A MORPHOLOGICAL RECOGNIZER WITH SYNTACTIC AND PHONOLOGICAL RULES

John Bear
Artificial Intelligence Center
SRI International
333 Ravenswood Ave
Menlo Park, CA 94025
U.S.A.

Abstract

This paper describes a morphological analyzer which, when parsing a word, uses two sets of rules: rules describing the syntax of words, and rules describing facts about orthography.

1 Introduction[1]

In many natural language processing systems currently in use, the morphological phenomena are handled by programs which do not interpret any sort of rules, but rather contain references to specific morphemes, graphemes, and grammatical categories. Recently Kaplan, Kay, Koskenniemi, and Karttunen have shown how to construct morphological analyzers in which the descriptions of the orthographic and syntactic phenomena are separable from the code. This paper describes a system that builds on their work in the area of phonology/orthography and also has a well defined syntactic component which applies to the area of computational morphology for the first time some of the tools that have been used in syntactic analysis for quite a while.

This paper has two main parts. The first deals with the orthographic aspects of morphological analysis, the second with its syntactic aspects. The orthographic phenomena constitute a blend of phonology and orthography. The orthographic rules given in this paper closely resemble phonological rules, both in form and function, but because their purpose is the description of orthographic facts, the words *orthography* and *orthographic* will be used in preference to *phonology* and *phonological*.

The overall goal of the work described herein is the development of a flexible, usable morphological analyzer in which the rules for both syntax and spelling are (1) separate from the code, and (2) descriptively powerful enough to handle the phenomena encountered when working with texts of written language.

2 Orthography

The researchers mentioned above use finite-state transducers for stipulating correspondences between surface segments, and underlying segments. In contrast, the system described in this paper does not use finite state machines. Instead, orthographic rules are interpreted directly, as constraints on pairings of surface strings with lexical strings.

The rule notation employed, including conventions for expressing abbreviations, is based on that described in Koskenniemi [1983,1984]. The rules actually used in this system are based on the account of English in Karttunen and Wittenburg [1983].

2.1 Rules

What follows is an inductive introduction to the types of rules needed. Some pertinent data will be presented, then some potential rules for handling these data. We shall also discuss the reasons for needing a weaker form of rule and indicate what it might look like.

Let us first consider some data regarding English /s/ morphemes:

ALWAYS -ES
box+s ⟷ boxes
class+s ⟷ classes
fizz+s ⟷ fizzes
spy+s ⟷ spies
ash+s ⟷ ashes
church+s ⟷ churches
ALWAYS -S
slam+s ⟷ slams
hit+s ⟷ hits
tip+s ⟷ tips
...
SOMETIMES -ES,
SOMETIMES -S
piano+s ⟷ pianos
solo+s ⟷ solos
do+s ⟷ does
potato+s ⟷ potatoes
banjo+s ⟷ banjoes or banjos
cargo+s ⟷ cargoes or cargos

Below are presented two possible orthographic rules for describing the foregoing data:

R1) + ⟶ e {x | z | y/i | s (h) | c h} _ s
R2) + ⟶ e {x | z | y/i | s (h) | c h | o} _ s

The first of these rules will be shown to be too weak; the second, in contrast, will be shown to be too strong. This fact will serve as an argument for introducing a second kind of rule.

[1]I am indebted to Lauri Karttunen and Fernando Pereira for all their help. Lauri supplied the initial English automata on which the orthographic grammar was based, while Fernando furnished some of the Prolog code. Both provided many helpful suggestions and explanations as well. I would also like to thank Kimmo Koskenniemi for his comments on an earlier draft of this paper.

This research was supported by the following grants: Naval Electronics Systems Command N00039-84-K-0078; Navelex N00039-84-C-0524 P00003; Office of Naval Research N00014-85-C-0013.

Before describing how the rules should be read, it is necessary to define two technical terms. In phonology, one speaks of underlying segments and surface segments; in orthography, characters making up the words in the lexicon contrast with characters in word forms that occur in texts. The term *lexical character* will be used here to refer to a character in a word or morpheme in the lexicon, i.e., the analog of a phonological underlying segment. The term *surface character* will be used to mean a character in a word that could appear in text. For example, [l o v e + e d] is a string of lexical characters, while [l o v e d] is a string of surface characters.

We may now describe how the rules should be read. The first rule should be read roughly as, "a morpheme boundary [+] at the lexical level corresponds to an [e] at the surface level whenever it is between an [x] and an [s], or between a [z] and an [s], or between a lexical [y] corresponding to a surface [i] and an [s], or between an [s h] and an [s] or between a [c h] and an [s]." This means, for instance, that the string of lexical characters [c h u r c h + s] corresponds to the string of surface characters [c h u r c h e s] (forgetting for the moment about the possibility that other rules might also obtain). The second rule is identical to the first except for an added [o] in the left context.

When we say [+] corresponds to [e] between an [x] and an [s], we mean between a lexical [x] corresponding to a surface [x] and a lexical [s] corresponding to a surface [s]. If we wanted to say that it does not matter what the lexical [x] corresponds to on the surface, we would use [x/=] instead of just [x].

The rules given above get the facts right for the words that do not end in [o]. For those that do, however, Rule 1 misses on [do+s] $\Longleftrightarrow$ [does], [potato+s] $\Longleftrightarrow$ [potatoes]; Rule 2 misses on [piano+s] $\Longleftrightarrow$ [pianos], [solo+s] $\Longleftrightarrow$ [solos]. Furthermore, neither rule allows for the possibility of more than one acceptable form, as in [banjo+s] $\Longleftrightarrow$ ([banjoes] or [banjos]), [cargo+s] $\Longleftrightarrow$ ([cargoes] or [cargos]).

The words ending in [o] can be divided into two classes: those that take an [es] in their plural and third-person singular forms, and those that just take an [s]. Most of the facts could be described correctly by adopting one of the two rules, e.g., the one stating that words ending in [o] take an [es] ending. In addition to adopting this rule, one would need to list all the words taking an [s] ending as being irregular. This approach has two problems. First, no matter which rule is chosen, a very large number of words would have to be listed in the lexicon; second, this approach does not account for the coexistence of two alternative forms for some words, e.g., [banjoes] or [banjos].

The data and arguments just given suggest the need for a second type of rule. It would stipulate that such and such a correspondence is *allowed* but *not required*. An example of such a rule is given below:

R3) +/e allowed in context o _ s.

Rule 3 says that a morpheme boundary may correspond to an [e] between an [o] and an [s]. It also has the effect of saying that if a morpheme boundary ever corresponds to an [e], it must be in a context that is explicitly allowed by some rule.

If we now have the two rules R1 and R3,

R1) + $\longrightarrow$ e / {x | z | y/i | s (h) | c h} _ s
R3) +/e allowed in context o _ s,

we can generate all the correct forms for the data given. Furthermore, for the words that have two acceptable forms for plural or third person singular, we get both, just as we would like. The problem is that we generate both forms whether we want them or not. Clearly some sort of restriction on the rules, or "fine tuning," is in order; for the time being, however, the problem of deriving both forms is not so serious that it cannot be tolerated.

So far we have two kinds of rules, those stating that a correspondence always obtains in a certain environment, and those stating that a correspondence is allowed to obtain in some environment. The data below argue for one more type of rule, namely, a rule stipulating that a certain correspondence never obtains in a certain environment.

DATA FOR CONSONANT DOUBLING
DOUBLING:
bar+ed $\longleftrightarrow$ barred
big+est $\longleftrightarrow$ biggest
refer+ed $\longleftrightarrow$ referred
NO DOUBLING:
question+ing $\longleftrightarrow$ questioning
hear+ing $\longleftrightarrow$ hearing
hack+ing $\longleftrightarrow$ hacking
BOTH POSSIBILITIES:
travel+ed $\longleftrightarrow$ (travelled or traveled) both are allowed

In English, final consonants are doubled if they, "follow a single [orthographic] vowel and the vowel is stressed." [from Karttunen and Wittenburg 1983]. So for instance, in [hear+ing], the final [r] is preceded by two vowels, so there is no doubling. In [hack+ing], the final [k] is not preceded by a vowel, so there is no doubling. In [question+ing], the last syllable is not stressed so again there is no doubling.

In Karttunen and Wittenburg [1983] there is a single rule listed to describe the data. However, the rule makes use of a diacritic (') for showing stress, and words in the lexicon must contain this diacritic in order for the rule to work. The same thing could be done in the system being described here, but it was deemed undesirable to allow words in the lexicon to contain diacritics encoding information such as stress. Instead, the following rules are used. Ultimately, the goal is to have some sort of general mechanism, perhaps negative rule features, for dealing with this sort of thing, but for now no such mechanism has been implemented.

RULES FOR CONSONANT DOUBLING
"Allowed-type" rules

'+'/b allowed in context vV b _ vV[2]
'+'/c allowed in context vV c _ vV
'+'/d allowed in context vV d _ vV
'+'/f allowed in context vV f _ vV
'+'/g allowed in context vV g _ vV
'+'/l allowed in context vV l _ vV
'+'/m allowed in context vV m _ vV
'+'/n allowed in context vV n _ vV
'+'/p allowed in context vV p _ vV
'+'/r allowed in context vV r _ vV
'+'/s allowed in context vV s _ vV
'+'/t allowed in context vV t _ vV
'+'/z allowed in context vV z _ vV

"Disallowed-type" rules

'+'/b disallowed in context vV vV b _ vV
'+'/c disallowed in context vV vV c _ vV
'+'/d disallowed in context vV vV d _ vV

[2]In these rules, the symbol vV stands for any element of the following set of orthographic vowels: {a,e,i,o,u}.

'+'/f disallowed in context vV vV f _ vV
'+'/g disallowed in context vV vV g _ vV
'+'/l disallowed in context vV vV l _ vV
'+'/m disallowed in context vV vV m _ vV
'+'/n disallowed in context vV vV n _ vV
'+'/p disallowed in context vV vV p _ vV
'+'/r disallowed in context vV vV r _ vV
'+'/s disallowed in context vV vV s _ vV
'+'/t disallowed in context vV vV t _ vV
'+'/z disallowed in context vV vV z _ vV

The allowed-type rules in the top set are those that license consonant doubling. The disallowed-type rules in the second set constrain the doubling so it does not occur in words like [eat+ing] $\Longleftrightarrow$ [eating] and [hear+ing] $\Longleftrightarrow$ [hearing]. The disallowed-type rules say that a morpheme boundary [+] may not ever correspond to a consonant when the [+] is followed by a vowel and preceded by that same consonant and then two more vowels.

The rules given above suffer from the same problem as the previous rules, namely, over generation. Although they produce all the right answers and allow multiple forms for words like [travel+er] $\Longleftrightarrow$ ([traveller] or [traveler]), which is certainly a positive result, they also allow multiple forms for words which do not allow them. For instance they generate both [referred] and [refered]. As mentioned earlier, this problem will be tolerated for the time being.

2.2 Comparison with Koskenniemi's Rules

Koskenniemi [1983, 1984] describes three types of rules, as exemplified below:

R4) a > b $\Longrightarrow$ c/d e/f _ g/h i/j
R5) a > b $\Longleftarrow$ c/d e/f _ g/h i/j
R6) a > b $\Longleftrightarrow$ c/d e/f _ g/h i/j.

Rule R4 says that if a lexical [a] corresponds to a surface [b], then it must be within the context given, i.e., it must be preceded by [c/d e/f] and followed by [g/h i/j]. This corresponds exactly to the rule given below:

R7) a/b allowed in context c/d e/f _ g/h i/j.

The rule introduced as R5 and repeated below says that if a lexical [a] occurs following [c/d e/f] and preceding [g/h i/j], then it must correspond to a surface [b]:

R5) a > b $\Longleftarrow$ c/d e/f _ g/h i/j.

The corresponding rule in the formalism being proposed here would look approximately like this:

R10) a/sS disallowed in context c/d e/f _ g/h i/j,

where sS is some set of characters to which
[a]
can correspond that does not include [b].

A comparison of each system's third type of rule involves compostion of rules and is the subject of the next section.

2.3 Rule Composition and Decomposition

In Koskenniemi's systems, rule composition is fairly straightforward. Samples of the three types of rules are repeated here:

R4) a > b $\Longrightarrow$ c/d e/f _ g/h i/j
R5) a > b $\Longleftarrow$ c/d e/f _ g/h i/j
R6) a > b $\Longleftrightarrow$ c/d e/f _ g/h i/j

If a grammar contains the two rules, R4 and R5, they can be replaced by the single rule R6.

In contrast, the composition of rules in the system proposed here is slightly more complicated. We need the notion of a default correspondence. The default correspondence for any alphabetic character is itself. In other words, in the absence of any rules, an alphabetic character will correspond to itself. There may also be characters that are not alphabetic, e.g., the [+] representing a morpheme boundary, currently the only non-alphabetic character in this system. Other conceivable non-alphabetic characters would be an accent mark for representing stress, or say, a hash mark for word boundaries. The default for these characters is that they correspond to 0 (zero). Zero is the name for the null character used in this system.

Now it is easy to say how rules are composed in this system. If a grammar contains both R11 and R12 below, then R13 may be substituted for them with the same effect:

R11) a/b allowed in context c/d e/f _ g/h i/j
R12) a/ *"a's default"* disallowed in context c/d e/f _ g/h i/j
R13) a $\longrightarrow$ b / c/d e/f _ g/h i/j

In fact, when a file of rules is read into the system, occurrences of rules like R13 are internalized as if the grammar really contained a rule like R11 and another like R12.

2.4 Using the Rules

Again consider for an example the rule R1 repeated below.

R1) + $\longrightarrow$ e / {x | z | y/i | s (h) | c h} _ s

When this rule is read in, it is expanded into a set of rules whose contexts do not contain disjunction or optionality. Rules R14 through R19 are the result of the expansion:

R14) '+' $\longrightarrow$ e / x _ s
R15) '+' $\longrightarrow$ e / z _ s
R16) '+' $\longrightarrow$ e / y/i _ s
R17) '+' $\longrightarrow$ e / s _ s
R18) '+' $\longrightarrow$ e / s h _ s
R19) '+' $\longrightarrow$ e / c h _ s.

R14 through R19 are in turn expanded automatically into R20 through R31 below:

R20) '+'/0 disallowed in context x _ s
R21) '+'/0 disallowed in context z _ s
R22) '+'/0 disallowed in context y/i _ s
R23) '+'/0 disallowed in context s _ s
R24) '+'/0 disallowed in context s h _ s
R25) '+'/0 disallowed in context c h _ s
R26) '+'/e allowed in context x _ s
R27) '+'/e allowed in context z _ s
R28) '+'/e allowed in context y/i _ s
R29) '+'/e allowed in context s _ s
R30) '+'/e allowed in context s h _ s
R31) '+'/e allowed in context c h _ s.

The disallowed-type rules given here stipulate that a morpheme boundary, lexical [+], may never be paired with a null surface character, [0], in the environments indicated. Another way to describe what disallowed-type rules do, in general, is to say that they expressly rule out certain sequences of pairs of letters. For example, R20

R20) +/0 disallowed in context x _ s

states that the sequence

```
...x + s...
   | | |
...x 0 s...
```

is never permitted to be a part of a mapping of a surface string to a lexical string.

The allowed-type rules behave slightly differently than their disallowed-type counterparts. A rule such as

R26) '+'/e allowed in context x _ s,

says that lexical [+] is not normally allowed to correspond to surface [e]. It also affirms that lexical [+] may appear between an [x] and an [s]. Other rules starting with the same pair say, in effect, "here is another environment where this pair is acceptable." The way these rules are to be interpreted is that a rule's main correspondence, i.e., the character pair that corresponds to the underscore in the context, is forbidden except in contexts where it is expressly permitted by some rule.

Once the rules are broken into the more primitive allowed-type and disallowed-type rules, there are several ways in which one could try to match them against a string of surface characters in the recognition process. One way would be to wait until a pair of characters was encountered that was the main pair for a rule, and then look backwards to see if the left context of the rule matches the current analysis path. If it does, put the right context on hold to see whether it will ultimately be matched.

Another possiblility would be to continually keep track of the left contexts of rules that are matching the characters at hand, so that when the main character of a rule is encountered, the program already knows that the left context has been matched. The right context still needs to be put on hold and dealt with the same way as in the other scheme.

The second of the two strategies is the one actually employed in this system, though it may very well turn out that the first one is more efficient for the current grammar of English.

2.5 Possible Correspondences

The rules act as filters to weed out sequences of character pairs, but before a particular mapping can be weeded out, something needs to propose it as being possible. There is a list — called a list of possible correspondences, or sometimes, a list of feasible pairs — that tells which characters may correspond to which others. Using this list, the recognizer generates possible lexical forms to correspond to the input surface form. These can then be checked against the rules and against the lexicon. If the rules do not weed it out, and it is also in the lexicon, we have successfully recognized a morpheme.

3 Syntax

The goal of the work being described was an analyzer that would be easy to use. In the area of syntax, this entails two subgoals. First, it should be easy to specify which morphemes may combine with which, and second, when the recognition has been completed, the result should be something that can easily be used by a parser or some other program.

Karttunen [1983] and Karttunen and Wittenburg [1983] have some suggestions for what a proper syntactic component for a morphological analyzer might contain. They mention using context-free rules and some sort of feature-handling system as possible extensions of both their and Koskenniemi's systems. In short, it has been acknowledged that any such system really ought to have some of the tools that have been used in syntax proper.

The first course of action that was followed in building this analyzer was to implement a unification system for dags (directed acyclic graphs), and then to have the analyzer unify the dags of all the morphemes encountered in a single analysis. That scheme turned out to be too weak to be practical. The next step was to implement a PATR rule interpreter [Shieber, et al. 1983] so that selected paths of dags could be unified. Finally, when that turned out to be still less flexible than one would like, the capability of handling disjunction in the dags was added to the unification package, and the PATR rule interpreter [Karttunen 1984].

The rules look like PATR rules with the context free skeleton. The first two lines of a rule are just a comment, however, and are not used in doing the analysis. The recognizer starts with the dag [cat: empty]. The rule below states that the "empty" dag may be combined with the dag from a verb stem to produce a dag for a verb.

```
% verb ⟶ empty + verb_stem
%   1        2          3
<2 cat> = empty
<3 cat> = verb_stem
<3 type> = regular
<1 type> = <3 type>
<1 cat> = verb
<1 word> = <3 lex>
<1 form> = {inf
            [tense: pres
            pers: {1 2} ] } .
```

The resulting dag will be ambiguous between an infinitive verb, and a present tense verb that is in either the first or second person. (The braces in the rule are the indicators of disjunction.) The verb stem's value for the feature *lex* will be whatever spelling the stem has. This value will then be the value for the feature *word* in the new dag.

The analyzer applies these rules in a very simple way. It always carries along a dag representing the results found thus far. Initially this dag is [cat: empty]. When a morpheme is found, the analyzer tries to combine it, via a rule, with the dag it has been carrying along. If the rule succeeds, a new dag is produced and becomes the dag carried along by the analyzer. In this way the information about which morphemes have been found is propagated.

If an [ing] is encountered after a verb has been found, the following rule builds the new dag. It first makes sure that the verb is infinitive (form: inf) so that the suffix cannot be added onto the end of a past participle, for instance, and then makes the tense of the new dag be pres_part for present participle. The category of the new dag is *verb*, and the value for *word* is the same as it was in the original verb's dag. The form of the input verb is a disjunction of *inf* (infinitive) with [tense: pres, pers: {1 2}], so the unification succeeds.

```
% verb ⟶ verb + ing
%   1        2     3
<2 cat> = verb
<3 lex> = ing
<2 form> = inf
<1 cat> = verb
<1 word> = <2 word>
<1 form> = [tense: pres_part] .
```

The system also has a rule for combining an infinitive verb with the nominalizing [er] morpheme, e.g., swim : swimmer. This rule, given below, also checks the form of the input verb to verify that it is infinitive. It makes the resulting dag have *category: noun, number: singular*, and so on.

```
% noun ⟶ verb + er
%   1        2     3
<2 cat> = verb
<3 lex> = er
<2 form> = inf
<1 cat> = noun
<1 word> = <2 word>
<1 nbr> = sg
<1 pers> = 3 .
```

The noun thus formed behaves just the same as other nouns. In particular, a pluralizing [s] may be added, or a possessive ['s], or any other affix that can be appended to a noun.

There are other rules in the grammar for handling adjective endings, more verb endings, etc. Irregular forms are handled in a fairly reasonable way. The irregular nouns are listed in the lexicon with *form: irregular*. Other rules than the ones shown here refer to that feature; they prevent the addition of plural morphemes to words that are already plural. Irregular verbs are listed in the lexicon with an appropriate value for tense (not unifiable with inf) so that the test for infinitiveness will fail when it should. Irregular adjectives, e.g. good, better, best are dealt with in an analogous manner.

4 Further Work

There are still some things that are not as straightforward as one would like. In particular, consider the following example. Let us suppose as a first approximation that one wanted to analyze the [un] prefix in English as combining with adjectives to yield new ones, e.g., unfair, unclear, unsafe. Suppose also that one wanted to be able to build past participles of transitive verbs (passives) into adjectives, so that they could combine with [un], as in uncovered, unbuilt, unseen.

What we would need, would be a rule to combine an "empty" with an [un] to make an [un] and then a rule to combine an [un] with a verb stem to form a thing1, and finally a rule to combine a thing1 with a past participle marker to form a negative adjective. More rules would be needed for the case where [un] combines with an adjective stem like [fair]. In addition, rules would be needed for irregular passives, etc.

In short, without a more sophisticated control strategy, the grammar would contain a fair amount of redundancy if one really attempted to handle English morphology in its entirety. However, on a more positive note, the rules do allow one to deal effectively and elegantly with a sufficient range of phenomena to make it quite acceptable as, for instance, an interface between a parser and its lexicon.

5 Conclusion

A morphological analyzer has been presented that is capable of interpreting both orthographic and syntactic rules. This represents a substantial improvement over the method of incorporating morphological facts directly into the code of an analyzer. The use of these rules leads to a powerful, flexible morphological analyzer.

References

[1] Karttunen, L. (1983) "Kimmo: A General Morphological Processor," in *Texas Linguistic Forum* #22, Dalrymple et al., eds., Linguistics Department, University of Texas, Austin, Texas.

[2] Karttunen, L. (1984) "Features and Values," in *COLING 84*.

[3] Karttunen, L. and K. Wittenburg (1983) "A Two-level Morphological Analysis Of English," in *Texas Linguistic Forum* #22, Dalrymple et al., eds., Linguistics Department, University of Texas, Austin, Texas.

[4] Kay, M. (1983) "When Meta-rules are not Meta-rules," in K. Sparcke-Jones, and Y. Wilkes, eds. *Automatic Natural Language Processing*, John Wiley and Sons, New York.

[5] Koskenniemi, K. (1983) "Two-level Model for Morphological Analysis," *IJCAI 83*, pp. 683-685.

[6] Koskenniemi, K. (1984) "A General Computational Model for Word-form Recognition and Production," *COLING 84*, pp. 178-181.

[7] Selkirk, E. (1982) *The Syntax of Words*, MIT Press.

[8] Shieber, S., H. Uszkoreit, F. Pereira, J. Robinson, and M Tyson (1983) "The Formalism and Implementation of PATR-II," in B. Grosz, and M. Stickel (1983) *Research on Interactive Acquisition and use of Knowledge*, SRI Final Report 1894, SRI International, Menlo Park, California.