Morphology with a Null-Interface

Harald Trost and Johannes Matiasek
Austrian Research Institute for Artificial Intelligence[*]
Vienna, Austria
email: harald@ai.univie.ac.at

Abstract

We present an integrated architecture for word-level and sentence-level processing in a unification-based paradigm. The core of the system is a CLP implementation of a unification engine for feature structures supporting relational values. In this framework an HPSG-style grammar is implemented. Word-level processing uses X2MORF a morphological component based on an extended version of two-level morphology. This component is tightly integrated with the grammar as a relation. The advantage of this approach is that morphology and syntax are kept logically autonomous while at the same time minimizing interface problems.

1 Introduction

Over the last few years there has been a growing interest in computational morphology and phonology. A number of systems have been developed that deal with word-level processing. A widely used approach is finite-state morphology, most notably two-level morphology (for an introduction, see Sproat 92). Morphological components are successfully used for a wide range of stand-alone applications like spelling correction and hyphenation. One obvious application is the use in NLP systems geared at the analysis/generation of text. Surprisingly, they have not been widely applied in this domain up to now.

A main reason is the problem of interfacing morphology with syntax. Reflecting the current trend in syntax towards lexicalism unification-based systems use highly structured feature structures as input. Translating the output of morphological components into such a representation has proved to be difficult. Reducing interface problems is therefore crucial for success.

A tight integration between word and sentence level processing also has linguistic advantages. The boundary between morphology and syntax is fuzzy. When processing written

text the units morphology has to deal with are—in a technical sense—not words but character strings separated by delimiters. While these strings roughly correspond to the words of a sentence there are problematic cases. In German, e.g., *zu*-infinitive or verbs with separable prefixes are written as a single unit in some instances and separately in others.

The problem has been recognized and some possible remedies have been proposed. They all try to minimize or to eliminate the interface between word and sentence level processing. One step is the description of word formation in terms of a unification-based grammar to make the result of morphological processing directly available to syntax and vice versa, an approach already taken in X2MORF (Trost 90, Trost 91), an extension of two-level morphology.

The harder problem is the integration of morphophonology which is traditionally formalized in a way not easily translatable into the feature formalism. We will show, how this can be achieved by merging the word-level grammar of X2MORF into an HPSG-style grammar, and by adopting a relational view of its two-level rules.

In this paper we assume basic familiarity with unification-based NLP techniques and two-level morphology.

2   Integrating Morphology into HPSG

Head-driven Phrase Structure Grammar (HPSG, Pollard & Sag 87, Pollard & Sag in press) can be viewed as a mono-level but multi-stratal theory of grammar, where different strata relate to different aspects of linguistic information, but are represented uniformly in feature logics. As such it is well suited as a linguistic theory for our enterprise.

HPSG differentiates between three strata—PHON, SYNSEM and DTRS. Though morphology is not considered in the standard approach, it suggests itself to include it as a fourth stratum by introducing a feature MORPH into the type *sign*. Morphotactics are easily described in terms of a feature based grammar. The problem is, how to deal with morphophonology. Two proposals have been made to overcome this problem.

Krieger et al. 93 encode finite state automata directly in the feature formalism. Since two-level rules can be compiled into such automata, morphophonology can be straightforwadly integrated into the grammar. While this is formally elegant it seems to be no good solution for practical considerations. First, it is not entirely clear from their paper how the problem of null characters can be handled. Second, encoding large automata will result in a very large and unwieldy type hierarchy. In general, introducing automata into feature structures and encoding morphophonology directly at that level seems to be too low-level.

Bird & Klein 93 argue against the use of two-level morphology because of linguistic considerations. The linguistic background of two-level rules—main stream segmental phonology—has widely been rejected as a valid linguistic model. Instead, they base their implementation on autosegmental phonology (cf. Goldsmith 90).

This is certainly linguistically appealing. But there are reasons for sticking to a more conservative approach. Finite-state morphology as a formalism is not necessarily tied to segmental phonology. There are various approaches to cope with non-concatenative

phenomena—one of them X2MorF (Trost 90). Also, for a number of languages complete sets of two-level rules do exist and can immediately be brought to bear. Finally, finite-state morphology has proven to be efficient while the method proposed by Bird & Klein 93 seems to be computationally costly.

Like the other approaches ours is also based on HPSG. However, we do employ a different approach to integration. Our grammar is encoded using a unification engine based on constraint logic programming (CLP). Besides conventional attribute-value descriptions this system allows for the direct representation of more general relations, as they are required by HPSG. This extension of the formalism is used for the integration of morphology. Thus X2MorF is treated as one special relation of the grammar. As a result, our approach is more modular than the others. While being fully integrated morphology can still be viewed as an autonomous component leading to a more flexible design.

We will now give an overview of X2MorF before describing the integrated system and its implementation in detail.

3 Word Level Processing – X2MorF

X2MorF differs from standard two-level morphology in two important aspects. Continuation classes are replaced by a feature-based word grammar. This allows for a more fine-grained description of morphs. It is also a prerequisite for a tight integration with a unification-based grammar. X2MorF uses a morph lexicon where each morph has one or more feature structures assigned. The word grammar itself is simple. Morphs have a functor-argument structure along the lines of di Sciullo & Williams 87. Affixes are unary functors while stems are arguments without any further structure resulting in a binary tree structure.

The other extension concerns the two-level rules which are supplemented with a morphological filter consisting of a feature structure. This is important because in morphophonology only some rules are purely phonologically motivated. Others are triggered by a mixture of phonological and morphological facts. Such rules can not be properly represented in the standard approach.

Take, e.g., umlaut and schwa epenthesis in German: The third person singular present tense suffix for German verbs is -t, e.g., *sag-t* → *sagt*. For stems ending in a dental, schwa is inserted before the ending, e.g., *bad-t* → *badet*. This rule does not hold across the whole vocabulary though. Stems of the strong paradigm do exhibit umlaut in 3rdPersSgPres which blocks schwa epenthesis. Therefore, the final dental of the stem must be omitted instead, e.g., *rat-t* → *rät*.

The following three rules[1]—together with the appropriate entries in the morph lexicon (cf. Fig. 6 below)—produce the required behaviour.In particular, these rules relate surface *rät* to lexical *$rAt+t$*.

---

[1]These rules as well as other data presented in the examples are simplified for the purpose of demonstration

(i) $A{:}\ddot{a} \iff \_ ;$ [MORPH|MHEAD|UMLAUT *aou-umlaut*][2]

(ii) $t{:}0 \iff \_ +{:}0\ t$

(iii) $+{:}e \iff dental\ \_ +{:}0\ [s \mid t] ;$ [MORPH|MHEAD|EPENTHESE +]

X2MORF can be seen as a relation between a surface string (the word form), a lexical string and a feature structure (the interpretation of the word form). Relevant for sentence level processing is the morphosyntactic information and the stem found as the values of pathes MORPH|MHEAD and MORPH|STEM respectively (cf. Fig. 8 below). This is supplemented by lexeme specific information in the value of SYNSEM (for a detailed description see Trost 93).

4   Implementing HPSG in a CLP Framework

HPSG employs strongly typed feature structures and principles to further constrain them. *Well-typedness* requirements restrict the space of valid feature structures (cf. Carpenter 92): Every feature structure must be associated with a type and every type restricts its associated feature structure in that only certain features are allowed and the values of these features must be of a certain type. Appropriateness and value restrictions are inherited along the type hierarchy.

The second source of constraints to admit only linguistically valid feature structures are the principles of grammar. Pollard & Sag 87 allow general implicative and negative constraints in the form of conditional feature structures, in Pollard & Sag in press principles are given only in verbal form. Recent work on formalizing the basis of HPSG models them as constraints attached to types (e.g., Carpenter et al. 91). However, these distinctions affect only the way how the applicability of a principle is specified. More important for our present purpose is the form which the constraints expressed by a principle may take. Besides constraints enforcing simple structure sharing (e.g., the Head Feature Principle given in Fig.1) there are also complex relational dependencies (e.g., in the Subcategorization Principle[3]). Constraints like these go beyond the expressivity of pure feature formalisms alone and need to be defined in a recursive manner.

In order to integrate such complex constraints in the feature unification framework, we interpret unification of typed feature structures under the restrictions of principled constraints as constraint solving in the CLP paradigm (Jaffar & Lassez 87).

In CLP the notion of unification is replaced by the more general notion of constraint solving. Constraint solvers may be embedded into a logic programming language either by writing a metainterpreter or by making use of a system which allows for the implementation of unification extensions.

---

[2] The lexical character A may have the surface realizations a and $\ddot{a}$. The rule has an empty phonological context but a morphological filter. This is an example for the treatment of non-concatenative phenomena in X2MORF.

[3] "In a headed phrase (i.e., a phrasal sign whose DTRS value is of sort *head-struc*), the SUBCAT value of the head daughter is the concatenation of the phrase's SUBCAT list with the list (in order of increasing obliqueness) of SYNSEM values of the complement daughters."(Pollard & Sag in press)

The second approach is taken by DMCAI CLP[4] (Holzbaur 92), a Prolog system whose unification mechanism is extended in such a way that the user may introduce interpreted terms and specify their meaning with regard to unification through Prolog predicates. The basic mechanism to achieve this behavior is the use of *attributed variables*, which are allowed to be qualified by arbitrary user-defined attributes. Attributed variables behave like ordinary Prolog variables with two notable exceptions: when an attributed variable is to be unified with a non-variable term or another attributed variable the unification extensions come into play. For either case the user has to supply a predicate which explicitly specifies how the attributes interact and how they have to be interpreted with respect to the semantics of the application domain. Unification succeeds only if these constraint solving clauses managing the combination—or verification—of the involved attributes are successful.

The implementation of typed feature structures in our system makes use of the CLP facilities provided by this enhanced Prolog system. Feature structures are implemented by the attribute `fs(Type,Dag,Goals)`, where `Dag` is a list of feature-value pairs, which is empty in case of atomic types, or a marker indicating uninstatiatedness of the substructure (feature structures are instantiated lazily). `Goals` is a list of delayed constraints (see below). Well-typed unification of two feature structures is implemented via the constraint solving clauses mentioned above, taking into account type hierarchy and feature appropriateness (for a detailed description cf. Matiasek & Heinz 93).

Constraints imposed onto feature structures by the principles of grammar are stated in a conditional form where the antecedent is restricted to contain only typing requirements.[5] In order to account for these conditional constraints we adopt a licensing view: Every node of a feature structure has to be licensed by all principles of grammar.

A node is **licensed** by a principle if either *(i)* the feature structure F rooted in that node *satisfies* the applicability conditions of the principle and the constraints expressed by the principle successfully unify with F, or *(ii)* the feature structure F rooted in that node is *incompatible* with the applicability conditions of the principle. The interesting case arises when a feature structure does *not satisfy* the applicability conditions of the principle but is *compatible* with them. Thus applicability of the principle can be decided only later, after further instantiation or unification steps have restricted the (sub)structure rooted at that node. In precisely this case the application (or the abandoning) of the constraint has to be delayed. The delay mechanism utilizes the `Goals` slot in the `fs/3`[6] attribute, which is dedicated to hold the delayed constraints. As an example take the well known Head Feature Principle of HPSG (Fig.1[7]). The conditional operator `===>` is translated at read time via `term_expansion/2` and implements the delay mechanism by compiling precondition checks

---

[4]DMCAI CLP is an enhanced version of SICStus Prolog, available by anonymous ftp from `ftp.ai.univie.ac.at`

[5]This is only a syntactic variant of attaching constraints solely to types (Carpenter et al. 91) and does not permit general conditional structures as used in Pollard & Sag 87.

[6]`pred/n` is the usual notation for a n-ary Prolog predicate.

[7]The operators `::=`, `::`, `:`, `===` are defined for typing of a node, path restriction, path concatenation and value restriction (type *or* coreference) respectively.

AVM	*headed-phrase* $\begin{bmatrix} \text{SYNSEM\|LOC\|CAT\|HEAD} & \boxed{1} \\ \text{DTRS\|HEAD-DTR\|SYNSEM\|LOC\|CAT\|HEAD} & \boxed{1} \end{bmatrix}$

Prolog	`head_feature_principle(X) :-`
	`        X::=headed_phrase`
	`    ===>`
	`        X::synsem:loc:cat:head===H,`
	`        X::dtrs:head_dtr:synsem:loc:cat:head===H.`

Figure 1: Head Feature Principle

into the principle. These antecedent checks trigger either the application of the principle, or its dropping or its delay (by annotating the variables not sufficiently constrained to decide on the antecedent with the delayed goals).

Two advantages of this approach to implement principled constraints are especially important for our present purpose: First, stating redundant typing requirements for embedded structures (i.e. type restrictions that would follow automatically from well-typing) functions as forcing delay of the conditional constraint until these substructures are instantiated. This device can, e.g., be used to block infinite recursion in recursively defined constraints. Second, the right hand part of the conditional is not restricted to feature logical expressions, but instead can contain arbitrary Prolog goals. This way constraints involving relational dependencies (such as the Subcategorization Principle and the morphological relation between a lexical and a surface string) can be expressed within the feature formalism. Thus there is no need for external devices controlling this interaction. Furthermore, the conditional constraint syntax is not restricted to unary licensing principles, but can also be used to express relations, such as **fs_append/3**—needed for implementing the Subcat Principle—which appends two feature structure lists (Fig. 2). Note that dis-

```
fs_append(X,Y,Z)         :- fs_empty_append(X,Y,Z),
                            fs_nonempty_append(X,Y,Z).
fs_empty_append(X,Y,Z)    :- X::=elist  ===> Y = Z.
fs_nonempty_append(X,Y,Z) :- X::=nelist ===> X::first===F,Z::first===F,
                                            X::rest===XRest,Z::rest===ZRest,
                                            fs_append(XRest,Y,ZRest).
```

Figure 2: Append for feature structure lists

junctive relations such as *append* can now be written as the conjunction of two specialized cases applying conditionally. Furthermore, infinite loops due to uninstantiated variables can never occur, a crucial requirement when integrating relational dependencies into a lazy instantiating feature formalism.

5 Embedding X2MorF into the Feature System

Originally, X2MorF was realized as a separate morphological component interfaced to the sentence analyzer/generator only via sequential data transfer. In case of analysis the feature structure representing the word form was transmitted to the parser. For generation, X2MorF expected a feature structure as input reproducing one or more word forms. This purely sequential architecture was not satisfying because of the problems mentioned in the introduction.

In order to achieve tight integration, we adopt a relational view of X2MorF, and encode the relation between surface string and lexical string directly without using finite state automata (for arguments supporting this approach cf. Abramson 92). However, our approach extends the one of Abramson 92 in that it *(i)* explicitly accounts for insertion of null characters and *(ii)* introduces the filter concept of X2MorF into the relational approach.

The general format of a two-level rule specification in our system is

`LeftContextPairs <=> TransitionPair <=> RightContextPairs` $\big[$`:- Filter`$\big]$

in the case of equivalence rules, optional rules are written using only single arrows (`=>` and `<=`). These rules are compiled into Prolog clauses[8] relating the lexical and surface character streams appropriately (see Fig.3 for an example of the *t-elision* rule for German).

	$t{:}0 \Longleftrightarrow \_ +{:}0\ t$
Input	`_ <=> t:0 <=> ['+':0, t:t].`

Compiled	`morphrule([116,43,116|LS],[Sc,48,116|SS0],SS,LCon,SCon,F) :-`
	`    !, Sc=48,`
	`    morphology([43,116|LS],[48,116|SS0],SS,[116|LCon],[H|SCon],F).`

Figure 3: Sample Two-Level Rule

To obtain a correct relationship between surface and lexical string, every transition has to be licensed by a morphological rule. Transitions not mentioned by rules are handled by a default rule. Instantiation of contexts may not be done by the rules itself, since this would make it impossible to obtain negation via the cut-operator but, instead, is handled separately in a backtrackable fashion.

The central relation is the `morphology` predicate, (see Fig. 4) mediating between lexical string, surface string (with inserted null elements), the pure (denullified) surface string and the feature structure of the morphological sign. Instantiation of pairs is done depending

---

[8]Note, that left contexts are encoded reversed to account for the left to right traversal of the pair of character streams. Left contexts can be rememebered and checked most efficiently this way.

```
morphology(LexStream,SurfStream0,SurfPlainIn,LexContext,SurfContext,F) :-
   instantiate(LexStream,SurfStream0,SurfPlainIn,SurfPlainOut,F),
   morphrule(LexStream,SurfStream,SurfPlainOut,LexContext,SurfContext,F).
```

```
instantiate([LC|LCs],[SC|SCs],SurfPlainIn,SurfPlainOut,F) :-
  valid_alphabet_pair(LC,SC],
  synchronize([SC|SCs],SurfPlainIn,SurfPlainOut),
  lookahead(LC,LCs,SCs,SurfPlainOut).
```

```
synchronize([48|_],Stream,Stream) :- !.
synchronize([Char|_],[Char|Stream],Stream).
```

Figure 4: The `morphology` relation

on the possible lexical continations (the lexicon being represented by a trie-structure). The amount of lookahead is determined by the current pair which is to be licensed by `morphrule`.[9] Synchronization of surface and lexical string by insertion of null characters is also handled on the instantiation level.

The integration of the two-level relation into the general framework of the feature based sentence-level and word-level grammars is now performed by adding this relation as a principled constraint at the appropriate level.

In a definite clause style AVM notation this could be written as follows (given that `morphology/3` is a wrapper around the morphology relation given above starting with the empty contexts and hiding the nullified surface stream):

$$
\begin{bmatrix}
\text{PHON} & \boxed{2}\ string \\
\text{MORPH} & \boxed{3} \begin{bmatrix} \text{MSTRING} & \boxed{1}\ string \\ \text{STEM} & string \\ \text{MHEAD} & mhead \end{bmatrix} \\
\text{HEAD} & head \\
\text{SYNSEM} & synsem \\
word
\end{bmatrix} \leftarrow morphology(\boxed{1},\boxed{2},\boxed{3})
$$

The actual implementation as a principled constraint in our formalism additionally takes care of delaying the actual enforcement of this relation in case none of the strings is sufficiently instantiated.

A second provision has to be made in the word level grammar to assure proper concatenation of the lexical strings of the morphological signs being combined. Given the subtyping of *msign* into *marg* and *mfunctor*, which in turn has the subtypes *leftfunctor* and *rightfunctor*, the principled constraints ensuring concatenation of a left functor with its argument are shown in Fig. 5. Concatenation is delayed until the argument's MSTRING is instantiated. Thus, infinite loops when concatenating are avoided.

---

[9]This interaction and the lexicon lookup of the feature structure corresponding to the current morph taking place when encountering a morph boundary is not shown for the sake of simplicity.

8

```
concat_right_functor(X) :-
        X::=rightfunctor,
        X::arg:mstring===subtype_of(string)
  ===>
        X::arg:mstring===Arg,X::affix===Suffix,X::mstring===Mstring,
        concat(Arg,Suffix,Mstring).
```

Figure 5: Concatenation of lexical strings

As an example we demonstrate how these constraints interact in forming the third person singular present tense form of the German verb *raten* (to guess). The lexical string is composed of the stem *rAt* and the suffix *+t*. The lexical entries of these two morphs are given in Fig. 6.
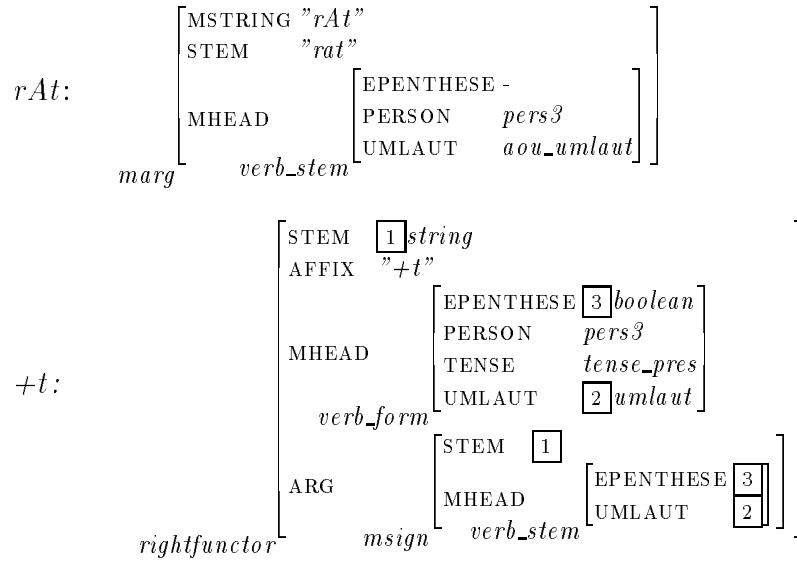

Figure 6: Lexical entries

The two-level rules applicable for this example are the *t-elision* rule (Fig.3) and two rules with filters licensing *a-umlaut* and *epenthesis*, given in the input notation for our system (Fig.7).

Interaction between syntactic and morphological processes takes place at the word level. Also the application of the two-level rules relating the surface string (i.e the PHON-value of the *word*) and the lexical-string (i.e. MORPH|MSTRING) is triggered here. This interaction is completely neutral with respect to the direction of processing due to its relational nature. Parsing is performed by simply instantiating the PHON value. Generation can be achieved when MORPH|MSTRING is present, which in turn is obtained by concatenating the lexical

A-umlaut: `_ <=> A:"a <=> _ :- filter(X, [X::mhead:umlaut===aou_umlaut])`
Epenthesis `dental <=> '+':e <=> s_or_t :- filter(X, [X::mhead:epenthese==='+'])`

Figure 7: Filter Rules

strings of the *msigns* instantiated by the morph grammar.

As a result of this constraint interaction the structure shown in Fig. 8 is obtained. Syntactic features relevant on the sentence level (such as PERSON and TENSE) are per-
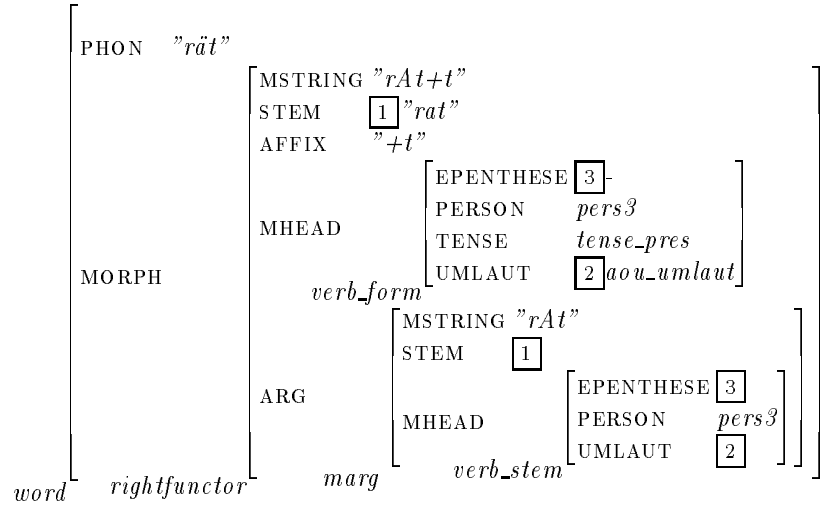

Figure 8: Result of constraint interaction (simplified)

colated from MORPH|MHEAD to SYNSEM|LOC|CAT|HEAD via structure sharing constraints attached to the type *word* (this interaction is not shown in Fig. 8). Information on subcategorization and semantic content for the word is obtained from the lexeme lexicon using MORPH|STEM as a key. These constraints complete the interaction between syntactic and morphological processing at the word-level.

6 Conclusion

We have presented a framework for the tight integration of word level and sentence level processing in a unification-based paradigm. The system is built upon a unification engine implemented in a CLP language supporting types and definite relations as part of feature descriptions. Using this extended feature formalism which is independently motivated by

requirements of standard HPSG, a reimplementation of X2MORFwas integrated into the grammar as a spezialized relation.

This architecture has computational as well as linguistic advantages. Integrating morphology and morphophonology directly into the grammar is in the spirit of HPSG which views grammar as a relation between the phonological (or graphemic) form of an utterance and its syntactic/semantic representation. This way also the treatment of phenomena transcending the boundary between morphology and syntax is made possible.

On the implementation side, the practical problems of interfacing two inherently different modules are eliminated. For applications this means that using a morphological component is made easy. Nevertheless, this tight integration still leaves morphology and syntax/semantics as autonomous components, enabling direct use of existing data sets describing morphophonology in terms of the two-level paradigm.

References

Abramson H.: A Logic Programming View of Relational Morphology, in Proceedings of the 15th International Conference on Computational Linguistics, August 23-28, 1992, Vol.III, pp.850-854, 1992.

Bird S., Klein E.: Enriching HPSG Phonology, University of Edinburgh, UK, Research Paper EUCCS/RP-56, 1993.

Carpenter B., Pollard C., Franz A.: The Specification and Implementation of Constraint-Based Unification Grammars, Proceedings of the Second International Workshop on Parsing Technology, Cancun, Mexico, 143-153, 1991.

Carpenter B.: *The Logic of Typed Feature Structures*, Cambridge University Press, Cambridge Tracts in Theoretical Computer Science 32, 1992.

Goldsmith J.A.: *Autosegmental and Metrical Phonology*, Basil Blackwell, Oxford, 1990.

Holzbaur C.: Metastructures vs. Attributed Variables in the Context of Extensible Unification, in Bruynooghe M. and Wirsing M.(eds.), Programming Language Implementation and Logic Programming, Springer, LNCS 631, pp.260-268, 1992.

Jaffar J., Lassez J.L.: Constraint Logic Programming, in Proceedings 14th ACM POPL Conf., Munich, 1987.

Krieger H.-U., Pirker H., Nerbonne J.: Feature-based Allomorphy, Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics, Columbus, Ohio, pp.140-147, 1993.

Matiasek J., Heinz W.: A CLP Based Approach to HPSG, Österreichisches Forschungsinstitut für Artificial Intelligence, Wien, TR-93-26, 1993.

Pollard C.J., Sag I.A.: *Information-Based Syntax and Semantics*, University of Chicago Press, Chicago, 1987.

Pollard, C.J, Sag I.A.: *Head-Driven Phrase Structure Grammar*, To be published by University of Chicago Press and CSLI Publications, in press.

di Sciullo A.-M., Williams E.: *On the Definition of Word*, MIT Press, Cambridge, MA, 1987.

Sproat R.: *Morphology and Computation*, MIT Press, Cambridge, MA, ACL-MIT Series in NLP, 1992.

Trost H.: The Application of Two-Level Morphology to Non-Concatenative German Morphology, in Karlgren H.(ed.), Proceedings of the 13th International Conference on Computational Linguistics, Helsinki, Finland, pp.371-376, 1990.

Trost H.: X2MORF: A Morphological Component Based on Augmented Two-Level Morphology, in Proceedings of the 12th International Conference on Artificial Intelligence, Morgan Kaufmann, San Mateo, CA, pp.1024-1030, 1991.

Trost H.: Coping with Derivation in a Morphological Component, in 6th Conference of the European Chapter of the Association for Computational Linguistics, Utrecht, pp.368-376, 1993.