FORMALISMS FOR MORPHOGRAPHEMIC DESCRIPTION

Alan Black, Graeme Ritchie,
Dept of Artificial Intelligence, University of Edinburgh
80 South Bridge, Edinburgh EH1 1HN, SCOTLAND

Steve Pulman and Graham Russell
Computing Laboratory, University of Cambridge
Corn Exchange Street, Cambridge CB2 3QG, ENGLAND

ABSTRACT

Recently there has been some interest in rule formalisms for describing morphologically significant regularities in orthography of words, largely influenced by the work of Koskenniemi. Various implementations of these rules are possible, but there are some weaknesses in the formalism as it stands. An alternative specification formalism is possible which solves some of the problems. This new formalism can be viewed as a variant of the "pure" Koskenniemi model with certain constraints relaxed. The new formalism has particular advantages for multiple character changes. An interpreter has been implemented for the formalism and a significant subset of English morphographemics has been described, but it has yet to be used for describing other languages.

Background

This paper describes work in a particular area of computational morphology, that of **morphographemics**. Morphographemics is the area dealing with systematic discrepancies between the surface form of words and the symbolic representation of the words in a lexicon. Such differences are typically orthographic changes that occur when basic lexical items are concatenated; e.g. when the stem *move* and suffix *+ed* are concatenated they form *moved* with the deletion of an *e+*. The work discussed here does not deal with the wider issue of which morphemes can join together. (The way we have dealt with that question is described in Russell *et al.* (1986)).

The framework described here is based on the two-level model of morphographemics (Koskenniemi 1983) where rules are written to describe the relationships between **surface forms** (e.g. *moved*) and **lexical forms** (e.g. *move+ed*). In his thesis, Koskenniemi (1983) presents a formalism for describing morphographemics. In the early implementations (Koskenniemi 1983, Karttunen 1983) although a high-level notation was specified the actual implementation was by *hand*-compilation into a form of finite state machine. Later implementations have included automatic compilation techniques (Bear 1986, Ritchie *et al* 1987), which take in a high-level specification of surface-to-lexical relationships and produce a directly interpretable set of automata. This pre-compilation is based on the later work of Koskenniemi (1985).

Note that there is a distinction between the *formalism* and its *implementation*. Although the Koskenniemi formalism is often discussed in terms of automata (or transducers) it is not always necessary for the morphologist using the system to know exactly how the rules are implemented, but only that the rules adhere to their defined interpretation. A suitable formalism should make it easier to specify spelling changes in an elegant form. Obviously for practical reasons there should be an efficient implementation, but it is not necessary for the specification formalism to be identical to the low-level representation used in the implementation.

As a result of our experience with these rule systems, we have encountered various limitations or inelegances, as follows:

- in a realistically sized rule set, the description may be obscure to the human reader;
- different rules may interact with each other in non-obvious and inconvenient ways;
- certain forms of correspondence demand the use of several rules in an clumsy manner;
- some optional correspondences are extremely difficult to describe.

Some of these problems can be overcome using a modified formalism, which we have also implemented and tested, although it also has its limitations.

Koskenniemi Rules

The exact form of rule described here is that used in our work (Russell *et al.* 1986, Ritchie *et al.* 1987) but is the same as Koskenniemi's (1983, 1985) apart from some minor changes in surface syntax. Koskenniemi Rules describe relationships between a sequence of **surface characters** and a sequence of **lexical characters**. A rule consists of a **rule pair** (which consists of a lexical and a surface character), an **operator**, a **left context** and a **right context**. There are three types of rule:

Context Restriction: These are of the form

pair $\rightarrow$ LeftContext --- RightContext

This specifies that the rule pair may appear *only* in the given context.

Surface Coercion: These are of the form

pair $\leftarrow$ LeftContext --- RightContext

This specifies that if the given contexts and lexical character appear then the surface character *must* appear.

Combined Rule: This final rule type is a combination of the above two forms and is written

pair $\longleftrightarrow$ LeftContext --- RightContext

This form of rule specifies that the surface character of the rule pair *must* appear if the left and right context appears and the lexical character appears, and also that this is the *only* context in which the rule pair is allowed.

The operator types may be thought of as a form of implication. Contexts are specified as regular expressions of lexical and surface pairs. For example the following rule:

Epenthesis
+:e $\longleftrightarrow$ {s:s x:x z:z < {s:s c:c} h:h>} --- s:s

specifies (some of) the cases when an *e* is inserted at the conjunction of a stem morpheme and the suffix *+s* (representing plurals for nouns and third person singular for verbs). The braces in the left context denote optional choices, while the angled brackets denote sequences. The above rule may be summarised as "an *e* must be inserted in the surface string when it has *s*, *x*, *z*, *ch* or *sh* in its left context and *s* in its right".

Another addition to the formalism is that alternative contexts may be specified for each rule pair. This is done with the *or* connective for multiple left and right contexts on the right hand side of the rule e.g.

Elision
e:0 $\longleftrightarrow$ C:C --- < +:0 V:V>
or < C:C V:V> --- < +:0 e:e>

This example also introduces sets - *C* and *V* (which are elsewhere declared to represent consonants and vowels). The *or* construct states that *e* can correspond to *0* (the null symbol) when (and only when) in *either* of the two given contexts. The first option above copes with words such as *moved* resolving with *move+ed* and the second deals with examples like *agreed* resolving with *agree+ed*.

Sets have a somewhat non-standard interpretation within this basic formalism. The expansion of them is done in terms of the **feasible set**. This is the set of all lexical and surface pairs mentioned anywhere in the set of rules. That is, all identity pairs from the intersection of the lexical and surface alphabets and all **concrete pairs** from the rules, where concrete pairs are those pairs that do not contain sets. The interpretation of a pair containing a set is all members of the feasible set that match. This means that if *y:i* is a member of the feasible set and a set *Ve* is declared for the set {*a e i o u y*} the pair *Ve:Ve* represents the pair *y:i* as well as the more obvious ones.

Traditionally, (if such a word can be used), Koskenniemi Rules are implemented in terms of finite state machines (or transducers). KIMMO (Karttunen 1983), one of the early implementations, required the morphologist to specify the rules directly in transducer form which was difficult and prone to error. Koskenniemi (1985) later described a possible method for compilation of the high-level specification into transducers. This means the morphologist does not have to write and debug low-level finite state machines.

Problems with Koskenniemi Formalism

The basic idea behind the Koskenniemi Formalism - that rules should describe correspondences between a surface string and a lexical string (which effectively represents a normal form) - appears to be sound. The problems listed here are not fundamental to the underlying theory, that of describing relationships between surface and lexical strings, but are more problems with the exact form of the rule notation. The formalism as it stands does not make it impossible to describe many phenomena but can make it difficult and unintuitive.

One problem is that of interaction between rules. This is when a pair that is used in a context part of a rule A is also restricted by some other rule B, but the context within which the pair appears in A is not a valid context with respect to B. An example will help to illustrate this. Suppose, having developed the *Elision* rule given above, the linguist wishes to introduce a rule which expresses the correspondence between *reduction* and the lexical form *reduce+ation*, a phenomenon apparently unrelated to elision. The obvious rules are:

Elision
e:0 $\longleftrightarrow$ C:C --- < +:0 V:V >
or < C:C V:V > --- < +:0 e:e >
A-deletion
a:0 $\longleftrightarrow$ < c:c e:0 +:0 > --- t:t

However, these rules do not operate independently. The pair *e:0* in the left context of the *A-deletion* rule is not licensed by the *Elision* rule as it occurs in a context (*c:c* --- < *+:0 a:0* >) which is not valid with respect to the right context of the *Elision* rule, since the *V:V* pair does not match the pair *a:0*. The necessary *Elision* rule to circumvent this problem is:

Elision
e:0 ←→ C:C --- < +:0 V:V >
or < C:C V:V > --- < +:0 e:e >
or c:c --- < +:0 a:0>

Such possible situations mean that the writer of the rules must check, every time the *rule pair* from a rule A is used within one of the context statements of another rule B, that the character sequence in that context statement is valid with respect to rule A. Theoretically it would be possible for a compiler to check for such cases although this would require finding the intersection of the languages generated by the set of finite state automata which is computationally expensive (Garey and Johnson 1979 p266).

A similar problem which is more easily detected is what can be termed *double coercion.* This is when two rules have the same lexical character in their rule pair, and their respective left and right contexts have an intersection. The situation which could cause this is where an underlying lexical character can correspond to two different surface characters, in different contexts, with the correspondence being completely determined by the context, but with one context description being more general than (subsuming) the other. For example, the following rules allow lexical *l* to map to surface null or surface *i* (and might be proposed to describe the generation of forms like *probably* and *probability* from *probable*):

L-deletion
l:0 ←→ b:b --- < e:0 +:0 l:l >
L-to-I
l:i ←→ b:b --- { e:0 e:l }

Matching the surface string *b00* to the lexical string *ble* (as demanded by the first rule) would be invalid because the second rule is coercing the lexical *l* to a surface *i*; similarly the surface string *bi0* would not be able to match the lexical string *ble* because of the first rule coercing the lexical *l* to a surface *0*. (Again, such conflicts between rules could in principle be detected by a compiler). There appears to be no simple way round this within the formalism. A possible modification to the formalism which would stop conflicts occurring would be to disallow the inclusion of more than one rule with the same lexical character in the rule-pair, but this seems a little too restrictive.

One argument that has been made against the Koskenniemi Formalism is that multiple character changes require more than one rule. That is where a *group* of characters on the surface match a *group* on in the lexicon (as opposed to one character changing twice, which is not catered for nor is intended to be in the frameworks presented here). For example in English we may wish to describe the relationship between the surface form *application* and the lexical form *apply+ation* as a two character change *i c* to *y +*. The general way to deal with multiple character changes in the Koskenniemi Formalism is to write a rule for each character change. Where a related character change is referred to in a context of rule it should be written as a lexical character and an "=" on the surface. Where "=" is defined as a surface set that consists of all surface characters. Thus the *application* example can be encoded as follows.

Y-to-I
y:i ←→ --- < +:= a:a {t:t l:l b:b}>
C-insertion
+:c ←→ y:= --- < a:a{t:t l:l b:b}>

The "=" on the surface must be used to ensure that the rules enforce each other. If the following were written

Y-to-I
y:i ←→ --- < +:c a:a {t:t l:l b:b}>
C-insertion
+:c ←→ y:i --- < a:a {t:t l:l b:b}>

then *apply0ation* would still be matched with *apply+ation*. This technique is not particularly intuitive but does work. It has been suggested that a compiler could automatically do this.

Another problem is that because only one rule may be written for each pair, the rules are effectively sorted by pair rather than phenomena so when a change is genuinely a multiple change the characters changes in it cannot necessarily be described together, thus making a rule set difficult to read.

Because of the way sets are expanded, the interpretation of rules depends on all the other rules. The addition or deletion of a spelling rule may change the feasible pair set and hence a rule's interpretation may change. The problem is not so much that the rules then need re-compiled (which is not a very expensive operation) but that interpretation of a rule cannot be viewed independently from the rest of the rule set.

The above problems are all actually criticisms of the elegance of the formalism for describing spelling phenomena as opposed to actual restrictions in its descriptive power. However, one problem that has been pointed out by Bear is that rule pairs can only have one type of operator so that a pair may not be optional in one context but mandatory in another.

There has also been some discussion of the formal descriptive power of the formalism, particularly the work of Barton (1986). Barton has shown that the question of finding a lexical/surface correspondence from an arbitrary Koskenniemi rule set is NP-complete. It seems intuitively wrong to suggest that the process of morphographemic analysis of natural language is computationally difficult, and hence Barton's result suggests that the formalism is actually more powerful than is really needed to describe the phenomenon. A less powerful formalism would be desirable.

A final point is that although initially this high-level formalism appears to be easy to read and comprehend from the writer's point of view, in practice when a number of rules are involved this ceases to be the case. We have found that debugging these rules is a slow and difficult task.

Alternative Formalism

This section proposes a formalism which is basically similar to the "pure" Koskenniemi one. Again a description consists of a set of rules. There are two types of rule which allow the description of the two types of changes that can occur, *mandatory* changes and *optional* changes.

The rules can be of two types, first **surface-to-lexical rules** which are used to describe optional changes and **lexical-to-surface rules** which are used to describe mandatory changes. the interpretation is as follows

Surface-to-lexical Rules: These rules are of the form

LHS $\rightarrow$ RHS

Where *LHS* and *RHS* are simple lists of surface and lexical characters respectively, each of the same length. The interpretation is that for a surface string and lexical string to match there must be a partition of the surface string such that each partition is a *LHS* of a rule and that the lexical string is equal to the concatenation of the corresponding *RHS*s.

Lexical-to-Surface Rules: These rules are of the form

LHS $\leftarrow$ RHS

The *LHS* and *RHS* are equal length strings of surface and lexical characters respectively. Their interpretation is that *any* substring of a lexical string that is a *RHS* of a rule must correspond to the surface string given in the corresponding *LHS* of the rule.

This asymmetry in the application rules means that *LS-Rules* (lexical-to-surface rules) can overlap while *SL-Rules* (surface-to-lexical rules) do not. An example may help to explain their use.

A basic set of spelling rules in this formalism would consist of first the simple list of identity SL-Rules

a $\rightarrow$ a
b $\rightarrow$ b
c $\rightarrow$ c
. . .
z $\rightarrow$ z

which could be automatically generated from the intersection of the surface and lexical alphabets. In addition to this basic set we would wish to add the rule

0 $\rightarrow$ +

which would allow us to match null with a special character marking the start of a suffix. These rules would then allow us to match strings like *boy0s* to *boy+s*, *girl* to *girl* and *walk0ing* to *walk+ing*.

To cope with epenthesis we can add SL-Rules of the form

s e s $\rightarrow$ s + s
x e s $\rightarrow$ x + s
z e s $\rightarrow$ z + s
c h e s $\rightarrow$ c h + s
s h e s $\rightarrow$ s h + s

This would allow matching of forms like *boxes* with *box+s* and *matches* with *match+s* but still allows *box0s* with *box+s*. We can make the adding of the *e* on the surface mandatory rather than just optional by adding a corresponding LS-Rule for each SL-Rule. In this case if we add the LS-Rules

s e s $\leftarrow$ s + s
x e s $\leftarrow$ x + s
z e s $\leftarrow$ z + s
c h e s $\leftarrow$ c h + s
s h e s $\leftarrow$ s h + s

the surface string *box0s* would not match *box+s* because this would violate the LS-Rule; similarly, *match0s* would not match *match+s*.

However if some change is optional and not mandatory we need only write the SL-Rule with no corresponding LS-Rule. For example, assuming the word *hoof* has the alternative plurals *hooves* or *hoofs*, we can describe this optional change by writing the SL-Rule

v e s $\rightarrow$ f + s

The main difference between this form of rules and the Koskenniemi rules is that now one rule can be written for multiple changes where the Koskenniemi Formalism would require one for *each* character change. For example, consider the double change described above for matching *application* with *apply+ation*. This required two distinct rules in the Koskenniemi Format, while in the revised formalism only two clearly related rules are required

i c a t → y + a t
i c a t ← y + a t

One problem which the formalism as it stands does suffer from is that it requires multiple rules to describe different "cases" of changes e.g. each case of epenthesis requires a rule — one each for words ending in *ch*, *sh*, *s*, *x* and *z*. In our implementation rules may be specified with sets instead of just simple characters thus allowing the rules to be more general. Unfortunately this is not sufficient as the user really requires to specify the left and right hand sides of rules as regular expressions, thus allowing rules such as:

$$< \{ < \{ s\ c \}\ h > x\ z\ s \}\ e\ s > \rightarrow$$
$$< \{ < \{ s\ c \}\ h > x\ z\ s \} + s >$$

but this seems to significantly reduce the readability of the formalism.

One useful modification to this formalism could be the collapsing of the two types of rule (LS and SL). It appears that an LS-Rule is never required without a corresponding SL-Rule so we could change the formalism so that we have two operators → for the simple SL-Rule for optional changes and ←→ to represent the corresponding SL and LS-Rules for mandatory changes.

So far we have implemented an interpreter for this alternative formalism and written a description of English. Its coverage is comparable with our English description in the Koskenniemi Formalism but the alternative description is possibly easier to understand. The implementation of these rules is again in the form of special automata which check for valid and invalid patterns, like that of the Koskenniemi rules. This is not surprising as both formalisms are designed for licensing matches between surface and lexical strings. The time for compilation and interpretation is comparable with that for the Koskenniemi rules.

Comparison of the two formalisms

It is interesting to note that if we extended the Koskenniemi formalism to allow regular expressions of *pairs* on the left hand side of rules rather than just simple pairs, we get a formalism that is very similar to our alternative proposal. The main difference then is the lack of contexts in which the rules apply — in the alternative formalism the rules are also specifying the correspondences for what would be contexts in the Koskenniemi formalism.

Because SL-Rules do not overlap this means phenomena which are physically close together or overlapping have to be described in one rule, thus it may be the case that changes have to be declared in more than one place. For example, one could argue that there is e-deletion in the matching of *reduction* to *reduce+ation* (thus following the Koskenniemi Formalism) or that the change is a double change in that the e-deletion *and* the a-deletion are the same phenomena (as in this new formalism). But there may also be cases where the morphologist identifies two separate phenomena which can occur together in some circumstances. In this new formalism rules would be required for each phenomena and also where the two overlap. One example of this in English may be *quizzes* where both consonant doubling and e-insertion apply. In this formalism a rule would need to be written for the combined phenonmena as well as each individual case. Ideally, a rule formalism should not require information to be duplicated, so that phenomena are only described in one place. In English this does not occur often so seems not to be a problem but this is probably not true for languages with richer morphographemics such as Finnish and Japanese.

Interaction between rules however can in a sense still exist, but in the formalism's current form it is significantly easier for a compiler to detect it. SL-Rules do not cause interaction, since different possible partitions of the surface string represent different analyses (not conflicting analyses). Interaction can happen only with LS-Rules, which in principle may have overlapping matches and hence may stipulate conflicting surface sequences for a single lexical sequence. Interaction will occur if any RHS of a rule is a substring of a RHS of any other rule (or concatenation of rules) and has a different corresponding LHS. With the formalism only allowing simple strings in rules this would be relatively easy to detect but if regular expressions were allowed the problem of detection would be the same as in the Koskenniemi Formalism. Double coercion in the new formalism is actually only a special case of interaction.

The interpretation of symbols representing sets of characters has been changed so that adding and deleting rules does not affect the other rules already in the rule set. This seems to be an advantage, as each rule may be understood in isolation from others.

One main advantage of the new formalism is that changes can be optional or mandatory. If some change (say e-deletion) is sometimes mandatory and sometimes optional there will be distinct rules that describe the different cases.

As regards the computational power of the formalism, no detailed analysis has been made, but intuitively it is suspected to be equivalent to the Koskenniemi Formalism. That is, for every set of these rules there is a set of Koskenniemi rules that accepts/rejects the same surface and lexical matches and vice versa. The formal power seems an independent issue here as neither formalism has particular advantages.

It may be worth noting that both formalisms are suitable for generation as well as recognition. This is due to the use of the two-level model (surface and lexical strings), rather than the formalism notations.

Future Work

Although this alternative formalism seems to have some advantages over the Koskenniemi Formalism (optional and mandatory changes, set notation and multiple character changes), there is still much work to be done on the development of the new formalism. The actual surface syntax of this new formalism requires some experimentation to find the most suitable form for easy specification of the rules. Both the Koskenniemi Formalism and the new one seem adequate for specification of English morphographemics (which is comparatively simple) but the real issue appears to be which of them allows the writer to describe the phenomena in the most succinct form.

One of the major problems we have found in our work is that although formalisms appear simple when described and initially implemented, actual use often shows them to be complex and difficult to use. There is a useful analogy here with computer programming languages. New programming languages offer different and sometimes better facilities but in spite their help, effective programming is still a difficult task. To continue the analogy, both these morphographemic formalisms require a form of debugger to allow the writer to test the rule set quickly and find its short-comings. Hence we have implemented a debugger for the Koskenniemi Formalism. This debugger acts on user given surface and lexical strings and allows *step* or *diagnosis* modes. The *step* mode describes the current match step by step *in terms of the user written rules*, and explains the reason for any failures (rule blocking, no rule licensing a pair etc). The *diagnosis* mode runs the match to completion and summarises the rules used and any failures if they occur. The important point is that the debugger describes the problems in terms of the user written rules rather than some low level automata. In earlier versions of our system debugging of our spelling rules was very difficult and time consuming. We do not yet have a similar debugger for our new formalism but if fully incorporated into our system we see a debugger as a necessary part of the system to make it useful.

Another aspect of our work is that of testing our new formalism with other languages. English has a somewhat simple morphographemics and is probably not the best language to test our formalism on. The Koskenniemi Formalism has been used to describe a number of different languages (see Gazdar (1985) for a list) and seems adequate for many languages. Semitic languages, like Arabic, which have discontinuous changes have been posed as problems to this framework. Koskenniemi (personal communication) has shown that in fact his formalism is adequate for describing such languages. We have not yet used our new formalism for describing languages other than English, but we feel that it should be at least as suitable as the Koskenniemi Formalism.

Conclusion

This paper has described the Koskenniemi Formalism which can be used for describing morphographemic changes at morpheme boundaries. It has pointed out some problems with the basic formalism as it stands and proposes a possible alternative. This alternative is at least as adequate for describing English morphographemics and may be suitable for at least the languages which the Koskenniemi Formalism can describe.

The new formalism is possibly better, as initially it appears to be more intuitive and simple to write but from experience this cannot be said with certainty until the formalism has been significantly used.

Acknowledgements

We would like to thank Kimmo Koskenniemi for comments on an earlier draft of this paper. This work was supported by SERC/Alvey grant GR/C/79114.

References

Barton, G. Edward 1986 Computational Complexity in Two-Level Morphology In *Proceedings ACL '86, 24th Annual Meeting of Association for Computational Linguistics* 53-59.

Bear, John 1986 A Morphological Recogniser with Syntactic and Phonological Rules In *Proceedings COLING '86, 11th International Conference on Computational Linguistics* 272-276.

Garey, Michael R.; and Johnson, David S. 1979 *Computers and Intractability: A Guide to the Theory of NP-Completeness* W.H.Freeman and Co., New York.

Gazdar, Gerald 1985 Finite State Morphology: a review of Koskenniemi (1983). Report No. CSLI-85-32, CSLI, Stanford University.

Karttunen, Lauri 1983 KIMMO: A General Morphological Analyser *Texas Linguistics Forum*, 22:165-186.

Koskenniemi, Kimmo 1983 Two-level Morphology: a general computational model for word-form recognition and production. Publication No.11, Department of General Linguistics, University of Helsinki, Finland.

Koskenniemi, Kimmo 1985 Compilation of Automata from Two-Level Rules. Talk given at Workshop on Finite-State Morphology, CSLI, Stanford

University, July 1985.

Ritchie, Graeme D.; Pulman, Steve G.; Black, Alan W.; and Russell Graham J. 1987 A Computational Framework For Lexical Description. DAI Research Paper no. 293, University of Edinburgh. Also to appear in *Computational Linguistics*.

Russell Graham J.; Pulman, Steve G.; Ritchie, Graeme D.; and Black, Alan W. 1986 A Dictionary and Morphological Analyser for English. In *Proceedings COLING '86, 11th International Conference on Computational Linguistics* 277-279.