

Feature Structures, Unification and Finite-State Transducers

Rémi Zajac

Computing Research Laboratory, New Mexico State University

zajac@crl.nmsu.edu

We present a new framework for describing morphological models which combines several types of descriptions in a single unified declarative language: simple morphological rules relate strings to feature structures; rules can be grouped into inheritance hierarchies of paradigms; rules can be composed for describing simple agglutinative morphology; they can also be combined to describe more complex morphotactic structures. Rules are compiled as extended finite-state transducers where left projections are characters and right projections are feature structures: unification is used instead of concatenation to compute the output of a transduction.

1 Introduction

Language-related work at NMSU/CRL focuses today on the development of linguistic processing models for a large variety of languages with a realistic coverage and with limited resources. In our research projects, it is not realistic to spend more than six months to develop an adequate morphological grammar for a given language, and we very clearly feel the need for higher level descriptive devices, even more so since some of the linguists who are working on the particular have little computational background. In this paper, we describe a new framework for the specification of morphological models, the Samba language, which combines, in a unified declarative framework, several types of descriptions allowing for the specification of:

- simple affixations and infixations on a stem using regular expressions on character strings and typed feature structures for specifying morphological properties associated with a string pattern;
- paradigmatic morphology using tables of rules, themselves organized in an inheritance hierarchy.
- agglutination of affixes by prefixation or suffixation by simple (binary) composition of rules (or tables) and affixes.
- complex morphotactics by using regular expressions of rules (or tables).

Work on languages with complex morphology such as Arabic, Korean or Turkish shows that the difficulty in developing a computational model of morphology has been often underestimated. In many NLP systems, the morphological analyzer is often implemented directly in some programming language such as C. One reason can be the lack of a general declarative formalism powerful enough to describe complex morphological models; another reason can be that powerful formalisms are often complex and difficult to learn: the direct use of a general programming language might require less effort in the development of a morphological analyzer. This has, for example, been the case at the Computing Research Laboratory, where morphological analyzers for Arabic, Spanish, Russian and Serbo-Croatian were developed using C, Lisp or Prolog for the lack of a system that would be sufficiently easy to learn and would match the intuition of linguists about the structure of a morphological model (Sheremetyeva et al. 97). In particular, as noted by (Anick & Artemieff 92), many formalisms derived from the two-level model (Koskenniemi 83) concentrate on orthographic and affixation rules, whereas a large part of the morphological description of languages such as Spanish or Serbo-Croatian focus on capturing regularities between and within paradigms. Two-level formalisms are also difficult to use to describe complex affixation or infixation phenomena (Ofazer 93, Beesley et al. 89, Kiraz 94). There are however formalisms which are explicitly based on the notion of paradigm. For example, in the Morfogen system (Pentheroudakis & Higginbotham 91), the basic unit of description is a paradigm defined as a set of tables which describe the different inflectional patterns of words. The system described in (Anick & Artemieff 92) is based on the same ideas within a more declarative framework. Declarative

systems based on string unification such as the one described by (Calder 89) are also very attractive but more difficult to implement; they also lack the power to describe complex affixation phenomena as well as morphotactics in a simple way.

The work described in this paper can also be related to the first efforts in building generic unification-based parsing systems such as D-PATR (Karttunen 86). Earlier parsing systems were built using (augmented) context-free parsers: the grammar was a pure context-free grammar sometimes augmented with attributes or actions. The output of an analysis was a derivation tree with property values on the nodes. In unification-based parsing system, the backbone is still a context-free parser, but rules are augmented using feature structures and equations over feature structures and the output is a feature structure. By comparison, many current morphological systems are based on finite-state technology, and the most popular ones use some version of two-level morphology (Koskeniemi 83) which is based on finite-state transduction: left projections are characters and right projections are also characters. In Samba, we depart from this basic model, and in a way similar to unification-based parsers, we retain the finite-state backbone, but instead of using characters in the right projection, we use feature structures, and instead of using concatenation to build the output of the transducer, we use unification on feature structures: the output of the transducer is a feature structure, not a string of characters.

We face, of course, the same efficiency issues that were raised for unification-based parsers, and techniques developed for unification-based parsers for dealing with these issues can be adapted to our morphological system. It is clear that such a system will be outperformed by traditional finite-state transducers in the same way that unification-based parsers are outperformed by simpler context-free parsers. However, we hope that the finite-state backbone will help in keeping runtime performances at a reasonable level, and we believe that a unification-based morphological system will bring us the same benefits that were sought from unification-based parsers: declarativity and concise formulation of linguistic knowledge. The initial implementation provides an interpreted mode only which is necessary for debugging the morphological model. It also makes it easier to implement and debug the Samba system itself. This choice does not preclude a compiled mode which will be necessary for building realistic applications.

There have been several proposals to integrate feature structures and finite-state models for morphology which have been sources of inspiration for our work. Calder (1988) and Bird (1992) first suggested the use of feature structures to describe morphological models. Trost and Matiasek (1994) integrate an augmented two-level model (X2MorF) in a unification-based system written in Prolog. Krieger et al. (1994) suggest an encoding of morphophonemics and morphotactics using feature structures to represent finite-state machines and use unification on feature structures to implement operations on finite-state machines (intersection, etc.). Our work concentrates on the definition of a high-level descriptive language for morphology based on regular expressions, feature structures and inheritance, and on an implementation of a parsing and generation system allowing for rapid development of formal morphological models in the spirit of (Carter 1995).

2 The Samba language

Paradigmatic inflectional morphology

The basic element of a morphological description is a *morphological rule* which associates a *form* representing a sequence of morphemes to a linguistic *structure*, a set of morphological features. The *form* itself is formally represented as a regular expression on characters. The linguistic *structure* describes how the morphological features of the stem and the morpheme are combined. For example, the following rule describes the conjugation of a French verb belonging to the paradigm Chanter in the first person singular of

the indicative present tense (string variables are prefixed with the dollar sign, regular expressions are enclosed between angle brackets):

```
[form: <$chant 'e'>, /* A simple suffixation */
 structure: [infl: Chanter, /* Paradigm of the stem */
 stem: <$chant>, /* Form of the stem */
 tense: Present, mood: Ind, /* Conjugation information */
 num: Sing, pers: First]] /* Morpheme information */
```

A Samba morphological rule describes the concatenation of stems and morphemes (using regular expressions) and the combination of morphological features of words and morphemes (using feature structures and unification). Stems and their features are stored in the lexicon: a lexicon entry is a feature structure, and the `structure` zone in a rule is a partial description of an entry. In the example above, the `structure` describes a set of words which belong to the inflectional paradigm `Chanter` (`infl` feature). The stem of the verb is specified under the feature `stem` as the string variable `$chant` and this variable is used in the regular expression describing the inflected *form*.¹ The other features in the `structure` encode the morphological properties of this particular *form* by specifying values for tense, mood, number and person.

Rules can be grouped together in *tables* describing forms that belong to the same paradigm. A table is simply a disjunction of rules which share some common information: `VPI_ER` below is an example of a table for a sample of French verb morphology² which matches traditional descriptions given in French grammar books. Formally, a table is defined as follows, using the language of feature structures:

```
TableId = [paradigm: <paradigm>, features: <features>, rules: <rules>];
```

The `paradigm` feature structure defines the *sub-set of lexical entries* which belong to the paradigm thus defined. The paradigm described in table `VPI_ER` (Present Indicative for -er verbs) below is reduced to two features and specifies both the morphological inflectional paradigm of the verbs and their stems. For example, the verb ‘ajouter’, which belongs to the ‘chanter’ paradigm, will have [`stem: 'ajout'`] in the dictionary and the table simply refers to the value of this stem using the `$chant` variable. The second feature structure under `features` specifies the set of morphological features (tense, mood, mode, etc.) that is associated to the *conjugation table as a whole*: it represents the common contribution of all rules of the table. The successful application of a rule will add (unify) this structure to the output feature structure. The rules in the table are grouped under the `rules` feature where each rule is named individually. A rule needs only to specify its own particular contribution. The following morphological table defines a set of rules which describe a conjugation of French verbs of the first group (i.e., the `Chanter` paradigm) as concatenations of a stem and suffixes.

```
VPI_ER =
[paradigm: [infl: Chanter, stem: <$chant>],
 features: [tense: Present, mood: Ind],
 rules:
  [sg1: [form: <$chant 'e'>, structure: [num: Sing, pers: First]],
   sg2: [form: <$chant 'es'>, structure: [num: Sing, pers: Second]],
   sg3: [form: <$chant 'e'>, structure: [num: Sing, pers: Third]],
   pl1: [form: <$chant 'ons'>, structure: [num: Plu, pers: First]],
   pl2: [form: <$chant 'ez'>, structure: [num: Plu, pers: Second]],
   pl3: [form: <$chant 'ent'>, structure: [num: Plu, pers: Third]]];
```

1. This is a simplified example. In most cases, the stem is derived from the citation form and the relation between the citation form and the stem will be expressed using regular expressions with variables (see below).
2. Examples are adapted from (Pentheroudakis and Higginbotham 91).

The above examples use only suffixation, but a *form* is a full regular expression and allows us to express prefixation and infixation as well as suffixation. As an example of a combination of prefixation and suffixation, German past participles in the ‘leben’ paradigm (‘gelebt’) can be analyzed as follows:

```
VPP_LEB = /* A table with a single rule */
         [paradigm: [infl: Leben, stem: <$leb>],
          rules: [pp: [form: <'ge' $leb 't'>, structure: PastParticiple]]];
```

Agglutinative morphology

Agglutinative morphology can be described using composed tables that combine descriptions of other tables. Simple tables describe inflections on stems stored in the lexicon (as described above); composed tables describe further inflections on a *form* described in other tables, simple or composed. Besides composed tables, one can also use regular expressions (on morphological rules or tables) to describe more complex structures. Table composition and regular expressions allow for simple descriptions of agglutinative morphology as well as derivational morphology (when derivational morphology is based on prefixation or suffixation). However, the formalism forbids the type of recursion that would make the formalism truly context-free. In composed tables, the base table used in the composition is supplied in the base declaration: it has the same structure as a morphological rule with *form* and *structure* features:

```
TableId = [base: [<form>, <structure>], <rules>;
```

The *base* feature holds the result of the application of the rules of the base table: the *form* holds the *form* of the word as described in the base table, and the *structure* holds the corresponding partially specified feature structure. The definitions below are an example of an analysis for French participles: the two first tables are composed tables; the third is a regular expression (a simple disjunction in this case), and the last two are simple tables.

```
VPAG_N =
[base: VPA_G[form: <$vpa>],
 rules: [sing: [form: <$vpa ` `>, structure: [num: Sing]],
         plu: [form: <$vpa `s`>, structure: [num: Plu]]];
```

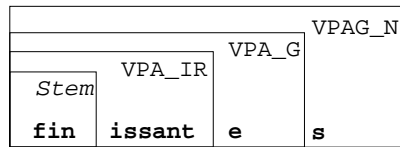
```
VPA_G =
[base: VPA_EIR[form: <$vpa>],
 rules: [masc: [form: <$vpa ` `>, structure: [gen: Masc]],
         fem: [form: <$vpa `e`>, structure: [gen: Fem]]];
```

```
VPA_EIR = <VPA-ER | VPA-IR>;
```

```
VPA_ER =
[paradigm: [infl: Chanter, stem0: <$chant0>],
 rules: [presPart: [form: <$chant0 `ant`>, [structure: PresentParticiple]],
         pastPart: [form: <$chant0 `e`>, [structure: PastParticiple]]];
```

```
VPA_IR =
[paradigm: [infl: Finir, stem0: <$fin0>],
 rules: [presPart: [form: <$fin0 `issant`>, [structure: PresentParticiple]],
         pastPart: [form: <$fin0 `i`>, [structure: PastParticiple]]];
```

Applying this description to plural feminine form of the present participle of ‘finir’ (to end), ‘finissantes’, yields the following decomposition:



3 Transduction in Samba

A Samba grammar is composed of a set of definitions of rules, tables or regular expressions over rules or tables. A set of definitions is compiled as a finite-state transducer where the lower part of the transduction (left projection) is a string and the higher part (right projection) is a feature structure. In such a finite-state machine, an edge carries a string (input) and a feature structure (output). From an analysis point of view, the inflected form is decomposed into lexemes and morphemes by traversing the network using the left projection, as in standard finite-state transducers. However, instead of simply concatenating the symbol of the right projection to build the output, the transducer uses the right projection to build a single feature structure by using unification instead of concatenation. For each edge traversed, the transducer unifies each right projection with the output feature structure: the structural contribution of each morpheme or lexeme is added to the word structure by unifying in the feature structure representing the morpheme. From a generation point of view, the network is traversed based on the right projection using feature unification instead of symbol equality, and concatenating the string elements of the left projection to build the inflected word form.

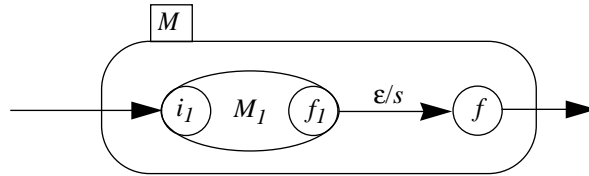
Formally, a Samba transducer T is a tuple (I, O, S, s, F, δ) where I is the input alphabet, O the output alphabet, S is a finite set of states, s is the initial state, F is a set of final states, δ is the transition function from $S \times I \times O$ to S . The input alphabet I is a (finite) set of characters. The output alphabet O is an (infinite) set of typed feature structures specified by a set of type definitions (Zajac 92). The compilation of a set of Samba definitions as a transducer is done in four major successive steps:

1. All tables are expanded as disjunctions of simple morphological rules. This step transforms all Samba table definitions as definitions which have regular expressions on morphological rules as right-hand sides: the leaves of the regular expression trees are morphological rules.
2. In all Samba definitions, all morphological rules are then compiled as finite-state transducers producing regular expression trees where leaves are FSTs.
3. These regular expressions are compiled as finite-state transducers where the leaves of the regular expressions becomes sub-FSTs.
4. In a definition, all references to other definitions are then replaced with the corresponding FSTs (this process terminates since recursion is not allowed).

Rules and dictionary entries

A morphological rule is a pair $\langle f, s \rangle$ where the *form* f is a regular expression on characters (the left projection) and the *structure* s is a typed feature structure (the right projection). It is compiled in an FST M in two steps. First, the *form* f is compiled as a finite-state transducer M_f where all right projections are empty (the empty feature structure \top is the top of the lattice of feature structures). Then, from the (single) final state

f_1 of this intermediary FST, we add one more epsilon-transition to the final state f of M where the right projection is the feature structure s (transition ϵ/s). The initial state i of M is the initial state i_1 of M_1 :



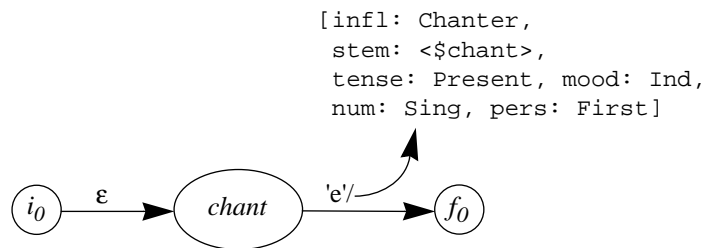
In the *form* of a rule, there will usually be string variables that are also present in the *structure* of the rule. Typically, the *structure* will specify the value of a stem and this variable will be used in the definition of the *form*. We impose the following restrictions on variables:

1. String variables in the *structure* refer to string values in a dictionary entry.
2. Any variable in the regular expression defining the *form* must also appear in the *structure*.

These constraints are imposed in order to guaranty finiteness of substitutions. With these constraints, the compilation of a rule is done as follows. First, all string variables in the *structure* are instantiated by querying the dictionary for all entries subsumed by the *structure*. Then, the occurrences of the variables in the *form* are replaced with FSTs representing these strings. For example, the following rule:

```
[form: <$chant 'e'>,
 structure: [infl: Chanter,
            stem: <$chant>,
            tense: Present, mood: Ind,
            num: Sing, pers: First]]
```

will be compiled as the FST pictured below. The variable $\$chant$ is instantiated by querying the dictionary for all entries subsumed by the *structure*: this produces a disjunction of pairs $\{ \langle f_i, s_i \rangle \mid f_i \text{ is a string (stem), } s_i \text{ is a dictionary entry} \}$. This disjunction is compiled as the sub-FST *chant*:



Tables

Tables are expanded as disjunctions of simple morphological rules. This step reformulates all Samba table definitions as regular expressions on morphological rules. A disjunction of morphological rules is simply a set $\{ \langle f_i, s_i \rangle \}$ of morphological rules. A simple table is a triple $\langle P, F, \{ \langle f_i, s_i \rangle \} \rangle$, where P is the paradigm (a feature structure), F represents the features of the table, and $\{ \langle f_i, s_i \rangle \}$ the set of rules. The compilation of a simple table rewrites the table as a set of rules:

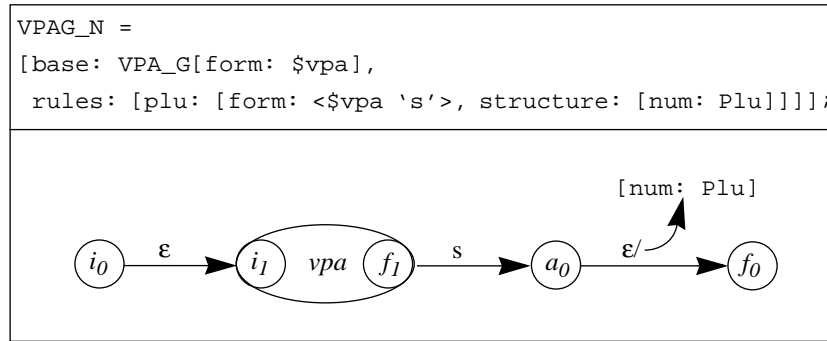
$$\langle P, F, \{ \langle f_i, s_i \rangle \} \rangle \rightarrow \{ \langle f_i, P \wedge F \wedge s_i \rangle \}$$

where $P \wedge F \wedge s_i$ is the unification of the feature structures P, F and s_i . The Samba compiler makes sure that the unification succeeds during the type checking phase of the compilation process.

A composed table is a pair $\langle \langle f, s \rangle, \{ \langle f_i, s_i \rangle \} \rangle$, where $\langle f, s \rangle$ is the base and $\{ \langle f_i, s_i \rangle \}$ the set of rules. The compilation of a composed table rewrite the table as a set rules:

$$\langle \langle f, s \rangle, \{ \langle f_i, s_i \rangle \} \rangle \rightarrow \{ \langle f \otimes f_i, s \wedge s_i \rangle \}$$

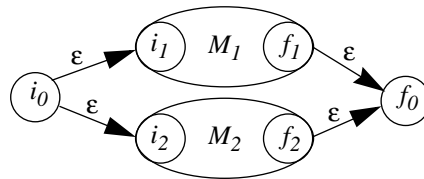
where $f \otimes f_i$ is the composition of the two regular expressions and $s \wedge s_i$ is the unification between the two feature structures. The compiler checks that the intersections are not empty and that the unifications are successful. The composition $f \otimes f_i$ is computed as follows. We impose constraints on variables similar to the one for rules: the *form* of the base f is a string variable and this variable must appear in each *form* f_i of the rules of the composed table. Each regular expression f_i is compiled as an FST, and each occurrence of the variable f in rules is replaced with the corresponding sub-FST: the base table is a sub-FST of the composed FST as shown in the example below.



Regular expressions

All Samba definitions which are regular expressions on rules are compiled as finite-state transducers as follows (we use standard construction algorithms, see e.g., Hopcroft & Ullman 79):

- The concatenation of two morphological rules is simply defined as the rule whose *form* is the concatenation of the forms of the two rules and whose *structure* is the unification of the *structures*: $\langle f_1, s_1 \rangle \bullet \langle f_2, s_2 \rangle \rightarrow \langle f_1 \bullet f_2, s_1 \wedge s_2 \rangle$.
- The Kleene closure is defined as $\langle f, s \rangle^* \rightarrow \langle f^*, s \rangle$.
- The disjunction follows the standard construction: $\langle f_1, s_1 \rangle \mid \langle f_2, s_2 \rangle$ is compiled as:



When a regular expression contains an identifier which stands for a Samba definition, the identifier is replaced with its compiled definition, exactly like in macro-expansion. Since recursivity is not allowed, this process terminates, producing a single FST which is defined as a composition of sub-FSTs.

The Samba interpreter

The Samba interpreter takes as input either a string (for analysis), or a feature structure (for generation). The interpreter is parametrized by the start state, specified by the user as the identifier of some Samba definition. The interpreter exits in the corresponding final state. This facility allows us to debug a Samba grammar definition by definition.

The Samba interpreter in analysis mode works as a standard finite-state transducer: input is read one character at a time. The main difference with classical FSTs is that the output of a transition is not appended to the output string; instead, the output of a transition (right projection) is *unified* with the output of the transducer: the transducer's output is initialized to the empty feature structure (top of the lattice of feature structures) instead of the empty string. Since unification can fail, a transition is traversed successfully only if the transition input (left projection) matches the current character input *and* if the unification of the transition output with the transducer's output succeeds. Traversing the graph of transitions, the right projection of each transition is unified with the global transducer's output. When reaching a final state, the output has accumulated all information defined in the right projection of each transition. For generation, the interpreter simply reverses the interpretation of input and output on transitions: when traversing a transition, the interpreter first unifies the input with the transition's right projection. If unification succeeds, it appends the transition's left projection to the output string and proceeds with the new input feature structure.

Since the transducer is non-deterministic, alternative branches must all be explored. Several implementations of the search are possible. For parsing, we use a variant of the algorithm for computing the intersection of two finite-state machines: the input word is represented as a linear finite-state machine and the output is also represented as a finite-state machine with a tree-like topology. The parsing algorithm computes the intersection of the transducer and the input word using the left projections of the transducer; during traversal, right projections are unified with the output. The unification algorithm for feature structures uses structure sharing (see e.g., Emele 91) which allows to minimize the amount of copying: the feature structures in various branches of the output graph record exactly the increment of information added by the corresponding transitions of the transducer.

4 Conclusion

Preliminary research in the morphological description of Arabic, Russian, Serbo-Croatian and Japanese using the Samba language supports our hypothesis that this formalism can be used for describing a variety of morphological models in a compact and declarative way. A mock-up of a Samba interpreter has been built to test various implementation choices and a complete implementation is currently under way at CRL. Future work on Samba will include the implementation of a version supporting dynamic access to dictionaries, the addition of orthographic rules, and an extension to handle composition. The Samba language is currently used to develop a morphological analyzer for Persian and will also be used for several other languages including for example Spanish and Korean.

Acknowledgments This work is part of the Corelli project funded by the Maryland Procurement Office, Fort George G. Meade, MD under grant MDA904-96-C-1040.

5 References

1. Peter Anick, Suzanne Artemieff. 1992. "A high-level morphological description language exploiting inflectional paradigms". In *Proceedings of Coling '92*, Nantes, 1992. pp67-73.
2. Stephen R. Anderson. 1992. *A-Morphous Morphology*. Cambridge University Press, 1992.
3. Robert Beard. 1995. *Lexeme-Morpheme Base Morphology*. State University of New-York Press, 1995.

4. Ken Beesley, Tim Buckwalter, Stuart Newton. 1989. "Two-level finite-state analysis of Arabic morphology". In *Proceedings of the Seminar on Bilingual Computing in Arabic and English*, 6-7 Sept. 1989, Cambridge, GB.
5. Steven Bird. 1992. "Finite-State Phonology in HPSG". In *Proceedings of Coling'92*, 23-28 August 1992, Nantes, FR. pp74-80.
6. Jonathan Calder. 1989. "Paradigmatic Morphology". In *Proceeding of the European ACL*, 1989.
7. David Carter. 1995. "Rapid development of morphological descriptions for full language processing systems". In *Proceedings of EACL'95*, pp202-209.
8. Marc Domenig, Pius ten Hacken. 1992. *Word Manager: A System for Morphological Dictionaries*. Olms.
9. Martin Emele. 1991. "Unification with Lazy Non-Redundant Copying". In *Proceedings of the 29th Annual Meeting of the ACL*, 18-21 June 1991, Berkeley, CA. pp323-330.
10. John E. Hopcroft, Jeffrey D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.
11. Ron Kaplan, Martin Kay. 1994. "Regular models of phonological rule systems". *Computational Linguistics* 20(3). pp331-378.
12. George A. Kiraz. 1994. "Multiple-Tape Two-Level Morphology: A Case study in Semitic non-linear Morphology". In *Proceedings of Coling'94*, 5-9 August 1994, Kyoto, Japan. pp180-186.
13. Kimmo Koskenniemi. 1983. "Two-level model for morphological analysis". In *Proceeding of IJCAI'83, the 8th International Joint Conference on Artificial Intelligence*, Karlsruhe, DE, 1983. pp683-685.
14. Hans-Ulrich Krieger, John Nerbonne, Hannes Pirker. 1994. "Feature Based Allomorphy". DFKI Research Report RR-93-28, Saarbrücken, DE.
15. Peter H. Matthews. 1974. *Morphology: An Introduction to the Theory of Word Structure*. Cambridge University Press.
16. Kemal Oflazer. 1993. "Two-level description of Turkish morphology". In *Proceedings of the 6th Conference of the European Chapter of the ACL*, April 1993.
17. Joseph E. Pentheroudakis, Dan W. Higginbotham. 1991. "Morfogen: a morphology grammar builder and dictionary interface tool". Presented at the 1991 Meeting of the Deseret Language and Linguistics Society, Brigham Young University, Provo, Utah.
18. Graeme D. Ritchie, Graham J. Russell, Alan W. Black, Stephen G. Pulman. 1992. *Computational Morphology. Practical mechanisms for the English Lexicon*. The MIT Press.
19. Emmanuel Roche, Yves Schabes (eds.). 1997. *Finite-State Language Processing*. The MIT Press.
20. Richard W. Sproat. 1992. *Morphology and Computation*. MIT Press.
21. Svetlana Sheremetyeva, Wanying Jin, Sergei Nirenburg. 1997. "Rapid Deployment Morphology". Technical Report MCCS-97-313, Computing Research Laboratory, New Mexico State University.
22. Harald Trost, Johannes Mataisek. 1994. "Morphology with a null interface". In *Proceedings of Coling'94*, 5-9 August 1994, Kyoto, Japan. pp141-147.
23. Evelyne Tzoukermann, Mark Y. Liberman. 1990. "A finite-state morphological processor for Spanish". In *Proceedings of Coling'90*, Helsinki, Finland. pp277-282.
24. Rémi Zajac. 1992. "Inheritance and Constraint-based Grammar Formalisms". *Computational Linguistics* 18 (2), June 1992. pp159-182.