COMPUTATIONAL COMPLEXITY IN TWO-LEVEL MORPHOLOGY

G. Edward Barton, Jr.
M.I.T. Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA 02139

ABSTRACT

Morphological analysis must take into account the spelling-change processes of a language as well as its possible configurations of stems, affixes, and inflectional markings. The computational difficulty of the task can be clarified by investigating specific models of morphological processing. The use of finite-state machinery in the "two-level" model by Kimmo Koskenniemi gives it the appearance of computational efficiency, but closer examination shows the model does not guarantee efficient processing. Reductions of the satisfiability problem show that finding the proper lexical/surface correspondence in a two-level generation or recognition problem can be computationally difficult. The difficulty increases if unrestricted deletions (null characters) are allowed.

INTRODUCTION

The "dictionary lookup" stage in a natural-language system can involve much more than simple retrieval. Inflectional endings, prefixes, suffixes, spelling-change processes, reduplication, non-concatenative morphology, and clitics may cause familiar words to show up in heavily disguised form, requiring substantial morphological analysis. Superficially, it seems that word recognition might potentially be complicated and difficult.

This paper examines the question more formally by investigating the computational characteristics of the "two-level" model of morphological processes. Given the kinds of constraints that can be encoded in two-level systems, how difficult could it be to translate between lexical and surface forms? Although the use of finite-state machinery in the two-level model gives it the appearance of computational efficiency, the model itself does not guarantee efficient processing. Taking the Kimmo system (Karttunen, 1983) for concreteness, it will be shown that the general problem of mapping between lexical and surface forms in two-level systems is computationally difficult in the worst case; extensive backtracking is possible. If null characters are excluded, the generation and recognition problems are NP-complete in the worst case. If null characters are completely unrestricted, the problems is PSPACE-complete, thus probably even harder. The fundamental difficulty of the problems does not seem to be a precompilation effect.

In addition to knowing the stems, affixes, and co-occurrence restrictions of a language, a successful morphological analyzer must take into account the spelling-change processes that often accompany affixation. In English, the program must expect `love+ing` to appear as `loving`, `fly+s` as `flies`, `lie+ing` as `lying`, and `big+er` as `bigger`. Its knowledge must be sufficiently sophisticated to distinguish such surface forms as `hopped` and `hoped`. Cross-linguistically, spelling-change processes may span either a limited or a more extended range of characters, and the material that triggers a change may occur either before or after the character that is affected. (Reduplication, a complex copying process that may also be found, will not be considered here.)

The Kimmo system described by Karttunen (1983) is attractive for putting morphological knowledge to use in processing. Kimmo is an implementation of the "two-level" model of morphology that Kimmo Koskenniemi proposed and developed in his Ph.D. thesis.[1] A system of lexicons in the *dictionary component* regulates the sequence of roots and affixes at the lexical level, while several finite-state transducers in the *automaton component* — $\approx$ 20 transducers for Finnish, for instance — mediate the correspondence between lexical and surface forms. *Null characters* allow the automata to handle insertion and deletion processes. The overall system can be used either for generation or for recognition.

The finite-state transducers of the automaton component serve to implement spelling changes, which may be triggered by either left or right context and which may ignore irrelevant intervening characters. As an example, the following automaton describes a simplified "Y-change" process that changes `y` to `i` before suffix `es`:

[1]University of Helsinki, Finland, *circa* Fall 1983.

```
"Y-Change" 5 5
            y  y  +  s  =    (lexical characters)
            i  y  =  s  =    (surface characters)
state 1:    2  4  1  1  1    (normal state)
state 2.    0  0  3  0  0    (require +s)
state 3.    0  0  0  1  0    (require s)
state 4:    2  4  5  1  1    (forbid +s)
state 5:    2  4  1  0  1    (forbid s)
```

The details of this notation will not be explained here; basic familiarity with the Kimmo system is assumed. For further introduction, see Barton (1985), Karttunen (1983), and references cited therein.

THE SEEDS OF COMPLEXITY

At first glance, the finite-state machines of the two-level model appear to promise unfailing computational efficiency. Both recognition and generation are built on the simple process of stepping the machines through the input. Lexical lookup is also fast, interleaved character by character with the quick left-to-right steps of the automata. The fundamental efficiency of finite-state machines promises to make the speed of Kimmo processing largely independent of the nature of the constraints that the automata encode:

> The most important technical feature of Koskenniemi's and our implementation of the Two-level model is that morphological rules are represented in the processor as automata, more specifically, as finite state transducers .... One important consequence of compiling [the grammar rules into automata] is that the complexity of the linguistic description of a language has no significant effect on the speed at which the forms of that language can be recognized or generated. This is due to the fact that finite state machines are very fast to operate because of their simplicity .... Although Finnish, for example, is morphologically a much more complicated language than English, there is no difference of the same magnitude in the processing times for the two languages .... [This fact] has some psycholinguistic interest because of the common sense observation that we talk about "simple" and "complex" languages but not about "fast" and "slow" ones. (Karttunen, 1983:166f)

For this kind of interest in the model to be sustained, it must be the model itself that wipes out processing difficulty, rather than some accidental property of the encoded morphological constraints.

Examined in detail, the runtime complexity of Kimmo processing can be traced to three main sources. The recognizer and generator must both run the finite-state machines of the automaton component; in addition, the recognizer must descend the letter trees that make up a lexicon. The recognizer must also decide which suffix lexicon to explore at the end of an entry. Finally, both the recognizer and the generator must discover the correct lexical-surface correspondence.

All these aspects of runtime processing are apparent in traces of implemented Kimmo recognition, for instance when the recognizer analyzes the English surface form `spiel` (in 61 steps) according to Karttunen and Wittenburg's (1983) analysis (Figure 1). The stepping of transducers and letter-trees is ubiquitous. The search for the lexical-surface correspondence is also clearly displayed; for example, before backtracking to discover the correct lexical entry `spiel`, the recognizer considers the lexical string `spy+` with `y` surfacing as `i` and `+` as `e`. Finally, after finding the putative root `spy` the recognizer must decide whether to search the lexicon `I` that contains the zero verbal ending of the present indicative, the lexicon `AG` storing the agentive suffix `+er`, or one of several other lexicons inhabited by inflectional endings such as `+ed`.

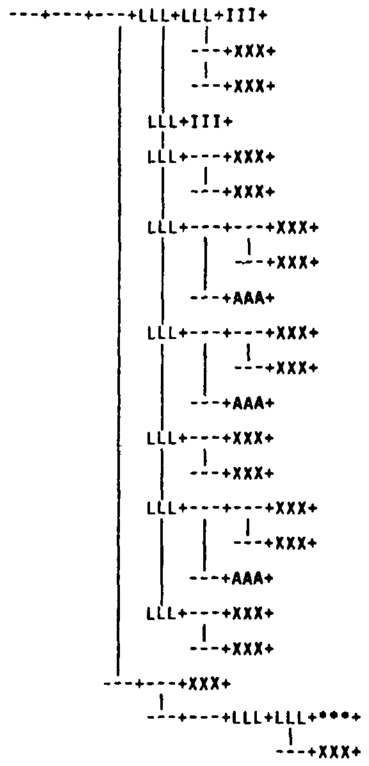
The finite-state framework makes it easy to step the automata; the letter-trees are likewise computationally well-behaved. It is more troublesome to navigate through the lexicons of the dictionary component, and the current implementation spends considerable time wandering about. However, changing the implementation of the dictionary component can sharply reduce this source of complexity; a merged dictionary with bit-vectors reduces the number of choices among alternative lexicons by allowing several to be searched at once (Barton, 1985).

More ominous with respect to worst-case behavior is the backtracking that results from local ambiguity in the construction of the lexical-surface correspondence. Even if only one possibility is globally compatible with the constraints imposed by the lexicon and the automata, there may not be enough evidence at every point in processing to choose the correct lexical-surface pair. Search behavior results.

In English examples, misguided search subtrees are necessarily shallow because the relevant spelling-change processes are local in character. Since long-distance harmony processes are also possible, there can potentially be a long interval before the acceptability of a lexical-surface pair is ultimately determined. For instance, when vowel alternations within a verb stem are conditioned by the occurrence of particular tense suffixes, the recognizer must sometimes see the end of the word before making final decisions about the stem.

```
Recognizing surface form "spiel".
  1        s              1,4,1,2,1,1
  2        sp             1,1,1,2,1,1
  3        spy            1,3,4,3,1,1
  4        "spy" ends,    new lexicon N
  5        "0" ends,      new lexicon C1
  6        spy            XXX extra input
  7 (5)    spy+           1,5,16,4,1,1
  8        spy+           XXX
  9 (5)    spy+           1,6,1,4,1,1
 10        spy+           XXX
 11 (4)    "spy" ends,    new lexicon I
 12        spy            XXX extra input
 13 (4)    "spy" ends,    new lexicon P3
 14        spy+           1,6,1,4,1,1
 15        spy+           XXX
 16 (14)   spy+           1,5,16,4,1,1
 17        spy+           XXX
 18 (4)    "spy" ends,    new lexicon PS
 19        spy+           1,6,1,4,1,1
 20        spy+e          1,1,1,1,4,1
 21        spy+e          XXX
 22 (20)   spy+e          1,1,4,1,3,1
 23        spy+e          XXX
 24 (19)   spy+           1,5,16,4,1,1
 25        spy+e          XXX Epenthesis
 26 (4)    "spy" ends,    new lexicon PP
 27        spy+           1,6,1,4,1,1
 28        spy+e          1,1,1,1,4,1
 29        spy+e          XXX
 30 (28)   spy+e          1,1,4,1,3,1
 31        spy+e          XXX
 32 (27)   spy+           1,5,16,4,1,1
 33        spy+e          XXX Epenthesis
 34 (4)    "spy" ends,    new lexicon PR
 35        spy+           1,6,1,4,1,1
 36        spy+           XXX
 37 (35)   spy+           1,5,16,4,1,1
 38        spy+           XXX
 39 (4)    "spy" ends,    new lexicon AG
 40        spy+           1,6,1,4,1,1
 41        spy+e          1,1,1,1,4,1
 42        spy+e          XXX
 43 (41)   spy+e          1,1,4,1,3,1
 44        spy+e          XXX
 45 (40)   spy+           1,5,16,4,1,1
 46        spy+e          XXX Epenthesis
 47 (4)    "spy" ends,    new lexicon AB
 48        spy+           1,6,1,4,1,1
 49        spy+           XXX
 50 (48)   spy+           1,5,16,4,1,1
 51        spy+           XXX
 52 (3)    spi            1,1,4,1,2,5
 53        spie           1,1,16,1,6,1
 54        spie           XXX
 55 (53)   spie           1,1,16,1,5,6
 56        spiel          1,1,16,2,1,1
 57        "spiel" ends,  new lexicon N
 58        "0" ends,      new lexicon C1
 59        "spiel"        *** result
 60 (58)   spiel+         1,1,16,1,1,1
 61        spiel+         XXX

(("spiel" (N SG)))
```



Key to tree nodes:

---	normal traversal
LLL	new lexicon
AAA	blocking by automata
XXX	no lexical-surface pairs compatible with surface char and dictionary
III	blocking by leftover input
***	analysis found

Figure 1: These traces show the steps that the KIMMO recognizer for English goes through while analyzing the surface form **spiel**. Each line of the table on the left shows the lexical string and automaton states at the end of a step. If some automaton blocked, the automaton states are replaced by an XXX entry. An XXX entry with no automaton name indicates that the lexical string could not be extended because the surface character and lexical letter tree together ruled out all feasible pairs. After an XXX or *** entry, the recognizer backtracks and picks up from a previous choice point, indicated by the parenthesized step number before the lexical string. The tree on the right depicts the search graphically, reading from left to right and top to bottom with vertical bars linking the choices at each choice point. The figures were generated with a KIMMO implementation written in an augmented version of MACLISP based initially on Karttunen's (1983:182ff) algorithm description; the dictionary and automaton components for English were taken from Karttunen and Wittenburg (1983) with minor changes. This implementation searches depth-first as Karttunen's does, but explores the alternatives at a given depth in a different order from Karttunen's.

Ignoring the problem of choosing among alternative lexicons, it is easy to see that the use of finite-state machinery helps control only one of the two remaining sources of complexity. Stepping the automata should be fast, but the finite-state framework does not guarantee speed in the task of guessing the correct lexical-surface correspondence. The search required to find the correspondence may predominate. In fact, the Kimmo recognition and generation problems bear an uncomfortable resemblance to problems in the computational class NP. Informally, problems in NP have solutions that may be hard to *guess* but are easy to *verify* — just the situation that might hold in the discovery of a Kimmo lexical-surface correspondence, since the automata can verify an acceptable correspondence quickly but may need search to discover one.

THE COMPLEXITY OF TWO-LEVEL MORPHOLOGY

The Kimmo algorithms contain the seeds of complexity, for local evidence does not always show how to construct a lexical-surface correspondence that will satisfy the constraints expressed in a set of two-level automata. These seeds can be exploited in mathematical reductions to show that two-level automata can describe computationally difficult problems in a very natural way. It follows that the finite-state two-level framework itself cannot guarantee computational efficiency. If the words of natural languages are easy to analyze, the efficiency of processing must result from some additional property that natural languages have, beyond those that are captured in the two-level model. Otherwise, computationally difficult problems might turn up in the two-level automata for some natural language, just as they do in the artificially constructed languages here. In fact, the reductions are abstractly modeled on the Kimmo treatment of harmony processes and other long-distance dependencies in natural languages.

The reductions use the computationally difficult Boolean satisfiability problems SAT and 3SAT, which involve deciding whether a CNF formula has a satisfying truth-assignment. It is easy to encode an arbitrary SAT problem as a Kimmo generation problem, hence the general problem of mapping from lexical to surface forms in Kimmo systems is NP-complete.[2] Given a CNF formula φ, first construct a string σ by notational translation: use a minus sign for negation, a comma for conjunction, and no explicit operator for disjunction. Then the σ corresponding to the formula $(\bar{x} \vee y)\&(\bar{y} \vee z)\&(x \vee y \vee z)$ is `-xy,-yz,xyz`. The notation is unambiguous without parentheses because φ is required to be in CNF. Second, construct a Kimmo automaton component A in three parts. (A varies from formula to formula only when the formulas involve different sets of variables.) The *alphabet specification* should list the variables in σ together with the special characters T, F, minus sign, and comma; the equals sign should be declared as the Kimmo wildcard character, as usual. The *consistency automata*, one for each variable in σ, should be constructed on the following model:

```
"x-consistency" 3 3
      x  x  =    (lexical characters)
      T  F  =    (surface characters)
  1:  2  3  1    (x undecided)
  2:  2  0  2    (x true)
  3:  0  3  3    (x false)
```

The consistency automaton for variable x constrains the mapping from variables in the lexical string to truth-values in the surface string, ensuring that whatever value is assigned to x in one occurrence must be assigned to x in every occurrence. Finally, use the following *satisfaction automaton*, which does not vary from formula to formula:

```
"satisfaction" 3 4
      =  =  -  ,    (lexical characters)
      T  F  -  ,    (surface characters)
  1.  2  1  3  0    (no true seen in this group)
  2:  2  2  2  1    (true seen in this group)
  3.  1  2  0  0    (-F counts as true)
```

The satisfaction automaton determines whether the truth-values assigned to the variables cause the formula to come out true. Since the formula is in CNF, the requirement is that the groups between commas must all contain at least one true value.

The net result of the constraints imposed by the consistency and satisfaction automata is that some surface string can be generated from σ just in case the original formula φ has a satisfying truth-assignment. Furthermore, A and σ can be constructed in time polynomial in the length of φ; thus SAT is polynomial-time reduced to the Kimmo generation problem, and the general case of Kimmo generation is at least as hard as SAT. Incidentally, note that it is *local* rather than *global* ambiguity that causes trouble; the generator system in the reduction can go through quite a bit of search even when there is just one final answer. Figure 2 traces the operation of the Kimmo generation algorithm on a (uniquely) satisfiable formula.

Like the generator, the Kimmo recognizer can also be used to solve computationally difficult problems. One easy reduction treats 3SAT rather than SAT, uses negated alphabet symbols instead of a negation sign, and replaces the satisfaction automaton with constraints from the dictionary component; see Barton (1985) for details.

[2]Membership in NP is also required for this conclusion. A later section ("The Effect of Nulls") shows membership in NP by sketching how a nondeterministic machine could quickly solve Kimmo generation and recognition problems.

```
Generating from lexical form "-xy,-yz,-y-z,xyz".
 1       -                   1,1,1,3           38   +   -FF,-FT,-F-T,FFT     3,3,2,2
 2       -F                  3,1,1,2           39       "-FF,-FT,-F-T,FFT"   *** result
 3       -FF                 3,3,1,2           40 (3)   -FT                  3,2,1,2
 4       -FF,                3,3,1,1           41       -FT,                 3,2,1,1
 5       -FF,-               3,3,1,3           42       -FT,-                3,2,1,3
 6       -FF,-T              XXX y-con.        43       -FT,-F               XXX y-con.
 7   +   -FF,-F              3,3,1,2           44   +   -FT,-T               3,2,1,1
 8       -FF,-FF             3,3,3,2           45       -FT,-TF              3,2,3,1
 9       -FF,-FF,            3,3,3,1           46       -FT,-TF,             XXX satis.
10       -FF,-FF,-           3,3,3,3           47 (45)  -FT,-TT              3,2,2,2
11       -FF,-FF,-T          XXX y-con.        48       -FT,-TT,             3,2,2,1
12   +   -FF,-FF,-F          3,3,3,2           49       -FT,-TT,-            3,2,2,3
13       -FF,-FF,-F-         3,3,3,2           50       -FT,-TT,-F           XXX y-con.
14       -FF,-FF,-F-T        XXX z-con.        51   +   -FT,-TT,-T           3,2,2,1
15   +   -FF,-FF,-F-F        3,3,3,2           52       -FT,-TT,-T-          3,2,2,3
16       -FF,-FF,-F-F,       3,3,3,1           53       -FT,-TT,-T-F         XXX z-con.
17       -FF,-FF,-F-F,T      XXX x-con.        54   +   -FT,-TT,-T-T         3,2,2,1
18   +   -FF,-FF,-F-F,F      3,3,3,1           55       -FT,-TT,-T-T,        XXX satis.
19       -FF,-FF,-F-F,FT     XXX y-con.        56 (2)   -T                   2,1,1,1
20   +   -FF,-FF,-F-F,FF     3,3,3,1           57       -TF                  2,3,1,1
21       -FF,-FF,-F-F,FFT    XXX z-con.        58       -TF,                 XXX satis.
22   +   -FF,-FF,-F-F,FFF    3,3,3,1           59 (57)  -TT                  2,2,1,2
23       -FF,-FF,-F-F,FFF    XXX satis. nf.    60       -TT,                 2,2,1,1
24 (8)   -FF,-FT             3,3,2,2           61       -TT,-                2,2,1,3
25       -FF,-FT,            3,3,2,1           62       -TT,-F               XXX y-con.
26       -FF,-FT,-           3,3,2,3           63   +   -TT,-T               2,2,1,1
27       -FF,-FT,-T          XXX y-con.        64       -TT,-TF              2,2,3,1
28   +   -FF,-FT,-F          3,3,2,2           65       -TT,-TF,             XXX satis.
29       -FF,-FT,-F-         3,3,2,2           66 (64)  -TT,-TT              2,2,2,2
30       -FF,-FT,-F-F        XXX z-con.        67       -TT,-TT,             2,2,2,1
31   +   -FF,-FT,-F-T        3,3,2,2           68       -TT,-TT,-            2,2,2,3
32       -FF,-FT,-F-T,       3,3,2,1           69       -TT,-TT,-F           XXX y-con.
33       -FF,-FT,-F-T,T      XXX x-con.        70   +   -TT,-TT,-T           2,2,2,1
34   +   -FF,-FT,-F-T,F      3,3,2,1           71       -TT,-TT,-T-          2,2,2,3
35       -FF,-FT,-F-T,FT     XXX y-con.        72       -TT,-TT,-T-F         XXX z-con.
36   +   -FF,-FT,-F-T,FF     3,3,2,1           73   +   -TT,-TT,-T-T         2,2,2,1
37       -FF,-FT,-F-T,FFF    XXX z-con.        74       -TT,-TT,-T-T,        XXX satis.

("-FF,-FT,-F-T,FFT")
```

Figure 2: The generator system for deciding the satisfiability of Boolean formulas in x, y, and z goes through these steps when applied to the encoded version of the (satisfiable) formula $(\overline{x} \vee y)\&(\overline{y} \vee z)\&(\overline{y} \vee \overline{z})\&(x \vee y \vee z)$. Though only one truth-assignment will satisfy the formula, it takes quite a bit of backtracking to find it. The notation used here for describing generator actions is similar to that used to describe recognizer actions in Figure ??, but a surface rather than a lexical string is the goal. A +-entry in the backtracking column indicates backtracking from an immediate failure in the preceding step, which does not require the full backtracking mechanism to be invoked.

THE EFFECT OF PRECOMPILATION

Since the above reductions require both the language description and the input string to vary with the SAT/3SAT problem to be solved, there arises the question of whether some computationally intensive form of precompilation could blunt the force of the reduction, paying a large compilation cost once and allowing Kimmo runtime for a fixed grammar to be uniformly fast thereafter. This section considers four aspects of the precompilation question.

First, the external description of a Kimmo automaton or lexicon is not the same as the form used at runtime. Instead, the external descriptions are converted to internal forms: RMACHINE and GMACHINE forms for automata, letter trees for lexicons (Gajek *et al.*, 1983). Hence the complexity implied by the reduction might actually apply to the construction of these internal forms; the complexity of the generation problem (for instance) might be concentrated in the construction of the "feasible-pair list" and the GMACHINE. This possibility can be disposed of by reformulating the reduction so that the formal problems and the construction specify machines in terms of their internal forms rather than their external descriptions. The GMACHINEs for the class of machines created in the construction have a regular structure, and it is easy to build them directly instead of building descriptions in external format. As traces of recognizer operation suggest, it is runtime processing that makes translated SAT problems difficult for a Kimmo system to solve.

Second, there is another kind of preprocessing that might be expected to help. It is possible to compile a set of Kimmo automata into a single large automaton (a BIGMACHINE) that will run faster than the original set. The system will usually run faster with one large automaton than with several small ones, since it has only one machine to step and the speed of stepping a machine is largely independent of its size. Since it can take exponential time to build the BIGMACHINE for a translated SAT problem, the reduction formally allows the possibility that BIGMACHINE precompilation could make runtime processing uniformly efficient. However, an expensive BIGMACHINE precompilation step does not help runtime processing enough to change the fundamental complexity of the algorithms. Recall that the main ingredients of Kimmo runtime complexity are the mechanical operation of the automata, the difficulty of finding the right lexical-surface correspondence, and the necessity of choosing among alternative lexicons. BIGMACHINE precompilation will speed up the mechanical operation of the automata, but it will not help in the difficult task of deciding which lexical-surface pair will be globally acceptable. Precompilation oils the machinery, but accomplishes no radical changes.

Third, BIGMACHINE precompilation also sheds light on another precompilation question. Though BIGMACHINE precompilation involves exponential blowup in the worst case (for example, with the SAT automata), in practice the size of the BIGMACHINE varies — thus naturally raising the question of what distinguishes the "explosive" sets of automata from those with more civilized behavior. It is sometimes suggested that the degree of *interaction among constraints* determines the amount of BIGMACHINE blowup. Since the computational difficulty of SAT problems results in large measure from their "global" character, the size of the BIGMACHINE for the SAT system comes as no surprise under the interaction theory. However, a slight change in the SAT automata demonstrates that BIGMACHINE size is not a good measure of interaction among constraints. Eliminate the satisfaction automaton from the generator system, leaving only the consistency automata for the variables. Then the system will not search for a *satisfying* truth-assignment, but merely for one that is internally consistent. This change entirely eliminates interactions among the automata; yet the BIGMACHINE must still be exponentially larger than the collection of individual automata, for its states must distinguish all the possible truth-assignments to the variables in order to enforce consistency. In fact, the lack of interactions can actually *increase* the size of the BIGMACHINE, since interactions constrain the set of reachable state-combinations.

Finally, it is worth considering whether the nondeterminism involved in constructing the lexical-surface correspondence can be removed by standard determinization techniques. Every nondeterministic finite-state machine has a deterministic counterpart that is equivalent in the weak sense that it accepts the same language; aren't Kimmo automata just ordinary finite-state machines operating over an alphabet that consists of *pairs* of ordinary characters? Ignoring subtleties associated with null characters, Kimmo automata can indeed be viewed in this way when they are used to verify or reject hypothesized pairs of lexical and surface strings. However, in this use they do not need determinizing, for each cell of an automaton description already lists just one state. In the cases of primary interest — generation and recognition — the machines are used as genuine *transducers* rather than acceptors.

The determinizing algorithms that apply to finite-state acceptors will not work on transducers, and in fact many finite-state transducers are not determinizable at all. Upon seeing the first occurrence of a variable in a SAT problem, a deterministic transducer cannot know in general whether to output T or F. It also cannot wait and output a truth-value later, since the variable might occur an unbounded number of times before there was sufficient evidence to assign the truth-value. A finite-state transducer would not be able in general to remember how many outputs had been deferred.

THE EFFECT OF NULLS

Since Kimmo systems can encode NP-complete problems, the general Kimmo generation and recognition problems are at least as hard as the difficult problems in NP. But could they be even harder? The answer depends on whether null characters are allowed. If nulls are completely forbidden, the problems are in NP, hence (given the previous result) NP-complete. If nulls are completely unrestricted, the problems are PSPACE-complete, thus probably even harder than the problems in NP. However, the full power of unrestricted null characters is not needed for linguistically relevant processing.

If null characters are disallowed, the generation problem for Kimmo systems can be solved quickly on a nondeterministic machine. Given a set of automata and a lexical string, the basic nondeterminism of the machine can be used to guess the lexical-surface correspondence, which the automata can then quickly verify. Since nulls are not permitted, the size of the guess cannot get out of hand; the lexical and surface strings will have the same length. The recognition problem can be solved in the same way except that the machine must also guess a path through the dictionary.

If null characters are completely unrestricted, the above argument fails; the lexical and surface strings may differ so radically in length that the lexical-surface correspondence cannot be proposed or verified in time polynomial in input length. The problem becomes PSPACE-complete — as hard as checking for a forced win from certain $N \times N$ Go configurations, for instance, and probably even harder than NP-complete problems (cf. Garey and Johnson, 1979:171ff). The proof involves showing that Kimmo systems with unrestricted nulls can easily be induced to work out, in the space between two input characters, a solution to the difficult Finite State Automata Intersection problem.

The PSPACE-completeness reduction shows that if two-level morphology is formally characterized in a way that leaves null characters completely unrestricted, it can be very hard for the recognizer to reconstruct the superficially null characters that may lexically intervene between two surface characters. However, unrestricted nulls surely are not needed for linguistically relevant Kimmo systems. Processing complexity can be reduced by any restriction that prevents the number of nulls between surface characters from getting too large. As a crude approximation to a reasonable constraint, the PSPACE-completeness reduction could be ruled out by forbidding entire lexicon entries from being deleted on the surface. A suitable restriction would make the general Kimmo recognition problems only NP-complete.

Both of the reductions remind us that problems involving finite-state machines can be hard. Determining membership in a finite-state language may be easy, but using finite-state machines for different tasks such as parsing or transduction can lead to problems that are computationally more difficult.

ACKNOWLEDGEMENTS

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research has been provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-80-C-0505. A version of this paper was presented to the Workshop on Finite-State Morphology, Center for the Study of Language and Information, Stanford University, July 29–30, 1985; the author is grateful to Lauri Karttunen for making that presentation possible. This research has benefited from guidance and commentary from Bob Berwick, and Bonnie Dorr and Eric Grimson have also helped improve the paper.

REFERENCES

Barton, E. (1985). "The Computational Complexity of Two-Level Morphology," A.I. Memo No. 856, M.I.T. Artificial Intelligence Laboratory, Cambridge, Mass.

Gajek, O., H. Beck, D. Elder, and G. Whittemore (1983). "LISP Implementation [of the KIMMO system]," *Texas Linguistic Forum* 22:187–202.

Garey, M., and D. Johnson (1979). *Computers and Intractability.* San Francisco: W. H. Freeman and Co.

Karttunen, L. (1983). "KIMMO: A Two-Level Morphological Analyzer," *Texas Linguistic Forum* 22:165–186.

Karttunen, L., and K. Wittenburg (1983). "A Two-Level Morphological Analysis of English," *Texas Linguistic Forum* 22:217–228.