)
*22 Dagmar Haumann: Lexikalische Argumentstrukturen und Derivation: Präpositionale Derivata der Form [x-ward(s)] (Juli 92)
23 Regina Pustet: The Lakota Article (Juli 92)
*24 Albert Ortmann: Zur Auflösung von Merkmalskonflikten unter Kongruenz
Dieter Wunderlich: Funktionale Kategorien im Lexikon (Aug. 91)
*25 Harald Clahsen/Teresa Parodi/Martina Penke: Functional categories in Early Grammars: Stage I in German (Aug. 92)
*26 Gisa Rauh: Kasus und Präpositionen im Englischen (Aug. 92)
27 Hans-Jürgen Sasse: Das Nomen - eine universale Kategorie? (Aug. 92)
28 Ursula Kleinhenz: Zum Lexikon isolierender Sprachen: Baka (Kamerun) (Sept. 92)
29 James Kilbury/Petra Naerger/Ingrid Renz: New Lexical Entries for Unknown Words
James Kilbury: Paradigm-Based Derivational Morphology
Angelika Koelzer: Eine Semantische Repräsentation für die SIMLEX-Grammatik (Sept. 92)
30 Barbara Stiebels/Dieter Wunderlich: A lexical account of complex verbs (Sept. 92)
31 Renate Pasch: Sind kausale und konzessive Konstruktionen Duale voneinander? (Okt. 92)
*32 Byong-Rae Ryu: On the Nature of the Two Types of Negation in Korean (Okt. 92)
33 Elisabeth Löbel: KP/DP-Syntax: Interaction of Case-marking with Referential and Nominal Features (Dez. 92)
34 Byong-Rae Ryu: Delimiter und Dualität im Koreanischen (Mai 93)
35 Eva Schultze-Berndt: Nomen und Nominalphrase in zwei australischen Sprachen - Nunggubuyu, Kayardild
Regina Pustet: A preliminary survey of lexical class formation in Lakota (Feb. 93)
36 Dieter Wunderlich: CAUSE and the structure of verbs (Nov. 92)
37 Harald Clahsen: Linguistic Perspectives on Specific Language Impairment (Dez. 92)
38 Ray Fabri: The Subject Agreement Paradigm in Classical Arabic
Martina Urbas: Numeruskongruenz und Numeruskonflikte in Kopulasätzen (Feb. 93)
*39 Gisa Rauh: Grammatische Kategorien (Feb. 93)
40 Gary Marcus/Ursula Brinkmann/Harald Clahsen/Richard Wiese/Andreas Woest/Steven Pinker: German Inflection: The Exception that Proves the Rule (Feb. 93)
42 Tibor Kiss: Infinite Komplementation (März 93)
*43 Harald Clahsen/Upyong Hong: Grammatical constraints in syntactic processing: sentence matching experiments on German (Apr. 93)
*44 Dagmar Haumann: Referential Argument (Apr. 93)

*45 Barbara Lenz: Affix-Negation und Dualität (Mai 93)
46 Gisa Rauh: Präpositionen und Rollen (Mai 93)
47 Sandra Joppen: Kausativ im Baskischen (Juni 1993)
*48 Kerstin Blume: Valenz deutscher Verben und (Nicht-) Notwendigkeit (Aug.1993)
49 Helga Weyerts / Harald Clahsen: Netzwerke und symbolische Regeln im Spracherwerb: Experimentelle Ergebnisse zur Entwicklung der Flexionsmorphologie (September 1993)
50 Daniel Büring: Interacting Modules, Word Formation, and the Lexicon in Generative Grammar (Nov. 93)
51 Barbara Lenz: Ausgewählte Aspekte von wer, sein und bleiben
-.Deutsch un-/- los vs. Russisch ne-/bz
Ewald Lang: Duale Operatoren- Annotierte Literatur (November 1993)
53 Joachim Jacobs: The Lexical Basis of Optional Complements (Dez. 1993)
54 Julius. E. Moravcik: Genericity and Linguistic Competence, (Nov. 94)
55 Ewald Lang: Antonymie im Lexikon und im Wörterbuch (Jan. 94)
56 Richard Wiese (ed.): Recent Developments in Lexical Phonology (Febr. 1994 für **DM 20** abzugeben)
57 Sonja Eisenbeiß / Susanne Bartke / Helga Weyerts / Harald Clahsen: Elizitationsverfahren in der Spracherwerbsforschung: Nominalphrasen, Kasus, Plural, Partizipien (Aug. 94)
58 Ingrid Kaufmann: Was ist ein (un)mögliches Verb? Agens- und Themaprädikate und der Aufbau der Semantischen Form (Aug. 94)
59 Petra Barg: Automatic Acquisition of DATR Theories from Observations (Aug.94)
60 Stefan Engelberg: Ereignisstrukturen, Zur Syntax und Semantik von Verben (Nov. 94)
61 Andrezey Boguslawski: Dualiy and knowledge, The root and the fruit (Okt. 94)

Zu bestellen unter der folgenden Adresse:

Sonderforschungsbereich 282
z. Hd. Brigitte Marczinzik/Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf , Tel.(0)211/311-5280

Die mit einem * versehenen Titel sind vergriffen, bei Bedarf bitte an die Autoren wenden.