A Two-level Morphological Processor

Lynne J. Cahill

August 1987

ognitive Studies Research Paper
erial No: 94
he University of Sussex
chool of Cognitive Science
almer

A Two-level Morphological Processor

Lynne J. Cahill

ABSTRACT

A system for the analysis and production of word forms is described. The system is based on the two-level model of Kimmo Koskenniemi, with the major changes being to his lexicon system, to allow greater linguistic generalizations. The lexicon system makes use of a rule system for describing inflectional morphology proposed by Arnold Zwicky. It incorporates aspects of both models to give a system which can analyse word forms to give their morphological features, or produce correctly inflected word forms, when given the stem and features.

Introduction

Natural language processing (NIP) systems have frequently, in the
neglected the morphological aspects of language, partly because o:
preoccupation with English, vfoich has a minimal inflectional mor]
and partly because of the predominance of syntax in linguistic
The increase in work on other languages in NIP, and in theoretical
on morphology has signalled an increase in interest in cotputatioi
counts of morphology.

One of the most widely known systems for the analysis and synthe
word forms is that of Kimmo Kbskenniemi, and his system forms th<
for the model presented here* While his rule system provides a li]
ically sound, as well as cranputationally effective means for handl:
morphonology of a language, the structure of his lexicon sysi
only irotivated by the data he uses, i.e. Finnish, and not by g<
cross-linguistic phenonena. It is this aspect of Kbskenniemi^f^s
which is radically changed in the model presented here, to provide
icon system, and, thereby, a means of handling the morphotactics
language, which reflects linguistic facts and makes claims about •
ture of language in a way in vfriich Kbskenniemi's system does not.

The alterations to Kbskenniemi^f^s lexicon system are based largely
the means for describing inflectional morphology proposed by
Zwicky in his paper, ^lf^Hcw to describe inflection" (1985). In this
Zwicky proposes a rule system which provides an elegant and natu

proposes, however, was not intended as a cxxrputational model* The model presented here, therefore, provides a ccnputational inplementation of the theory behind the rule system.

In the first section, we shall begin by looking briefly at Kbskenniemi's system, followed by a more detailed exposition of the rule system proposed by Zwicky. The second section will then present the aartibined model, with an explanation of hew the rule system was implemented, and how it was integrated into the Kbskeraiiemi-type system.

Finally, we shall consider the merits and the inadequacies of the system, together with sane suggestions for ways in which it could be extended and improved. Work is continuing on the system, and the model presented here represents its state in August 1986.

Section 1: The models of Koskenniemi and Zwicky

Koskenniemi's two-level model was presented as his doctoral thesis
University of Helsinki (1983). Although Koskenniemi himself de
the model as one of morphology, he uses the two-level rules, whi
the essence of the whole theory, as phonological or morphonologica
and it is interesting to note that in a later publication (with Ka
and Kaplan, 1987) he refers to the same rules as being phonologica
two-level rules take the form,

$$\begin{array}{cccccc} i & & = & + & & = \\ & \Longleftrightarrow & & & \text{——} & \\ j & & V & 0 & & V \end{array}$$

where the top row refers to the lexical representation, the bott
refers to the surface representation, and the rule describes a cor
dence between the lexical "i" and the surface "j", in the given c
i.e. between two surface vowels where there is a plural mar
present. The rules can refer to correspondences which are conditi
the presence of phonological features (e.g. the vowels) or by th
ence of morphological features (e.g. the "+") or , as in this case
combination of the two.

The rules in Koskenniemi's system are given to the system in the f
finite state automata, which are all applied in parallel. Koskenn
at pains to stress that the rules represent correspondences a
processes, and hence that they are entirely bi-directional, a

lexicons together with continuation classes. Each continuation class is defined simply as a set of sublexicons, an entry in any one of which may come after any lexical entry which has that continuation class stored with it in the lexicon. For example, a lexical stem may have the continuation class S123 stored with it, which together with the definition of that class in the lexicon (S123 = S1, S2, S3), ensures that that stem is only followed by an entry from one of those sub-lexicons (S1, S2, S3). This type of lexicon enables Koskenniemi to describe the inflectional patterns of Finnish, but, as we shall see in Section 3, we come upon problems when we look at the German system of inflection for determiners, adjectives and nouns, which is what Zwicky bases the demonstration of his rule system on.

It must be stressed from the start that the aim of Zwicky's paper, "How to Describe Inflection" is only to "sketch a framework for describing systems of inflectional morphology". Zwicky says in a footnote, "My aim in formalization in this paper is clarity, not completeness or any envisaged computational implementation. Nor do I intend the framework as an incipient processing model." He does concede, however, that the framework lends itself fairly naturally to some such enterprises.

The rule system proposed by Zwicky consists of two types of rule, rules of exponence and rules of referral. A rule of exponence describes how particular features are realised in certain contexts. For example, "in English, in the context of [CAT: Verb], [VFORM: Past] is realized by the

The rules are assumed to express defaults. That is, the more s
rule overrides the more general one. What this means in terms of
tual set of rules he gives, is quite easy to work out, as can be s
Section 3, but only in terms of the specific features mentioned
rules. There is no general principle, such as the number of featu
stantiated, which could apply to any rules. The decision is l
specific.

Another important aspect of the rule formalism is its ability to r
value-clusters and feature-clusters. A value-cluster is a g
values which a particular feature can take, such as direct case,
ring to nominative and accusative case, as opposed to genitive and
case, which can be called oblique case. A feature-cluster is a gr
features which may have a combined realization rule, such as the
CASE/GEND/NUM in German.

A feature-cluster is related to the notion of slots which Zwick
mentions. He suggests that in each language there is an ordered
(abstract) slots for inflectional material. What this means is th
each category or sub-category, there are certain generalization
can be made about affixation (although it must be noted, and will
cussed in more detail later, that affixation is not the only kind
flection which we want to be able to model). Zwicky defines sl
saying, "Any particular rule supplies material for a specified
slots, and several distinct rules can supply material to the same
The ordering of a rule with respect to others is then governed

ordering of slots". An example of what a slot is can be given from English, where nouns have slots for plural marker and possessive marker, for example, "cats'" can be divided "cat + s + 's", where the first "s" is the plural marker, the " 's" is the possessive marker, and the final "s" has been omitted for phonological reasons. In any case, it can be seen that the slot for the plural marker is positioned before the slot fro the possessive marker.

As mentioned before, the rules are assumed to express defaults, so a rule which expresses an exception, applying only to, say, one case, is going to be ordered before any rule which expresses a generalization. Zwicky demonstrates his rule system with a set of rules for the German declensional forms of determiners, adjectives and nouns. He specifies aliases for category groupings,

Adjal = {adjective, determiner}

Nounal = {adjective, determiner, noun}

where a rule with the first grouping will necessarily apply before a rule with the second, it being more specific. He also specifies the value-clusters mentioned before,

Direct = {nominative, accusative}

Oblique = {genitive, dative}

Zwicky's convention of using capital letters to distinguish aliases from straightforward values will be used in this chapter.

Zwicky also mentions, although not in great detail, VCRs - value couiar value, in zwicKy's interpretation, a VGR taxes trie form, if

tain feature has a particular value then a certain realization ruL

not apply. Thus, Zwicky does not really use his VCRs in the same

the FCRs, since his VCRs do not say that particular features or va

features nay not be present, merely that the rule which associa*

feature or value with a morpho- or phonological realization shoi

overlooked. This subtle distinction explains Why Zwicky's VCRs a

pily fit in the lexicon, since they refer to rules which belong

lexicon, while Gazdar and Pullum's FCRs belong in the syntax, wh<

feature value assignment is defined. Zwicky gives one VCR, wh

shall look at later, when we have defined the other rules.

The declension systems for German adjectives ard determiners are g

ly divided into three classes, strong, weak and mixed. Weak decle

endings occur on adjectives which follc*/ definite articles (der

das etc.); strong endings occur on determiners and adjectives w

not follow a determiner; and mixed endings (a mixture of strong an

occur on adjectives which follow an indefinite article (ein, mei

etc.). Mixed endings can be defined in terms of the other two c

as we shall see shortly.

Ihe set of weak endings can be seen in Table I, over.

As can be seen, all plural forms, and all Oblique forms take the e

-en , while all Direct singular forms, with the exception of the

tive masculine, take the ending -e . This is generally cumber

describe, but with Zwicky's system, the set of endings, including

	MASC-SG	NEUT-SG	FEM-SG	PLURAL
NOM	-e	-e	-e	-en
ACC	-en	-e	-e	-en
GEN	-en	-en	-en	-en
DAT	-en	-en	-en	-en

Table I.

exception is accounted for, with the following three rules:

i. In the context of [CAT:adj, CLASS:wk], [CASE:acc, GEND:masc, NUM:sg] is realized by the suffixation of /en/.

2. In the context of [CAT:adj, CLASS:wk], [CASE:Direct, NUM:sg] is realized by the suffixation of /e/.

3. In the context of [CAT:adj, CLASS:wk], any bundle of CASE, GEND and NUM values is realized by the suffixation of /en/.

The first rule describes the single exception, and overrides the second, so we are still able to make the generalization. The third rule then describes the general default, or "elsewhere application".

The strong endings are shown in Table II, over.

The groupings here are obviously nowhere near as simple to describe as with the weak endings. To start, though, we note that the accusative

1. In the context of [CAT:Adjal], [CASE:acc, GEND:masc, NUM:sg] is realized by the suffixation of /en/.

	MASC-SG	NEUT-SG	FEM-SG	PLURAL
NOM	-er	-es	-e	-e
ACC	-en	-es	-e	-e
GEN	-en	-en	-er	-er
DAT	-em	-em	-er	-en

Table II.

The nominative masculine singular is also a case on its own:

4. In the context of [CAT:Adjal], [CASE:nom, GEND:masc, NUM:sg] is realized by the suffixation of /er/. These rules need to mention the class, since in cases of weak class the more s rules will apply, and the cat is Adjal, since this also applies to miners.

Zwicky gives the rules for describing the rest of the strong endi follows:

ii. In the context of [CAT:Adjal], [CASE:acc] has the same realization as [CASE:nom].

5. In the context of [CAT:Adjal], [CASE:nom, GEND:neut, NUM: realized by the suffixation of /es/.

realization as [GEND:neut].

8. In the context of [CAT:Adjal], [CASE:gen, GEND:neut, NUM:sg] is realized by the suffixation of /en/.

9. In the context of [CAT:Adjal], [CASE:dat, GEND:neut, NUM:sg] is realized by the suffixation of /em/.

10. In the context of [CAT:Adjal], [CASE:Obl, GEND:fem, NUM:sg] is realized by the suffixation of /er/.

iii. In the context of [CAT:Adjal], [CASE:dat, NUM:pl] is realized by the suffixation of /en/.

12. In the context of [CAT:Adjal], [NUM:pl] has the same realization as [GEND:fem, NUM:sg].

In fact, it would appear that rule number ii is actually unnecessary, since all it does is refer the accusative case to the nominative, but he has already provided us with a means of referring to the nominative and accusative cases together, without having to define a separate rule to do so. I therefore propose that this rule should be eliminated and all occurrences of nom in the above rules replaced with Dir.

The other rules are fairly straightforward, referring the masculine to the neuter and the plural to the feminine singular, with the exception of the nom and acc masculine, which were defined before, and the dative plural, which has a separate rule. The rules of exponence define the actual realizations for these.

13. In the context of [CAT:det], [CASE:gen, GEND:neut, NUM:sg

realized by the suffixation of /es/.

The mixed declension of adjectives and determiners, as mentioned b

can be defined in terms of the others:

14. In the context of [CAT:Adjal, CLASS:mixed], [CASE:Dir, NU

has the same realization as [CLASS:str].

15. In the context of [CAT:Adjal, CLASS:mixed], any bundle of

GEND and NUM values has the same realization as [CLASS:wk].

So far, we have not mentioned the nouns. German nouns have two

one for plural marker and one for the CASE/GEND/NUM marker. For ex

the word <u>Buch</u> (book) can be made plural by the suffixation of -e

gether with an umlaut, giving <u>Bücher</u> and it can also carry a suffix

ciated with the CASE/GEND/NUM slot, dealt with for adjectives and

miners above. In fact, there is only one possible suffix - the -e

suffix representing the dative plural (any gender), which, after th

nological adjustment gives us <u>Büchern</u> .This shows another feat

slots - they do not have to be mutually exclusive regarding the fe

they represent. It can also be seen in this case, that the NUM s

ordered before the CASE/GEND/NUM slot. Zwicky's rules are fo

CASE/GEND/NUM slot only, so we shall ignore the fact, for the time

that the NUM slot has several different possible realizations (al

we will look at this in the combined model).

-en for the dative plural. These few instances of suf fixation can be easily accounted for by the rules:

17. In the context of [CM?:noun], [CASE:gen, GEND:neut, NUM:sg] is realized by the suf fixation of /es/.

18. In the context of [CAT:noun], [CASE:gen, GEND:masc, NLM:sg] is realized by the suf fixation of /s/.

The dative plural suffix can be handled by incorporating the category "noun" into rule number iii above, changing it to:

11. In the context of [CaJT:Nounal], [NUM:pl] has the same realization as [GEND:fem, NUM:sg].

There is also a class of nouns, known as weak nouns, which have almost the same declension as weak adjectives, for which Zwicky gives the following rule:

v. In the context of [CAT:noun, CLASS:wk], any bundle of CASE, GEND and NUM has the same realization as [CAT:adj].

together with the VCR we mentioned earlier, which takes the form:

vi. If a bundle contains CAT:ncun and CLASS:wk, it also contains KJI£|(2):no.

and means that rule 2 does not apply in that situation. However, this does not seem to tell the true story about weak nouns, since their declensional pattern, according to Hammer (1971), is an -en suffix in ALL cases except the nominative singular. This could be expressed by the rules:

vii. In the context of [CAT:noun, dASS:wk], any bundle of CASE,

is a result of the non-application of rules, so that if no suff
rules apply, then the bare base is what results. He does say "I c
reject the possibility that seme zero formations are stipulated b
I am, however, assuming that the normal source for zero formations
absence of any rule providing an exponent for certain bundles.[11]
not clear whether the above case is one which might have a zero f oi
stipulated by rule, intuitively. It would, however, make generali
much easier and make the rules for this particular aspect of
correspond more to the other rules described so far:

18. In the context of [GW?:noun, CLASS:wk], [CASE:ncm, NUM:s
realized by the suffixation of /-/•

19. In the context of [CAT:noun, CIASS:wk], any bundle of G
GEND anfl NUM values is realized by the suf fixation of /en/.

Hiese then are all the rules needed for the declension of German
determiners and adjectives. The full list of those given here, w
the final adjustments, is given in appendix A.

Another point which must be made about Zwicky's rule system is th
lating to directionality. As we saw in Chapter 1, Zwicky states t
component follows a syntactic component and precedes a phonologica
implying that the rules apply in the production of word forms only
is also the iirpsession given by the wording of the rules. Howeve
important aspect of the rules is that they can be seen to d<
correspondences between the realization (in this case suffixes) ai
set of features, and they can therefore, when viewed as such, be \

either direction.

The next section will now present the combined model, demonstrating ho
Zwicky's rule system can be integrated into the two-level model o
Koskenniemi.

Section 3: The Combined Model demonstrated using German

As stated before, the model being presented here uses a combinat
the model of Koskenniemi and the rules system of Zwicky. The mode
sists of five major modules, together with a "lexbuild" module,
constructs the lexicon initially. The other five modules are: 1.
bet, 2. Tables, 3. Fst (Finite State Transducer), 4. Lexicon, 5.
cess, which this section will describe in turn. We shall look a
module in respect of what it does, how it does it, and how it c
from Koskenniemi's model.

1. Alphabet:- The alphabet module, like Koskenniemi's, contains th
phabet, defining the possible surface characters and the subse
aliases, for example, the set of vowels. It also defines subsets c
morphological features, such as,

    alias(case,[nom, acc, gen, dat])

which defines the case features, and

    alias(dir,[nom, acc])

which defines a subset of these, the direct case, which was used i
last section. The morphological features are all self-explanatory
nom and acc, unlike the symbols used in Koskenniemi's model, "+
plural and so on. There is no need to use cryptic symbols, as mu
charactered objects are treated exactly the same as single characte
it seems more sensible to use names for the features which have an
ous meaning. This makes understanding the program much easier, and

terpretation of a word in a form something like, "ein + nom + pl".

As well as the aliases, the alphabet module in this model also contains definitions and procedures for checking character pair sets. In Koskenniemi's model, as we saw, the character pair sets were found by the finite state transducer module. This could also be done in this model, by the definition of a procedure to do it in the process module, but for the time being the concrete pair sets are simply defined using very simple Prolog rules, which allow every character to be paired with itself, all morphological features to be paired with a 0 on the surface, and all the other possible permutations allowed by the two-level rules. It also contains the procedures to insert a 0 in either level if there is a possibility of such a correspondence. For example, suppose there is a rule which involves the correspondence of a lexical "e" with a surface 0 (which there is in German), and we are analysing a surface form into its lexical representation, at each point we have to consider the possibility that there is a lexical "e" which has no surface realization. This procedure makes the overall program consider the possibility. (Of course, it only considers the possibility in the light of the morphonological and lexical rules.)

The list of aliases together with the procedures for checking the character pairs can be found in Appendix B.

2. Tables:- The tables module contains the tables which represent the two-level rules. There are only four tables in this model for German,

ready been demonstrated amply with Finnish, English, Rumania
Japanese. It is not claimed that the two-level rules given here a
haustive for the German language, but they give a flavour of what
ing on.

The first two-level rule modelled here, and probably the most c
used, is that for umlaut on certain plural forms. Nouns in Ger
six different means of forming the plural, and are therefore divid
six groups, marked in this model by a number feature, e.g. <u>der</u> <u>Ta</u>
belongs to group 3 which adds an -<u>e</u> to form the plural, die Tage,
would have in its lexical entry something like, [n, masc, 3], to s
it is a masculine noun of group three.

It has been suggested by some grammarians that the plural group to
a noun belongs can be determined by the gender, with some exceptio
the number of exceptions makes it more efficient for our purposes
store the plural group with each noun. However, there is so
linguistically desirable about generalizations of that form, sinc
example, the vast majority of feminine nouns take an (<u>e</u>)<u>n</u> suff
any German speaker who did not know the plural group of a feminin
would guess at this suffix, so the best situation would appear t
have the plural groups of those that are known listed directly,
also have, in the lexicon, rules to determine the likely plural g
any nouns for which it is not known. For example, a rule like Z
such as,

such as the number of syllables in the stem or the final segment of the stem, the suffix given could be in the form of a variable which is bound using the phonological rules, for example,

```
      Suff           e   l   neut   pl
           <==>
       0             e   l    0      0
```

determines that a neuter noun ending in -<u>el</u> has no surface realization of the suffix in the plural. For the current model, however, we shall use only the listed plural groups.

The first two-level rule, then, is that which matches a lexical back vowel with a surface front vowel, or "umlauted", and the rule can be written:

```
      bl                    ug   pl
          <==>   -----
      ul                    0    0
```

where bl is an alias for back vowels, ul is an alias for umlauted vowels, and ug is an alias for those plural groups which take an umlaut. It therefore says that a back vowel may correspond to an umlauted vowel if it is followed by a plural group which takes an umlaut, and a plural marker (all morphological features come after the lexical item, so the morphological features associated with the stem appear after the stem itself, and those associated with the suffix come after it, and so on). As it stands, this rule permits the correspondence of back vowels in the lexicon with surface umlauted vowels which do not match, e.g. a/ö, but this is taken care of by the alphabet module which ensures that there are only permitted correspondences. The table to represent this rule needs the alphabet:

ul , 0 0 0 =

that is, all the peiirs mentioned in the rule, the sg/O pairing to
trast with plural, and the "anything else" pair. The table then h

	bl Ul	ug 0	pi 0	sg 0	= =
1:	2	1	1	1	1
2.	0	3	0	0	2
3.	0	0	1	0	3

which moves to state 2 only if it encounters a bl/ul pairing and tl
can only return to the final state 1 if it encounters first a ug i
and then a plural marker* It will always encounter them in this
as the plural group marker is stored with the stem, while the
marker cones as a result of the suffix. Anything else just keeps 1
tomaton in the state it is currently in.

The other rules are to delete a surface "e" in an -en suffix if :
lows an "e", an "er" or an "ar", for example, Bauer/Bauern rathei
Baueren ; to add a surface "n" in front of -en plural suffi:
feminine nouns, for example, LehreriyifihrereriiTnen rather than]
inen ; and to delete a final -um on stems when adding a plura:
suffix, for example, A]Jjurq/Alben rather than Alburry/Albumen .

Apart from the umlaut rule, the others are all phonological or si
speaking, orthographical, and it should be mentioned that phono!

rules equally as effectively as those that are phonologically conditioned, with the distinction that the former depend on the values of features on the top row, vfoile the latter depend on the values of surface characters on the bottom rcw.

The rules mentioned above can be found with their tables in Appendix C.

2* Fst:- The Finite State Transducer is the part which most closely resembles Kbsk^nniemi's model. For each input pair it moves every automaton to the next state, according to the appropriate matching pair on the automaton labels. Kbskenrdemi's model does not interpret the labels, but uses an expanded version of the tables with every possible pairing explicitly listed. This model, however, has procedures to check for applicability of labels. It therefore has to check for each input pair and each table, that there is a label which will apply Owhich there always is because of the =/= label in each table), and that it is the most specific that can apply. This is not a simple procedure, since it involves several different checks. For example, if an *input* pair apparently only fits the =/= label, the procedure naast check that there is no label which exactly matches the input pair, then that there is no label vtfiich has one component which exactly matches one component of the input pair and whose other component is applicable to the other component of the input pair (this check has to be done for both ocaonponents), and finally that there is no other label with aliases which both fit the input pair, and vfriich are more specific than the =/= pair.

finds it is at the end of the input string, it calls the Fst moc
check that the automata are all in a final state.

Ihe procedures in the Fst module are listed in Appendix D.

£. Lexicon:- Ihe lexicon system has three main parts: i. the It
building module, ii. the lexicon checking procedures, and iii. th*
con itself. Ihe first two of these are general procedures for the
tern, while the third is the language specific data module. Let i
at the three in turn.

i. The lexicon building procedures are similar to Kbskenniemi's.
take a list of lexical entries and the information to be stored wi1
and build labelled letter trees, as described in Kbskenniemi i
Each lexicon, or sub-lexicon, has a name, and a list of structurec
which represent the letter trees. Additional information can be ii
into the lexicon at the appropriate point after the lexical ent
volved has been added.

Ihe lexicon building procedures are listed in Appendix E.i.

ii. The lexicon checking procedures are basically similar to the]
building procedures, but instead of inserting information, they <
information, or fail if the entry is not there, or has no info]
with it. Ihey do not just check to see if a word or word part is \
in the lexicon, but also return any information vfriich is stored wi1
If a word has separate interpretations, say as a noun or verb, 1
sets of information are listed separately, and the checking prc

will return one set of information at a time.

Ihe code for this module is given in Appendix E.ii.

iii. The actual lexicon module, the data module vfaere the lexical information for the language in question is stored, is the area where the Zwicky rules, as described in the previous chapter, are incorporated. It is this part of the model which most differs from Kbskenniemi^{f}s. In Kbskenniemi's model, the lexicon itself contains only the definition of continuation classes and the sub-lexicons. In the lexicon here, we need to define not what the continuation classes given with each entry mean, but the rules for the continuation classes. These rules take the name of the current lexicon, and the information currently knewn about the word being analysed or produced, and return the sublexicons from which the next part of the word may cone.

In many cases this is very strai^itforward. For example, in English, if we are in the stem lexicon and we knew that we have a noun, then we can go to the plural suffix lexicon or the possessive suffix lexicon next. Kbskenniemi admits that his lexicon system is limited in this aspect. He says that his system "seems powerful enough to cover the morphotactic structure of many languages. Only a small residue of structures ... forces one to resort to the use of rules and features for morphotactics." (p.27) Ihe small section of German being described here, and vfriich is described by Zwicky, however# does fall into this category, and cannot be handled by using simple continuation linkages. The set of endings which

inherits from its context in the overall structure, i.e. what, i
determiner it combines with in the noun phrase. Koskenniemi's sys
it stands could not, therefore, handle this - it would require at l
set of rules to interpret the suffixes in the light of the syntact
formation.

The Zwicky rules, as described in the last Chapter, are easily enco
Prolog rules, although the interpretation, as with the automata l
is not so simple. In fact, Zwicky's concept of which rule is
specific than the others is actually language specific, or to b
precise, depends on the particular features being used. In the s
rules for the noun/adj/det system of endings, the CAT and CLASS fe
are the most important, and the CASE, GEND and NUM features are les
portant in determining the most specific rule. Since there are n
eralizations that one can make within this rule system which would
to any set of rules, it was decided that, for the current purpos
would be just as efficient to use simple ordering of rules so that
can only apply if there is no other rule which could also apply, wh
ordered before it.

A rule of the form:

1. In the context of [CAT:Adjal], [CASE:acc, GEND:masc, NUM:sg] is realized by the suffixation of /en/.

is expressed in Prolog as,

```
cgntest(adjal,class,acc,masc,sg,suff2).
```

previous section, this rule can be used in Prolog to apply in either direction, to fill in either the features (or any subset of them) or the lexicon. For analysis of word forms, therefore, the rule would be applied with the suffix known, while in the production of word forms, the features would be known.

A rule of referral, like:

> 7. In the context of [CAT:Adjal], [GEND:masc, NUM:sg] has the same realization as [GEND:neut].

is expressed in Prolog as,

```
cgntest(adjal,class,case,masc,sg,Suff):-
cgntest(adjal,class,case,neut,sg,Suff).
```

where "Suff" is a variable which should be instantiated to whatever is in that position in the rule which satisfies the second clause. In fact, the rules are all numbered and marked with "e" or "r" to distinguish rules of exponence from those of referral, so it would actually look more like,

```
cgntest(r,15,adjal,class,case,masc,sg,Suff):-
    cgntest(e,X,adjal,class,case,neut,sg,Suff).
```

This appears very simple, but in fact there is more to it, as the rules as they stand do not know how to interpret the aliases. Again, we could just write out an expanded list of rules for every permutation for all the features, but this would mean an enormous increase in the number of rules, and would mean abandoning much of the theory behind the rule system. by abandoning the generalizations, which are not only convenient,

matches and membership of alias lists. It is not a complex pro
and nor is the procedure to check whether there is a more specif
which fits, since the latter simply involves checking the rule num
seeing if there is another rule which applies, and which has
number. Of course, the procedure must be called within the rules
ferral, to interpret the second clause.

Another complication which has been overlooked above, is that ru
referral actually need to be handled even more differently, as th
like the rules of exponence, can be applied in parallel. Thus, ea
of referral needs a second clause which says that it is satis
there is a rule of exponence with the relevant features (as given
or if there is another rule of referral which is also satisfie
those features specified by the first. For example, in the German
given above, there are two rules of referral which may both apply
er, numbers 7 and 14 in the list in Appendix A, which refer the ma
singular to the neuter singular and the mixed class of adjectives
direct singular to the strong class respectively. The first ru
should therefore refer to the other rule, 14, in the case of the
bundle, CAT:adj, CLASS:mixed, CASE:nom, GEND:masc, NUM:sg, givi
bundle CAT:adj, CLASS:str, CASE:nom, GEND:neut, NUM:sg, which mu
be matched to a rule of exponence.

It should be noted at this point, that Koskenniemi's interpretat
Finnish morphotactics could be easily fitted to this model.

The code and sub-lexicons described above are listed in Appendix E

That completes the description of the lexicon system. Let us now look at the final module, the process module, which brings together all the other modules described above.

5. Process:- The process module is the section which actually gets everything going. It takes the input, calls the other modules as they are needed, and returns the output. Unlike Koskenniemi's model (or most other models, for that matter) it does not have separate routines for analysing and synthesising word-forms.

To start, the process module finds all the tables, and creates a statelist, which is just a list of two-element lists, the name of each table (just a number) and its current state. It puts each table in state 1, the initial state. It then works through the input, first calling the alphabet module to check the character pair or find the possible alternatives if one element has to be found, then calling the Fst module to alter the statelist for each input pair, and calling the lexicon system to check membership and find lexical information. It does checking of tables and the lexicon in parallel, that is, it doesn't first find the possible phonological correspondences for the whole word, and then check the lexicon, but does it all as it goes along. This is very important with the system as it is at the moment, because it is required to do the processing in both directions using the same routine. If it only used one module at a time, it would be necessary to have different routines for analysis and production, to do each type of processing in different lyses the word kleines , which means "small" and is inflected nominative neuter singular of the weak adjective (amongst others program will find all possible analyses, but let us just look one). First, after it has set up the statelist, it checks the p character pair for the first letter given, and finds that there one possibility - k/k. It then moves each automaton to the next for this pair, that is, it changes the state on the statelist. N checks in all of the sub-lexicons which it knows can be initial le to see if there is a lexical entry "k". There isn't, so it co with the next letter pair in the same way, again the only possibl is l/l, and again there is no word part in the lexicon, "kl". It ues in this way until it gets to the "n", by which time the word p is looking for has grown to "klein", and it finds this in the ste con, together with the information that it is an adjective. If th part has not been found in the lexicon, then the "newlex" which it look in, is always the same one it was looking in anyway, but once been found, the lexicon rules have to be used to find which lex may go to from there. It therefore looks at the rules (from knowing only that it is an adjective. Since most of the rules a adjectives, there are several which can apply and it simply tries them in turn, until it finds one whose lexicon matches the en have, -es . It finds the rule,

```
cgntest(e,4,adjl,class,nom,neut,sg,suff3)
```

and assigns the value "weak" to the class, because no aliases are

```
        [k,l,e,i,n,adj,e,s,wk,nom,neut,sg]
```

This case did not use any of the morphophonological rules in the tables but an example of a run using the umlaut rule is shown in Appendix G.

The code for the process module is in Appendix F.

The system as described so far has certain limitations. The final section will discuss these and indicate how it is planned to remedy them.

Section 4: Concluding Remarks

The system as it stands still uses a number of sub-lexicons, Koskenniemi's model, most of which contain only one entry. This necessary and the system could be improved by having one lexicon co ing stems, together with "newlextest" type rules to provide the af However, if we do this, it becomes impractical to retain the co bi-directionality in the current system. If we want to give the sy stem and certain features for it to return the surface representati would be necessary to derive the lexical representation, with a fixes, before deriving the surface representation with all necessar nological alterations. If the two processes are not being carri simultaneously, then it is not practical to use the same routine fo duction and analysis of word forms. Although it could still be do would be extremely inefficient as the routine would necessarily be more to one direction of processing, and would be making guesses dark when doing processing in the other direction. For example, i routine began by checking for affixes according to the features i about, then when analysing word forms, it would be doing this while ing about no features, and would therefore do a lot of needless tracking.

Another disadvantage with having just one routine for both directio processing, is that in the production of word forms, the system wo doing far more checking of the lexicon than necessary. In Kosken ation rules in the lexicon, and find any morphological information stored with the stem for production as described above, it is not necessary to constantly check whether the word begins with a prefix or a stem, as the system should know that it has been given a stem. This checking is, however, necessary in the analysis of word forms.

It is therefore proposed that there should be distinct routines for the production and the analysis of word forms, the former taking the stem and features and returning the complete lexical representation before carrying out the phonological processing, and the latter proceeding in a similar way to the present system.

The adaptations mentioned above would also improve the system from a linguistic point of view, since the lexicon as proposed by Koskenniemi does not distinguish affixes from stems, but treats them as equal. While this may not be important from a purely implementational point of view, it would appear to be undesirable from a theoretical point of view, and indeed, it is easy to envisage difficulties when trying to incorporate the system into a larger language system, since we would want to extract syntactic and semantic information from the stem of a word, rather than any of its inflectional affixes.

Other problems which have not been mentioned above include other inflectional processes, such as infixation and reduplication, but space prohibits a discussion of these here. The improvements discussed above are

Appendix A: The set of Zwicky-type rules for German

1. In the context of [CAT:Adjal], [CASE:acc, GEND:masc, NUM:sg] is realized by the suffixation of /en/.
2. In the context of [CAT:adj, CLASS:wk], [CASE:Direct, NUM:sg] is realized by the suffixation of /e/.
3. In the context of [CAT:adj, CLASS:wk], any bundle of CASE, GEND and NUM values is realized by the suffixation of /en/.
4. In the context of [CAT:Adjal], [CASE:Dir, GEND:masc, NUM:sg] is realized by the suffixation of /er/.
5. In the context of [CAT:Adjal], [CASE:Dir, GEND:neut, NUM:sg] is realized by the suffixation of /es/.
6. In the context of [CAT:Adjal], [CASE:Dir, GEND:fem, NUM:sg] is realized by the suffixation of /e/.
7. In the context of [CAT:Adjal], [GEND:masc, NUM:sg] has the same realization as [GEND:neut].
8. In the context of [CAT:Adjal], [CASE:gen, GEND:neut, NUM:sg] is realized by the suffixation of /en/.
9. In the context of [CAT:Adjal], [CASE:dat, GEND:neut, NUM:sg] is realized by the suffixation of /em/.
10. In the context of [CAT:Adjal], [CASE:Obl, GEND:fem, NUM:sg] is realized by the suffixation of /er/.
11. In the context of [CAT:Nounal], [CASE:dat, NUM:pl] is realized by the suffixation of /en/.
12. In the context of [CAT:Adjal], [NUM:pl] has the same realization as [GEND:fem, NUM:sg].
13. In the context of [CAT:det], [CASE:gen, GEND:neut, NUM:sg] is realized by the suffixation of /es/.
14. In the context of [CAT:Adjal, CLASS:mixed], [CASE:Dir, NUM:sg] has the same realization as [CLASS:str].
15. In the context of [CAT:Adjal, CLASS:mixed], any bundle of CASE, GEN and NUM values has the same realization as [CLASS:wk].
16. In the context of [CAT:noun], [CASE:gen, GEND:neut, NUM:sg] is realized by the suffixation of /es/.
17. In the context of [CAT:noun], [CASE:gen, GEND:masc, NUM:sg] is realized by the suffixation of /s/.
18. In the context of [CAT:noun, CLASS:wk], [CASE:nom, NUM:sg] is realized by the suffixation of /-/.
19. In the context of [CAT:noun, CLASS:wk], any bundle of CASE, GEND and NUM values is realized by the suffixation of /en/.

Appendix B: The alphabet module - aliases and character pair sets

```
alias(nnl,[nn,adj,det]).
alias(adjl,[adj,det]).
alias(cat,[nn,adj,det,prep,vb]).
alias(class,[wk,str,inxd])•
alias(case,[nan,ace,gen,dat]).
alias(dir,[ncm,aoc]).
alias(dbl,[gen,dat]).
alias(gend,[masc,fera,neut]).
alias(num,[sg,pl]).
alias(sl,[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,'a"','o"
        •u"1])-
alias(vl,[a,e,i,o,u,"a111,'o111,fufl1])•
alias(cl,[b,c,d,f,g,h,j,k,l,in,n,p,q,r,s,t,v,w,x,2]).
alias(bl,[a,o,u])•
alias(ul,['a"','o"','u"']).
alias (xl, [nn,adj, det, wk,str,inxd,inasc,fem,neut,ncm,acc,gen/dat,sg/pl,l/
      4,5,6]).
alias(=,[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/lafff,lollf
     •u111  nn,adj,det,vi?k,stx,itD^,inasc,fem,neut,rx]m^acc,gen,dat,sg,pl,
      1,2,3,4,5,6]).
alias(plg,[1,2,3,4,5,6]).
alias(ug,[2,4,5]).
alias(suffixnum, [suffl#suff2,suff4]).
alias(ognlex,[suff1,suff2,suff3,suff4,suff5]).


chatest(X,X):-              definitions of ciiaracter pair sets
    findalias(X,sl,List).
chatest(X,O):-
    findalias(X,xl,Iist).
chatest(a,falfl).
chatest(o, fotn).
chatest(u, fuIlt).
chatest(O,n).
chatest(e,O)•
chatest(u,O).
chatest(m,0).


chacheck([Oial|Restl],[Cha2|Rest2],[Chal|Restl],[Cha2|Rest2]):-
    chatest(Chal,Cha2).
chacheck([Chal|Restl],[Cha2|Rest2],[ChaliPestl],[0,Cha2,Rest2]):-
    chatest(Chal,0),
    not(Cha2 = 0)
```

```
la.  e            =     =    6        =     pl
          <==>              ____
     0            a     r    0        n     0


Ib.  e            ≐     =    6        =     pl
          <=>           -·  --
     0            e     ⊂    0        n     0
```

these two rules are combined into a single automaton

```
table(l,[[final,[1,1,1,1,0,1,1]],
    [[a,a],[2,2,2,0,0,0,0]],
    [[e,e],[3,3,3,0,0,0,0]],
    [[r,r],[2,3,3,0,0,0,0]],
    [[6,0],[1,1,4,0,0,0,0]],
    [[e,0],[0,0,0,5,0,0,0]],
    [[n,n],[1,1,1,0,6,0,0]],
    [[pl,0],[1,1,1,0,0,6,0]],
    [[gen,0],[1,1,1,0,0,7,0]],
    [[-,-],[1,1,1,0,0,0,0]],
    [[x1,0],[1,2,3,4,5,6,7]]]).
```

```
2.  0        =   ≐   6     -   -   pl

  n          i   n   0     e   n   0
```

```
table(2,[[final,[1,1,1,1,0,0,0]],
    [[-,i],[2,1,1,0,0,0,0]],
    [[=,e],[1,1,1,0,6,0,0]],
    [[-,n],[1,3,1,0,0,7,0]],
    [[6,0],[1,1,4,0,0,0,0]],
    [[pl,0],[1,1,3,4,0,0,1]],
    [[0,n],[0,0,0,5,0,0,0]],
    [[Xl,0],[1,1,3,4,0,0,7]],
    [[-,-],[1,1,1,0,0,0,0]]]).
```

```
3. bl               ug   pl
    <=> ______
  Ul                0    0
```

```
table(3,[[final,[1,0,0,0]],
    [[bl,ul],[2,0,0,0]],
    [[ug,0],[4,3,0,0]],
    [[pl,0],[1,0,1,0]],
    [[sg,0],[i,o,o,i]],
    [[Xl,0],[1,2,3,4]],
    [[-,-],[1,2,3,4]]]).
```

4. u_0 m_0 $\Longleftrightarrow$ 6_0 $\_\_\_$ $=_e$ $=_n$ pl_0

```
table(4,[[final,[1,0,0,0]],
     [[U,0],[2,0,0,0]],
     [[m,0],[0,3,0,0]],
     [[6,0],[1,0,4,0]],
     [[pl,0],[1,0,0,1]],
     [[sg,0],[1,0,0,0]],
     [[Xl,0],[1,0,3,4]],
     [[=,=],[1,0,0,4]]]).
```

Appendix D: FST module for phonological processing Koskenniemi-style

```
?- reconsult('procedures.pl').
?- reconsult('tables.pl').
?- reconsult('alphabet.pl').


finalstate([],Statelist).
finalstate([[Name,Table]|Tail],Statelist):-
    final(Name,Table,Statelist),
    finalstate(Tail,Statelist).


final(Name,[[First|[List|_]]|_],[[Name,State]|_]):-
    listnum(State,List,Num),
    Num is 1.
final(Name,List,[Head|Tail]):-
    final(Name,List,Tail).


move([Word1,Word2],[],Statelist1,Statelist1).
move([Word1,Word2],[Head|Tail],Statelist1,Statelist2):-
    nextmove([Word1,Word2],Head,Statelist1,Statelist3),
    move([Word1,Word2],Tail,Statelist3,Statelist2).


newstate(Name,States,[[Name,State]|Oldtail],[[Name,Nextstate]|Oldtai
    listnum(State,States,Nextstate).
newstate(Name,States,[Head|Oldtail],[Head|Newtail]):-
    newstate(Name,States,Oldtail,Newtail).


nextmove([Word1,Word2],[Name,Table],Statelist1,Statelist2):-
    member([[Word1,Word2]|[States1|_]],Table),
    newstate(Name,States1,Statelist1,Statelist2);
    member([[Word1,Word3]|[States2|_]],Table),
    check(1,Word1,Word2,Word3,Table),
    newstate(Name,States2,Statelist1,Statelist2);
    member([[Word4,Word2]|[States3|_]],Table),
    check(2,Word1,Word2,Word4,Table),
    newstate(Name,States3,Statelist1,Statelist2);
    findmatch(Word1,Word2,Table,States4),
    newstate(Name,States4,Statelist1,Statelist2).


findmatch(Word1,Word2,Table,States4):-
    member([[X,Y]|[States4|_]],Table),
    not(checdc(4,Wordl/Word2,Listl,ˡrable)),
    not(dheck(3, X,Word2,List2, Table)),
    not(check(4,Y,Vfordl,Listl,Table)).


check(l,Wordl,Word2,Vford3,Table)  :-
    findalias(Word2 ,Wbrd3, list),
    ixjt(member([ [WDrdl,Word2] |_] ,Table)),
    rKDt(otheralias(l,Wordl,Wbrti2,List/'rable)) .
check(2,Wbrdl,WbrxJ2/Word3/Table)  :-
    findalias (Wbrdl,Wbrd3, List),
    not(meaiiber( [ [Wonil,Word2] |_] ,Table)),
    iK3t(otheralias(2,Wbrdl,Wbrd2,List/Table)) .
checik(3,WordlfWord2,Listl,Table)  :-
    member([[Wordl,X]|_],Table),
    alias(X,List2),
    member(Word2,List2),
    subset(List2,Listl).
check(4/Wordl,Word2/Listl,Table) :-
    member([[X,Wordl]|_],Table),
    alias(X,List2),
    member(Word2,List2),
    subset(List2,Listl).
dheck(5,Wordl,Wbrd2,Table):-
    member([[Wordl,X]|_],Table),
    findalias(Word2,X,List);
    member([[Y,Word2]|_]fTable),
    findalias(Wordl,Y,List).


findalias(Wordl,Word2,List) :-
    alias(Word2,List),
    member(Wbrdl,List).
```

Appendix E i: "Lexbuild" module for creating sub-lexicons

```
?- reconsult('procedures.pl').

lexlist(Lexname,List):-
    lexicon(Lexname,Lex),
    lexlist2(List,Lex,Newlex),
    assertlex(Lexname,Lex,Newlex).

lexlist2([],Lex,Lex).
lexlist2([[Word,C]|Tail],Lex,Newlex):-
    add(Word,Lex,Nextlex,C),
    lexlist2(Tail,Nextlex,Newlex).

assertlex(Lexname,Lex,Newlex):-
    Lex = Newlex;
    retract(lexicon(Lexname,Lex)),
    assert(lexicon(Lexname,Newlex)).

add([Head],[[Head,X,Rest1]|Rest2],[[Head,Z,Rest1]|Rest2],C):-
    X = 0,
    Z is C;
    X = C,
    Z is C;
    Z is [X,C].
add([Head1],[[Head2|Rest1]|Rest2],[[Head2|Rest1]|Rest3],C):-
    add([Head1],Rest2,Rest3,C).
add([Head|Tail],[[Head,X,Rest1]|Rest2],[[Head,X,Rest3]|Rest2],C):-
    add(Tail,Rest1,Rest3,C).
add([Head1|Tail],[[Head2|Rest1]|Rest2],[[Head2|Rest1]|Rest3],C):-
    add([Head1|Tail],Rest2,Rest3,C).
add([Head],[],[[Head,C,[]]],C).
add([Head|Tail],[],[[Head,0,Rest]],C):-
    add(Tail,[],Rest,C).

writelex(Lexname):-
    lexicon(Lexname,X),
    write(lexicon(Lexname,X)).
```

Appendix E ii: Lexicon checking module

```
?- reconsult('lexicon.pl').
?- reconsult('procedures.pl').

lex(Word,B,Lex):-
    test(Word,Lex,C),
    member(B,C).

test([Head],[[Head,C,_]|_],C).
test([Head1],[[Head2|Rest1]|Rest2],C):-
    test([Head1],Rest2,C).
test([Head|Tail],[[Head,X,Rest1]|Rest2],C):-
    test(Tail,Rest1,C).
test([Head1|Tail],[[Head2|Rest1]|Rest2],C):-
    test([Head1|Tail],Rest2,C).

lextest(Lexword,Cha1,Lexname,Lexname,Info,Info,Lexword):-
    findalias(Cha1,x1,List).
lextest(Lexword,Cha1,Lexname,Newlexname,Info,Newinfo,Newlexword):-
    conc(Lexword,[Cha1],Lexword2),
    lextest2(Lexword2,Lexname,Newlexname,Info,Newinfo,Newlexword).

lextest2(Lexword,Lexname,Lexname,Info,Info,Lexword):-
    lexicon(Lexname,Lex),
    not(lex(Lexword,X,Lex)).
lextest2(Lexword,Lexname,Newlexname,Info1,Newinfo2,[]):-
    lexicon(Lexname,Lex),
    lex(Lexword,Info2,Lex),
    conc(Info1,Info2,Newinfo1),
    newlextest(Lexname,Newinfo1,Newinfo2,Newlexname).

initiallex(Lexname):-
    newlextest(Lexname,X,Y,*).

finallex(Lexname):-
    newlextest(Lexname,X,Y,#).

match(X,Y,Z):-
    X = Y,
    Y = Z;
    X = Y,
    findalias(X,Z,L);
    findalias(X,Y,L1),
    findalias(X,Z,L2),
    subset2(L1,L2);
    alias(X,L1),
    alias(Z,L3),
    subset(L1,L2),
    subset(L2,L3).

match1(X,Y,Z):-
    match(X,Y,Z),
    not(alias(X,List)).

match2(X,Y,Z):-
    match(X,Y,Z);
    not(match(X,Y,Z)),
    match(Y,X,Z).

match3(Cat,Cat1,Class,Class1,Gend,Gend1,Case,Case1,Num,Num1):-
    match1(Cat,Cat1,cat),
    match1(Class,Class1,class),
    match1(Gend,Gend1,gend),
    match1(Case,Case1,case),
    match1(Num,Num1,num).
```

Appendix E iii: Lexicon - containing sublexicons and rules for ooribination

```
?- reoonsult(fprooedures.pl1).
?- reccnsult(falphabet.pl1).

lexioon(stem, [[d, 0, [[i, 0, [[e, 0, [[s, [[det, str, _l]], []]]]]
    [j, 0, [[e, 0, [[n, [[det, str, JL]], []]]]]], [e, 0, [[i, 0, [[i
    [[det, str, JL]], []]]]]], [g, 0, [[u, 0, [[t, [[adjf str, _i]]f
    []]]], [r# 0, [[o, 0f [[s, 0f [[s, [[adj, str, _l]], []]]]]]]]]
    [k, 0, [[1, 0, [[e, 0, [[i, 0, [[n, [[adj, str, _l]]f []]]]]]]],
    [a, 0, [[t, 6, [[z, 0, [[e, [[noun, str, fern, 6]], []]]]]]]], [i
    [[r, 0, [[c, 0, [[h, 0, [[e, [[noun, str, fern, 6]], []]]]]]]]]]
    [b, 0, [[i, 0, [[1, 0, [[1, 0, [[i, 0, [[g, [[adj, str, _1]],
    []]]]]]]]]]]# [wr 0, [[a, 0, [[g, 0, [[e, 0, [[n, [[noun, str,
   masc, 1]], []]]]]]]]]], [f, 0, [[u, 0, [[s, 0, [[s, [[noun, str,
   masc, 4]], []]]]]]]], [m, 0, [[e, 0, [[s, 0, [[s, 0, [[e, 0, [[r
    [[noun, str, neut, 1]], []]]]]]]]]]]]).


lexicx>n(suffl,  [[e,  [],  []]]).
Iexioon(suff2,  [[e,  0,  [[n,  [],  []]]]]).
Iexioon(suff3,  [[e,  0,  [[s,  [],  []]]]]).
Iexicon(suff4,  [[e,  0,  [[r,  [],  []]]]]).
Iexicx)n(suff5,  [[e,  0,  [[m,  [],  []]]]]).
Iexicon(suff6,  [[s,  [],  []]]).

newlextest (prefix, X,X,*).
nev^lextest (prefix, X,X, stem).
newlextest(stem,X,X,*) •
newlextest(stem,X,X,#).
nev/lextest(stem,I^o,Newinfo,Suffixlex)    :-
     iaesmber(nn,Info),
     findalias(Plg,plg,X),
     loernber (Pig, Info),
     suffixnum(Plg,Suffixlex),
     oonc(Info, [pi] ,Newinfo).
newlextest(stem,Iiifo,Newirifo,C3gnlex)  :-
     cgncheck Info Newingo C nlex).
newlextest(Suffixnum,X,X,#)  :-
     findal.ias(Suffixnum, suffixnum,A).
newlextest(Suffixram,Info,Newinfo,C3g^ex)  :-
     member(nn,Info),
     ineacnber(pl,Info),
     findalias(Suffixnum, suffixnum, X),
     cgncheck(Info,NewinfoJOgn..ex).
newlextest(Ognlex,X,X,#)  :-
     findalias(Ognlex, ognlex,A).
    member(Class,Info),
    findalias(Gend,gend,C),
    member(Gend,Info),
    cgntestproc1(T,X,Cat,Class,Case,Gend,Num,Suffixlex),
    not(checklex(T,X,Cat,Class,Case,Gend,Num)),
    conc(Info,[Case,Num],Newinfo).

cgntestproc1(T,X,Cat,Class,Case,Gend,Num,Lex):-
    match3(Cat,Cat1,Class,Class1,Gend,Gend1,Case,Case1,Num,Num1)
    cgntest(e,X,Cat1,Class1,Case1,Gend1,Num1,Lex);
    match3(Cat,Cat1,Class,Class1,Gend,Gend1,Case,Case1,Num,Num1)
    cgntest(r,X,Cat1,Class1,Case1,Gend1,Num1,Lex).


cgntestproc2(T,X,Cat,Class,Case,Gend,Num,Lex):-
    cgntest(T,X,Cat1,Class1,Case1,Gend1,Num1,Lex),
    match2(Cat,Cat1,cat),
    match2(Class,Class1,class),
    match2(Gend,Gend1,gend),
    match2(Case,Case1,case),
    match2(Num,Num1,num).

checklex(A,X,Cat,Class,Case,Gend,Num):-
    cgntestproc1(B,Y,Cat,Class,Case,Gend,Num,Lex),
    X = Y,!, fail.
checklex(A,X,Cat,Class,Case,Gend,Num):-
    cgntestproc1(B,Y,Cat,Class,Case,Gend,Num,Lex),
    X > Y.


suffixnum(3,suff1).
suffixnum(4,suff1).
suffixnum(5,suff4).
suffixnum(6,suff2).

cgntest(e,1,adj1,class,acc,masc,sg,suff2).
cgntest(e,2,adj,wk,dir,gend,sg,suff1).
cgntest(e,3,adj,wk,case,gend,num,suff2).
cgntest(e,4,det,class,gen,neut,sg,suff3).
cgntest(e,5,noun,class,gen,neut,sg,suff3).
cgntest(e,6,noun,class,gen,masc,sg,suff6).
cgntest(e,7,noun,wk,nom,gend,sg,#).
cgntest(e,8,noun,wk,case,gend,num,suff2).
cgntest(r,9,adj1,mxd,dir,gend,sg,Lex):-
    cgntestproc2(A,X,adj1,str,dir,gend,sg,Lex),
    not(X = 9).
cgntest(r,10,adj1,mxd,case,gend,num,Lex):-
    cgntestproc2(A,X,adj1,str,case,gend,pl,Lex),
     cgntestproc2(A#X#adjl#class/case,neiit/sg#Lex),
     not(X « 14).
ogntest(e,15,adjlfclass,gen,neiitfsg#suff2).
cgntest(e#16#adjl#class#dat#neutfsg#suff5) •
cutest(e, 17, adjl, class, chl, fem,sg,suff4).
cgntest(e, 18,nounal,class,dat,gerKi,pl,suff2).
ogntest(r,19,noun^,class,case,gend,pl,Lex) :-
     ogntestproca(A#X#r«inal/class#c^se#f^^ sg,Lex) ,
     not(X « 19),
```

Appendix F: The process module which controls the whole process

```
?- library(findall).
?- reconsult('lexproc.pl').
?- reconsult('procedures.pl').
?- reconsult('alphabet.pl').
?- reconsult('fst.pl').


processbegin(List1,List2):-
    findall([Name,1],table(Name,Table),Statelist1),
    bagof([Name,Table],table(Name,Table),Tablelist),
    initiallex(Lexname),
    process(List1,List2,Statelist1,Statelist2,Tablelist,Lexname,[],Info,
        Newinfo).


process([],[],Statelist1,Statelist2,Tablelist,Lexname,[],Info,Newinfo):-
    finalstate(Tablelist,Statelist1),
    finallex(Lexname).
process([Char1|String1],[Char2|String2],Statelist1,Statelist2,Tablelist,
        Lexname,Lexword,Info,Newinfo):-
    chacheck([Char1|String1],[Char2|String2],[Cha1|Rest1],[Cha2|Rest2]),
    move([Cha1,Cha2],Tablelist,Statelist1,Statelist3),
    lextest(Lexword,Cha1,Lexname,Newlexname,Info,Newlexword),
    process(Rest1,Rest2,Statelist3,Statelist2,Tablelist,Newlexname,Newlex
```

The following is a run of the program, giving the lexical representat
and asking for the surface representation, with the current input pai
the current statelist and the current lexicon printed out at each pai
Prolog will try to find the value of the variables X, which is the
surface representation which corresponds to the lexical representatio
given, and Y, which is the suffix needed for the lexical representati
given the morphological features provided.

```
?- process([f,u,s,s,nn,str,masc,4,Y,nom,pl],X).

input pair = [f,f]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = stem

input pair = [u,u]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = stem

input pair = [s,s]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = stem

input pair = [s,s]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = stem

input pair = [nn,0]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = stem

input pair = [str,0]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = stem

input pair = [masc,0]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = stem

input pair = [4,0]
statelist  = [[1,1],[2,1],[3,4],[4,1]]
lexicon    = stem
```

with the morphological information it has above, it finds that one possible continuation is the -<u>e</u> suffix, if it is plural ...

```
input pair = [e,e]
statelist  = [[1,3],[2,1],[3,4],[4,1]]
lexicon    = suff1
```

.... it then finds that the assumed plural feature was correct...

```
input pair = [pl,0]
statelist  = [[1,1],[2,1],[3,4],[4,1]]
lexicon    = suff1

        at this point it backtracks because table 3 is in a non-final
        state and the input string has ended, due to the umlaut not
        being present with the plural feature. Most of the backtracking
        has been cut out...

input pair = [u,'u"']
statelist  = [[1,1],[2,1],[3,2],[4,1]]
lexicon    = stem

input pair = [s,s]
statelist  = [[1,1],[2,1],[3,2],[4,1]]
lexicon    = stem

input pair = [s,s]
statelist  = [[1,1],[2,1],[3,2],[4,1]]
lexicon    = stem

input pair = [nn,0]
statelist  = [[1,1],[2,1],[3,2],[4,1]]
lexicon    = stem

input pair = [str,0]
statelist  = [[1,1],[2,1],[3,2],[4,1]]
lexicon    = stem

input pair = [masc,0]
statelist  = [[1,1],[2,1],[3,2],[4,1]]
lexicon    = stem

input pair = [4,0]
statelist  = [[1,1],[2,1],[3,3],[4,1]]
lexicon    = suff1

input pair = [e,e]
statelist  = [[1,3],[2,1],[3,3],[4,1]]
lexicon    = suff1

input pair = [pl,0]
statelist  = [[1,1],[2,1],[3,1],[4,1]]
lexicon    = #            - the # indicates that the word terminates here.
                            All the automata are in final states, and the
                            lexicon system is also, so the word has been
                            successfully produced:

X = [f,'u"',s,s,0,0,0,e,0] ?
Y = e ?
yes
```

References

Gazdar,G.J.M. and G.Pullum - "Generalized Phrase Structure Grammar: A Theoretical Synopsis", IULC, 1982.

Hammer,A.E. - "German Grammar and Usage", Arnold, 1971.

Karttunen,L., K.Koskenniemi and R.M.Kaplan - "A Compiler for Two-leve Phonological Rules", Xerox Palo Alto Research Paper, CSLI, 1987.

Koskenniemi,K. - "A Two-level Morphological Processor" - PhD dissertation, University of Helsinki, 1983.

Zwicky,A. - "How to describe Inflection" in BLS 1985.