# A Finite State Approach to German Verb Morphology

Günther GÖRZ
IBM — WT LILOG
Schloßstr. 70, D-7000 Stuttgart 1, W. Germany
(on leave from Univ. of Erlangen-Nürnberg)
Goerz@SUMEX-AIM.STANFORD.EDU, GOERZ@DS0LILOG.BITNET

Dietrich PAULUS
Univ. of Erlangen-Nürnberg, IMMD V
Martensstr. 3, D-8520 Erlangen, W. Germany
Paulus@faui53.uucp

**Abstract**

This paper presents a new, language independent model for analysis and generation of word forms based on Finite State Transducers (FSTs). It has been completely implemented on a PC and successfully tested with lexicons and rules covering all of German verb morphology and the most interesting subsets of French and Spanish verbs as well. The linguistic databases consist of a letter-tree structured lexicon with annotated feature lists and a FST which is constructed from a set of morphophonological rules. These rewriting rules operate on complete words unlike other FST-based systems.

## 1 Introduction

Until the beginning of this decade, morphological parsers usually were restricted to one particular language; in fact we do not know of any one which was language independent or even applicable to a wide class of non-trivial inflected languages. In the meantime, the situation has changed a lot through the usage of Finite State Transducers (FSTs). Although the formalism of generative phonology seems to be powerful enough to cover almost any language, it is very difficult to implement it computationally. Recent approaches to compile the rules of generative phonology into finite automata offer solutions to both problems. In the following, we report on a successful and complete application of this technique to the morphology of German verbs. To demonstrate its generality, it has also been applied to a large subset of French — in fact the most interesting cases — and some Spanish verbs.

## 2 The Finite State Approach

As Gazdar /1985/ [1] observed, only very few papers on the mathematical foundations of modern phonology exist. He quotes from Johnson's /1970/ PhD thesis [2], the earliest study of this kind, who states that "any theory which allows phonological rules to simulate arbitrary rewriting systems is seriously defective, for it asserts next to nothing about the sorts of mappings the rules can perform" (/Johnson 1970/ [2], p. 42). According to Gazdar /1985/ ([1], p. 2) Johnson "proves that a phonology that permits only simultaneous rule application, as opposed to iterative derivational application, is equivalent to an FST. And he then argues that most of the phonology current around 1970 could either be formalized or reanalyzed in terms of simultaneous rule application, and could thus be reduced to FSTs."

At the Winter LSA meeting at New York in December 1981, R. Kaplan and M. Kay gave a talk — a written account does not exist — in which they showed "how the iteratively applied rules of standard generative phonology could, individually, be algorithmically compiled into FSTs" (/Gazdar 1985/ [1], p. 2) under the constraint that rules may not be reapplied to their own outputs. Such a finite ordered cascade of FSTs can be collapsed into a single FST whose behavior is equivalent to that of the original generative rules (cf. /Kay 1983/ [4], p. 100–104). A FST is a special kind of finite automaton which operates simultaneously on an input and an output tape such that it inspects two symbols at a time. In Kay's approach, the FSTs carry two labels, each label referring to one of the two tapes. In general, a FST is said to *accept* a pair of tapes if the symbols on them match a sequence of transitions starting in an initial state and ending in one of the designated final states. If no such sequence can be found, the tapes are *rejected*. To allow tapes of different length to be accepted, a symbol to be matched against one or other of the tapes to do a transition may be empty, in which case the corresponding tape is ignored.

There are two advantages with this approach: The first is, that such a combined FST can be implemented easily as a simple and very efficient program. Second, unlike ordered sets of rewriting rules, there is no directionality in principle, so that the same machine can be used for analysis and generation as well.

Kimmo Koskenniemi /1983a, 1983b, 1984, 1985// ([5], [6], [7], [8]) took up this approach and applied a variation of it to some heavily inflected languages, first of all to Finnish. His "two-level" model proposes parallel rules instead of successive ones like those of generative phonology. The term "two-level" is supposed to express that there are only two levels, the lexical and the surface level, and that there are no intermediate ones, even logically. Besides its simplicity — in particular with respect to implementation — the problematic ordering of rules is avoided.

## 3 A Parallel Rewriting Variant of FSTs with Feature Unification

Although Koskenniemi's machinery works in parallel with respect to the rules, rewriting is still performed in a sequential manner: each word form is processed letter by letter (or morpheme by morpheme) such that all replacements are done one at a time. Certainly this model does not depend on the processing direction from left to right, but at any time during processing it focusses on only one symbol on the input tape.

It is precisely this feature, where our approach, based on a suggestion by Kay, differs from Koskenniemi's. Our work grew out of discussions with M. Kay, to which the first author had the opportunity during a research stay at CSLI, Stanford, in summer 1985. Without his help our investigations would not have been possible. In our system, rewriting is performed over *complete* surface words, not letters or morphemes. There is no translation from lexical to surface strings, because there is only one level, the level of surface strings. Rewriting is defined by rules satisfying the scheme

$$Pattern \rightarrow Replacement$$

where both, *Pattern* and *Replacement*, are strings that are allowed to contain the wild card "?" character which matches exactly one (and the same) letter. Let $a, b, w_1, w_2 \in \Sigma^*$ where $\Sigma$ is an alphabet. For all $w_1, w_2$ the rule $a \rightarrow b$, with *Pattern*= $a$ and *Replacement*= $b$, rewrites $w_1 a w_2$ to $w_1 b w_2$. It should be noted that only one occurrence of the *Pattern* is rewritten. Furthermore, it can be specified whether the search is to be conducted from left to right or vice versa. Hence, it is possible to perform *rewriting in parallel* in contrast to Koskenniemi's sequential mode.

The rules are attached to the edges of a FST; hence the application order of the rules is determined by the sequence of admissible transitions. Conflicts arising from the fact that at a given state the patterns of several rules match are resolved by the strategy described in sec. 5.

Matching of the left hand side of a rule is only one condition to do a transition successfully. The second condition is that the list of morphosyntactic features of the actual item can be successfully *unified* with the feature list attached to the resp. edge of the automaton.

The required unification procedure realizes a slightly extended version of the well known term unification algorithm. The objects to be unified are not lists of functors and arguments in a fixed order with fixed lengths, but sets of attributes (named arguments) of arbitrary length. The argument values, however, are restricted to atomic objects, and therefore not allowed to be attribute lists themselves (as it is the case with the recursively defined functional structure datatype in unification grammars).

*Example 1:*

Note that words are delimited by angle brackets such that affixes can be substituted for the empty string at the beginning or end of a word.

- Some rewriting rules

  a → o
  m → mm
  >→ en>

- Corresponding automaton fragment

```
((start nil
   ("a" "o" st1 ((tempus imperf) (group 1)))
   ...)
 (st1 nil
   ("m" "mm" st2 ((group 1))))
 (st2 nil
   (">" "en>" end ((pers 1) (num sing)
                   (mode indic)))
   ...)
 (end t))
```

This automaton fragment generates "<kam>" with the feature list ((tempus imperf) (group 1) (num sing) (mode indic) (pers 1)) from the infinitive form "<kommen>".

Currently, there is no compiler which generates an automaton from a given set of rules like the one by Karttunen et al. /1987/ [3], i.e. the automaton has to be coded manually.

## 4 The Lexicon

In order to achieve fast access and to avoid redundancy wherever possible, the lexicon is realized as a letter tree with annotated feature lists for terminal nodes.

*Example 2:* A section of the letter-tree lexicon containing "wagen", "wiegen", and "wägen".

```
(* (\w (\a (\g (\e (\n (\+ ((group 2)))))))
     (\i (\e (\g (\e (\n (\+ ((group 4))))))))
     (\a (\g (\e (\n (\+ ((group 3)))))))))
```

## 5 The Control Strategy

Our implementation follows Kay's suggestion, in that processing — analysis and generation as well — is done in two essential steps: First, along a path beginning at a start state, for all applicable rules the attached feature unifications are performed until a final state is reached. The search strategy is depth-first, i.e. at each state the first applicable rewriting rule in the list of transitions is selected. In a second phase, such a successful path is traced back to its origin with simultaneous execution of the corresponding rewriting rules. For rewriting, a device called *exclusion list* is employed, which allows to combine several distinct rules into one unit (which has been omitted in example 1 for the sake of simplicity). This adds a further restriction to transitions: A transition is blocked if the pattern of the corresponding rule matches, but is contained in the exclusion list.

## 6 German Verb Morphology

In German, inflected verb forms exist for the four tense/mode combinations: present tense/ indicative, present tense/ conjunctive, past tense/ indicative and past tense/ conjunctive. Furthermore there are two participles (present and perfect) and two imperative forms derived from the infinitive verb stem. This adds up to 29 possibly different forms per verb.

With respect to inflection, German verbs can be divided into three classes: regular "weak" verbs ("schwache Verben"), "strong" verbs ("starke Verben") and irregular verbs.

Inflection of weak verbs is done by simply adding a suffix to the stem. In the special case of the past participle the prefix "ge-" is added too. This class can easily be handled by existing algorithms, like the one of Kay described above.

Inflection of strong verbs is also done by adding to the stem suffixes, which slightly differ from the ones used with weak verbs. In addition to the change of the ending, the stem itself may vary, too. In most cases it is not the whole stem that changes, but only one special vowel in the stem, the stem vowel.

This change introduces the problems that make an extension of the existing algorithm necessary.

In most cases irregular verbs can be treated like regular strong verbs with the exception of some special forms.

*Example 3:* "sein" (engl.: to be)

To conjugate the verb in past tense ("ich war", "du warst", ...), conjugate "war-" as a regular strong verb.

The following fourteen graphemes can be stem vowels: "a", "e", "i", "o", "u", "ä", "ö", "ü", "ei", "ai", "au", "äu", "eu" and "ie".

When conjugating a verb, the stem vowel may change up to six times, as the following example demonstrates:

| *Form* | *Infl. Type* | *Grammatical Description* |
|---|---|---|
| ich helfe | (1) | present 1st person |
| du hilfst | (2) | present 2nd person |
| er half | (3) | past tense |
| er hülfe (rarely used, but correct) | (4) | past tense conj. |
| er hälfe | (4) | past tense conj. |
| er hat geholfen | (5) | past participle |

This gives rise to the combinatorial explosion of $14^6$ possible series of stem vowels for each verb conjugation ("paradigm"). Only a small number of those are actually used in the language, but even this number is too big to be handled easily by one of the described algorithms.

## 7 Hard Problems in German Verb Inflection

The following problems are hard to be solved by any one of the existing algorithms:

- How can the stem vowel be located? This may be difficult, especially when compound verbs are to be analyzed, like "beherzigen".
- Given an inflected verb form, how can we find the infinitive stem from which this form is derived? *Example:* "wöge": "wagen"? or "wiegen"? or "wägen"?
- How can the lexicon be kept small; i.e. can we get around adding all the possible changes of the stem to the lexicon?

The general idea behind our solution is to build a "shell" around Kay's generic two-state-morphology scheme which takes care of the special stem vowel problems in German verbs. The core of this scheme, which is the rewriting-rule algorithm, remains unchanged and adds all appropriate affixes to the stem. This leads to an algorithm that can generate all forms of any German verb, even of a

prefixed verb, and analyze these forms as well. One important part of the extended algorithm is a matrix called the stem-vowel table which contains all the information about the vowel series occurring in the conjugations of one verb. After some compression and combination of related series the size of the table is 40*5 lists of characters. This matrix is organized in the following manner:

There are five columns corresponding to the five cases of stem vowel change in example 3. Each entry in a column is a list of characters; mostly this list has length one. (The fourth element of the list corresponding to the verb "helfen" would have the two elements "ü" and "ä").

The rows list all the possible combinations of vowel change that occur in the present use of the language.

The shell consists of five basic parts (placed in order of the way they are called when the algorithm *generates* forms):

1. A routine for locating the stem vowel and replacing it by a generic symbol; it is realized by a simple function.

2. An algorithm that separates prefixes from the stem when a compound verb is to be analyzed. It also strips off the infinitive ending. This is done by a simple lookup in the prefix table.

3. A lexicon module which also adds some default information to the grammatical information obtained from the lexicon entry. Irregular and strong verbs get a group number added to the feature list. The prefix, if one is found, is compared with the list of permissible prefixes in the lexicon.

4. The core of the algorithm uses an automaton and rewriting rules to modify the affixes of the verb. In the course of unification new attributes are added to the feature list. In particular, if the verb is strong or irregular, information about the stem vowel is added to the list. The new information contains an offset into the stem vowel table.

5. The generic symbol is replaced by the stem vowel indicated by the feature list using a single rewriting rule. The new vowel is looked up in the table which is indexed by two values in the feature list, namely the group number of the verb (which is either defaulted or part of the lexical information), and a column number, which is added by the automaton.

## 8 Further Enhancements to Keep the Analysis of Verbs Fast

The main problem with the analysis of German verb forms is to find the infinitive stem belonging to the stem. As soon as this stem is found, the search tree can be pruned considerably. This is because the lexicon information of the infinitive form may restrict the possible unifications when stepping from one state of the automaton to another one.

This problem has been solved in the following way. Given an inflected form with a possible changed stem vowel, we can at least find the position of the actual stem vowel. We can also strip off the ending and the prefix, if one exists (e.g. "erwöge" [infinitive "erwägen"] → "wXg-"). This leads to a rather peculiar structure for the lexicon. The lexicon mainly contains verb infinitives in an encoded form. The stem vowel of the infinitive is replaced by a place holder, the stem vowel is added to the end of the form, separated from the stem by a hyphen. Stem vowels consisting of more than one character are encoded as a single symbol.

*Example 4:*

| | | |
|---|---|---|
| wiegen | → | wXg-I |
| wägen | → | wXg-ä |
| wagen | → | wXg-a |

Putting these forms into a lexicon tree we find that the three verbs differ only in the last position.

```
(* (\w (\X (\g (\- (\I (\+ ((group 2)))) ;"ie" too is
                                        ;encoded as "I"
                 (\a (\+ ((group 5))))
                 (\a (\+ ((group 4)))))))))
```

The analysis is simplified. Immediately after preprocessing the the form we can reduce the possible candidates for the related infinitive to the subtree below the hyphen. This special encoding has the side effect that the number of nodes of the lexicon tree is reduced when many similar forms are added to the lexicon.

## 9 Constraints On The Lexicon

Three other classes of verbs have to be considered, if we want to find the stem of any German verb easily:

1. Verbs which change the stem at places other than the stem vowel.

2. Verbs with an infinitive ending on "-ern" or "-eln". These verbs omit in some cases the "e" which belongs to the stem (!).

3. Verbs with the ending "-ssen" or "-ßen". For these verbs the "ss" and "ß" have to be exchanged in some forms.

For (1) all the changed stems are added to the lexicon together with the grammatical information, that restricts their use to the permissible forms, which results in about 75 new entries for the lexicon.

The verbs in (2) and (3) are encoded in a special way. The encoding has no side effects on the rest of the algorithm. It only adds some transitions to the automaton (cf. /Paulus 1986/ [9]).

## 10 Further Extensions

The algorithm as implemented can handle all cases of prefixed verbs, even the cases where the prefix is separated from the verb for some forms (e.g. "er kam an").

The prefixes are added to the lexical information of the infinitive form. Thus an extra prefix requires only little extra storage for the lexicon. The analysis-mode checks whether the prefix is allowable or not.

Finally the algorithm also takes care of the transitive and intransitive use of a verb, if this affects the way the verb is inflected (e.g. "er schrak", "er erschreckte mich").

## 11 Practical Experience

The complete system for analysis and generation including all of the mentioned extensions has been implemented in TLC-LISP on a PC. The lexicon contains all irregular and strong verbs with their prefixes, and many other verbs, without running into memory limitations.

In a first try the German lexicon was built in a straightforward way (as shown in example 2) and all the inflection was done using rewriting-rules only. Comparison with the extended algorithm showed a runtime improvement of more than 75 percent. In absolute figures the performance of analysis is less than 1 second per verb form; the present version of the program consists of non-optimized compiled LISP code. French and Spanish verbs can be handled directly by the kernel algorithm without the described extensions.

## 12 Conclusion

A general, language independent FST-based model for morphological analysis and generation has been implemented and applied to the full range of German verb morphology. In the course of our investigations, we found that the treatment of particular language dependent inflectional phenomena which cannot be handled by the general model, can be easily embedded in a way which does not require to modify the basic model, but can instead be wrapped around it. Hence, problems which might come up in localizing the stem vowel by means of rewriting rules alone do not occur. From a general point of view, the main innovations in our system are a new method for word stem recognition and a generalized framework for lexical representation.

## References


[1] Gazdar, G.: *Finite State Morphology. A Review of Koskenniemi (1983).* Center for the Study of Language and Information, Stanford University, Report No. CSLI-85-32, Stanford, Cal., 1985.

[2] Johnson, C.D.: *On the Formal Properties of Phonological Rules.* PhD Dissertation, University of California, Santa Barbara. *POLA Report* 11, University of California, Berkeley, 1970. (Published as *Formal Aspects: A Phonological Description.* The Hague: Mouton, 1972)

[3] Karttunen, L., Koskenniemi, K., Kaplan, R.M.: *A Compiler for Two-level Phonological Rules.* Technical Report, Xerox Palo Alto Research Center and Center for the Study of Language and Information, Stanford University, Stanford, Cal., 1987

[4] Kay, M.: *When Meta-Rules are not Meta-Rules.* In: Sparck Jones, K., Wilks, Y.: *Automatic Natural Language Parsing.* Chichester: Ellis Horwood, 1983, 94–116

[5] Koskenniemi, K.: *Two-level Morphology: A General Computational Model for Word-form Recognition and Production.* University of Helsinki, Department of General Linguistics, Publication 11, 1983

[6] Koskenniemi, K.: *Two-level Model for Morphological Analysis.* In: Proc. IJCAI-83, 1983, 683–685

[7] Koskenniemi, K.: *A General Computational Model for Word-form Recognition and Production.* In: Proc. COLING-84, 1984, 178–181

[8] Koskenniemi K.: *Compilation of Automata From Two-level Rules.* Paper presented to the CSLI Workshop on Finite State Morphology, Stanford, July 29–30, 1985

[9] Paulus, D.: *Ein Programmpaket zur Morphologischen Analyse.* Universität Erlangen-Nürnberg, RRZE, Diplomarbeit, RRZE-IAB-259, 1986.

[10] Paulus, D.: *Endliche Automaten zur Verbflexion und ein spezielles deutsches Verblexikon.* In: Morik, K. (Ed.): *GWAI-87 — German Workshop on Artificial Intelligence. Proceedings* Berlin: Springer (IFB 152), 1987, 340–344