Kimmo Koskenniemi
Research Unit for Computational Linguistics
University of Helsinki, Hallituskatu 11
SF-00100 Helsinki, Finland

COMPILATION OF AUTOMATA FROM MORPHOLOGICAL TWO-LEVEL RULES

1. INTRODUCTION

The two-level model is a framework for describing word inflection. The model consists of a lexicon system and a formalism for two-level rules. The lexicon system defines all possible lexical representations of word-forms whereas the rules express the permitted relations between lexical and surface representations. Word recognition is thus reduced into the question of finding a permissible lexical representation which is in a proper relation to the surface form. Similarly, generation is the inverse where the lexical representation is known and the task is to find a surface representation which is in a proper relation to it.

Within the two-level model these relations pertaining to the rule component have been expressed in two ways. A rule formalism has been used for communicating the idea of the rules, whereas the actual implementations have been accomplished by hand-coding the rules as finite state automata. The close connection between rules and finite state machines has facilitated this hand-coding.

Expressing rules as numbers in a transition matrix is, of course, not optimal. Although it has proven to be feasible, it is tedious. It also tends to distract the linguist's thoughts from morphophonological variations to technical matters. Furthermore, hand compiled automata are often not quite consistent with intended rules. Discrepancies arise because the design of rule automata is often affected by assumptions

Compilation of automata from morphological two-level rules
Kimmo Koskenniemi
Proceedings of NODALIDA 1985, pages 143-149

on the regularity of actual word-forms. Thus, such automata usually function correctly with most of the data but they have a less clear relation with original intended rules.

The rule compiler described below rectifies this problem by letting the linguist write rules in a true rule formalism while the computer produces the automata mechanically. These can then be used in conventional two-level programs, both for testing and and for production use. Several two-level descriptions are now on their way towards completion using the compiler.

2. THE FORMALISM OF TWO-LEVEL RULES

The actual rule formalism supported by the two-level compiler differs only slightly from the original formalism proposed in Koskenniemi (1983). One of the differences is the use of linear representation for pairs, thus **a:o** is used instead of $\frac{a}{o}$. Furthermore, the equal sign is no longer used for denoting the full lexical or surface alphabet. A surface vowel is written simply as **:V** and a lexical **a** as **a:**. Other basic elements are as they used to be:

(1) A sequence of elements are written one after another, thus **:V :V** stands for two successive surface vowels.

(2) Alternative elements are separated by a vertical bar and enclosed in square brackets, e.g. **[:i | :j]** stands for either a surface **i** or a surface **j**.

(3) Iteration is indicated with a superscript asterisk or plus sign, e.g. **:C*** stands for zero or more surface consonants whereas **:C+** requires at least one surface consonant.

Rules with operators => <= and <=> exist as before and they are interpreted as before. A rule:

I:j => :V __ :V

states that every occurrence of a pair **I:j** must be in a context of **.. :V __ :V ..** i.e. between two surface vowels. A rule:

I:j <= :V __ :V

states that between surface vowels a lexical **I** has no other possible realizations than a **j**. Rules with operators <=> are combinations of those with <= and =>.

There is one new type of rules with an operator **~<=**. This rule forbids any occurrences of **LC CP RC**, i.e. it forbids **CP** in the context **LC __ RC** .

Another difference is in the formalism for collapsing several similar rules into one rule. The initial formalism used angle brackets, but this has been replaced by equivalent means using so called "where" clauses. If W denotes a morphophoneme for vowel doubling then a rule for vowel doubling is expressed in the present formalism as:

W:x <=> :x __ ; where x in V;

The definition of the rule component of two-level descriptions consists of six sections:

- a surface alphabet as a list of surface characters
- subsets of the surface alphabet which are used in the rules
- a lexical alphabet
- subsets of the lexical alphabet
- definitions for abbreviations or subexpressions used in the rules
- two-level rules.

A sample two-level description is given below:

```
Lexical Alphabet
  a b c d e f g h i j k l m n o p q r s t u v w
  x y z å ä ö al a2 E I = W;
Lexical Sets
  V = a e i o u y å ä ö al a2 E I W;
  C = b c d f g h j k l m n p q r s t v w x z;
  Diacritics = / ^;
Surface Alphabet
  a b c d e f g h i j k l m n o p q r s t u v w
  x y x å ä ö;
Surface Sets
  V = a e i o u y å ä ö;
  C = b c d f g h j k l m n p q r s t v w x z;
Definitions
  Defaults = al:a a2:a E:e I:i;
Rules
  "Vowel doubling"        W:X  =>   :X __;
                          where X in V;
  "Suppressed doubling"   W:0  <=>  __ I:;
                                    I: __;
                                    W:V __;
  "Stem final V"          [al:0 | a2:o | E:0 | i:e]
                               <=>  __ I:;
  "Plural I"              I:j  <=>  :V __ :V;
```

Note that surface and lexical alphabets are declared separately. This guarantees that the role of each segment is uniquely determined. The sets are separate also because it is not always evident e.g. which segments are considered to be vowels on the lexical level.

The first rule represents a set of several rules:

```
W:a  =>  :a __;
W:e  =>  :e __;
...
W:ö  =>  :ö __;
```

The "where" clause is interpreted in this way because the dummy variable **X** occurs on both sides of the rule. If it would

occur only on the context side then the abbreviation denotes one single rule with many context parts (one for each possibility of **X**).

Another effect of the expansion of "where" clauses is the introduction of some new character pairs. Pairs **W:a**, **W:e**, ..., **W:ö** do not occur anywhere in the description but they are implicitly included.

3. STEPS OF THE COMPILATION

The compilation of the two-level description into finite state automata proceeds in several steps. The computation relies essentially on Ron Kaplan's program packages (FSM, FST) for manipulating finite state machines and transducers. The collection of rules has to be treated as a whole because the set of character pairs (CPS) might be changed if some rules are altered thus changing the interpretation of some other rules. The steps of the compilation are:

(1) Transformation of the two-level rule description into a recursive list expression where rule components and pairs are identified.
(2) Expansion of "where" clauses in the rules.
(3) Collecting all pairs explicitly mentioned in rules and definitions in addition to the default set of all **x:x** where **x** is both a lexical and a surface character.
(4) Computing the exact interpretation of pairs which are not concrete pairs (where both the lexical and the lexical components are single characters). Some pairs like **:V** leave one level fully open and others may use the defined subsets such as **W:V** (character **W** corresponding to any of the vowels but not to a zero **0**). All such (partially) unspecified pairs **X:Y** denote the subset of CPS consisting of pairs **x:y** where **x** is **X** or is in **X** and **y** is **Y** or is in **Y**.
(5) Expand each abbreviation of the above type into an alternation. Insert the defined expression in place of the name of the expression.

(6) Compile the components of the rules (correspondence parts, left contexts and right contexts) into finite state machines.
(7) Split rules with the operator <=> into one rule with operator => and another with <=.
(8) Expand rules with operator <= and with multiple contexts into distinct rules with one context each.
(9) Compile the individual component rules separately.
(10) Merge the automata resulting from a single original rule into one rule automaton by intersecting them.

In the present version of the compiler each rule as defined by the user is compiled into a single automaton. If the expansion or compilation splits the rule into subparts these are finally combined into a single machine by the compiler.

The compilation is done on a Xerox 1108 Lisp machine with programs written in Interlisp-D. The resulting automata can then be used either on the Lisp Machine or transported to other systems. In order to be used by the present version of the Pascal two-level program the automata are converted into a tabular format which can be readily used. The format is slightly different from the original one given in Koskenniemi (1983) but it is significantly faster to read in. Such automata have been successfully used on MS-DOS micro computers such as IBM PC and Olivetti M24. Martti Nyman at the University of Helsinki working on one description for Modern Greek and another for Classical Greek and Jorma Luutonen at the University of Turku is working on one for Cheremis. Olli Blåberg has reformulated his Swedish description in terms of the present compiler.

The compiler was written during the summer 1985 at the Center for Studies on Language and Information at Stanford University. In addition to the finite state package written by Ron Kaplan the compiler utilizes Kaplan's concept of compiling complex rules with operator => and several context parts. The compiler was presented at a symposium on finite state morphology on July, 29-30 1985 at CSLI. The compiler has also stimulated some parallel efforts (Bear, in press, Ritchie et. al. 1985, Kinnunen, in preparation).

REFERENCES

Bear, John, (in press). A morphological recognizer with syntactic and phonological rules. Proceedings of COLING-86, Bonn.

Kinnunen, Maarit, (in preparation). Morfologisten sääntöjen kääntäminen äärellisiksi automaateiksi. (The compilation of morphological rules into finite state automata) Masters thesis, Department of Computer Science, University of Helsinki.

Koskenniemi, K. 1983. Two-level morphology: A general computational model for word-form recognition and production. Publications, No. 11, University of Helsinki, Department of General Linguistics. Helsinki.

---, 1984. A general computational model for word-form recognition and production. Proceedings of COLING-84. pp. 178-181.

Ritchie, G.D., S.G. Pulman, and G.J. Russel, 1985. Dictionary and morphological analyzer (Prototype), User guide: Version 1.12. Department of Artificial Intelligence, University of Edinburgh.