Computing with Realizational Morphology

Lauri Karttunen

Palo Alto Research Center,
3333 Coyote Hill Road, Palo Alto, CA 94304
karttunen@parc.com
http://www.parc.xerox.com/istl/members/karttune

Abstract. The theory of realizational morphology presented by Stump in his influential book *Inflectional Morphology* (2001) describes the derivation of inflected surface forms from underlying lexical forms by means of ordered blocks of realization rules. The theory presents a rich formalism for expressing generalizations about phenomena commonly found in the morphological systems of natural languages.

This paper demonstrates that, in spite of the apparent complexity of Stump's formalism, the system as a whole is no more powerful than a collection of regular relations. Consequently, a Stump-style description of the morphology of a particular language such as Lingala or Bulgarian can be compiled into a finite-state transducer that maps the underlying lexical representations directly into the corresponding surface forms or forms, and vice versa, yielding a single lexical transducer.

For illustration we will present an explicit finite-state implementation of an analysis of Lingala based on Stump's description and other sources.

1 Introduction

Morphology is a domain of linguistics that studies the formation of words. It is traditional to distinguish between *surface forms* and their analyses, called *lemmas*. The lemma for a surface form such as the English word `bigger` typically consists of the traditional dictionary citation form of the word together with terms that convey the morphological properties of the particular form. For example, the lemma for `bigger` might be represented as `big+Adj+Comp` to indicate that `bigger` is the comparative form of the adjective `big`. Alternatively, the morphological properties might be encoded in terms of attribute-value pairs: `Cat:Adj, Degr:Comp`.

There are two challenges in modeling natural-language morphology:

1. **Morphotactics**
   Words are typically composed of smaller units: stems and affixes that must be combined in a certain order. Most languages build words by concatenation but some languages also exhibit non-concatenative processes such as interdigitation and reduplication [3].
2. **Morphological Alternations**
   The shape of the components that make up the word often depends on their

context. For example, the comparative degree of adjectives in English is expressed sometimes by *-er*, sometimes by *-r*, and the stem may also vary, as in *bigger*.

Computational linguists generally take it for granted that the relation between the surface forms of a language and their corresponding lemmas can be described as a *regular relation* [4]. If the relation is regular, it can be defined using the metalanguage of regular expressions; and, with a suitable compiler, the regular expression source code can be compiled into a finite-state transducer that implements the relation computationally. In the resulting transducer, each path (= sequence of states and arcs) from the initial state to a final state represents a mapping between a surface form and its lemma, also known as the *lexical form.*

Comprehensive lexical transducers have been created for a great number of languages including most of the European languages, Turkish, Arabic, Korean, and Japanese. They are commercially available through companies such as **Inxight**.

2 Realizational Morphology

The success of finite-state morphology has so far had very little impact within linguistics as an academic discipline. Practical issues that arise in the context of real-life applications such as completeness of coverage, physical size, and speed of applications are irrelevant from an academic morphologist's point of view. The main purpose of a morphologist writing to an audience of fellow linguists is to be convincing that his theory of word formation provides a more insightful and elegant account of this aspect of the human linguistic endowment than the competing theories and formalisms.

Gregory Stump's work on PARADIGM FUNCTION MORPHOLOGY [17] is a contribution to a theoretical tradition that goes back to Matthews [15], including works by authors such as Zwicky [18] and Anderson [1]. In these INFERENTIAL-REALIZATIONAL theories, as Stump characterizes them, the presence of affixes in the inflected form of a word arises from rules that express some morphological property or a combination of properties that are present in its lexical representation. The paradigm functions that generate all the possible forms of a word from all of its valid lexical representations are defined in terms of REALIZATION RULES, also called RULES OF EXPONENCE. These rules all have the general form shown in Table 1. The subscript n is an index for a particular block of rules; τ is the set of morphological features that are realized by the application of the rule; C is the lexical category that the rule is concerned with; X is a phonological

$$RR_{n,\tau,C}(< X, \sigma >) =_{def} < Y', \sigma >$$

Table 1. A Template for Realization Rules

input string that is either a part of the lexical representation or has been derived by realization rules that have already been applied, σ is a set of morphosyntactic properties (= features), and *Y'* is the resulting output string. The derivation of *Y'* may involve several steps. The first output of the rule, *Y*, is produced by adding some (possibly zero) affix to *X* and subjecting the result to any number of applicable morphophonological rules. An example of Stump's morphophonological rules is given in Table 2 (Stump p. 48). If no morphophonological rule is applicable, *Y'* consists of the input form *X* possibly with some added phonological material as in Table 3. This rule is in Block *B* and realizes the present tense of a verb as *e'* suffixed to the end of the stem.

If X=W[vowel$_1$] and Y = [vowel$_2$]Z, then the indicated [vowel$_1$] is absent from Y' and the indicated [vowel$_2$] is stressed in Y' iff [vowel$_1$] is stressed in Y.

Table 2. A Morphophonological Rule

$$RR_{B,Tns:pres,V}(< X, \sigma >) =_{def} < Xe', \sigma >$$

Table 3. A Simple Realization Rule

The rule blocks are applied in a given order. Within each block the rules are in principle unordered but ranked by PANINI'S PRINCIPLE: If two or more rules could apply, the most specific one takes precedence and the others do not apply.

Realization rules may also be specified in terms of other realization rules. Such rules Stump calls RULES OF REFERRAL. For example, if there is a rule that expresses some set of features by a given affix, another rule can be derived from it by modifying the feature set but retaining the affix. This is an important aspect of Stumps formalism because it gives an account of SYNCRETISM, that is, cases where the same affix appears in several places in a paradigm, possibly associated with different morphological properties. For example, in the case of the Lingala inflected form *bababetaki* 'they hit them', the same affix *ba* encodes both subject and object agreement features.

Lexical representations are of the general form $< Stem, Features >$ where *Stem* is a phonological representation and *Features* is some collection of attribute-value pairs. For example, the lexical representation of the Lingala inflected form *bambetaki* 'they hit me' might have a lexical representation shown in Table 4. The underlying stem of the verb is *bet* and its feature set consists of three attributes *Sub*, *Obj*, and *Tns* whose values encode the subject and object agreement features and tense.

$$< bet, Sub : [Per : 3, Num : Pl, Gen : 1, 2], Obj : [Per : 1, Num : Sg], Tns : Past : Hist >$$

Table 4. A Lexical Form

3 Formal and Computational Issues

Formal precision and unambiguous notation are clearly important for Stump but there is no discussion in the book about what the formal power of Realizational Morphology might be. It is obvious that the underlying lexical representations constitute a REGULAR LANGUAGE. Although the features may have set values, there is no recursion. All the examples of realization rules given by Stump seem to represent REGULAR RELATIONS. The same is clearly true of Stump's morphophonological rules that are essentially rewrite rules in the old Chomsky-Halle tradition [5]. As was first shown by Johnson [8] and subsequently by Kaplan and Kay [10], such rules represent regular relations. They can be written as regular expressions and compiled into transducers. If the lexicon itself is regular and if all the realization rules and morphophonological rules are regular, it is possible to compile the lexicon and the rules individually to finite-state automata and to compose them into a single transducer.

The possibility of a finite-state implementation of realizational morphology is not surprising to computational linguists. Lexical transducers have already been constructed for a great number of languages using other finite-state formalisms. However, it is not as evident that this can be done without losing the theoretical advantages of the framework. Notions such as Panini's Principle for resolving competition between competing rules and Stump's rules of referral have no obvious finite-state implementation. In the next section we will show that rules of exponence and rules of referral can be expressed simply and elegantly as regular expressions and compiled with the publicly available PARC/XRCE **xfst** tool [4].

A finite-state implementation of realizational morphology has a fundamental advantage over the implementations in systems such as DATR/KATR proposed by Finkel and Stump [7]. A system of realization rules expressed as a DATR theory can be used to generate an inflected surface form from its lexical description but such a system is not directly usable for recognition. In contrast, finite-state transducers are bidirectional. The same transducer can generate an inflected form from its underlying representation or analyze it into a lexical stem or stems and the associated feature bundles.

4 Application to Lingala

In this section we will show in detail how Realizational Morphology can be expressed in terms of the PARC/XRCE regular expression calculus as defined in Beesley and Karttunen [4]. The regular expressions given in this section constitute a script that can be directly compiled with the **xfst** tool. The data and the analysis of Lingala come from Chapter 5 in Stump's book, from a short monograph on Lingala by Meeuwis [16], and from Michael Gasser's course notes at `http://www.indiana.edu/~gasser/L103/hw10.html`. Lingala is a Bantu language spoken in Kinshasa and along the Congo river. Like other Bantu languages, Lingala has an elaborate system of noun classes or genders. The verbs contain affixes that mark agreement with the verb's subject's and object's person, number, and gender properties.

4.1 Features

We start with the auxiliary definitions in Table 5. The **xfst** tool interprets the command `define` as the instruction to bind the next symbol to the network compiled from the regular expression that follows. The stems of Lingala are assigned to the variable `Stem`. In this case, the set includes just the stem for the verb meaning 'hit'. The braces around `bet` indicate that the stem consists of a sequence of single-character symbols: *b e t*. The variable `L` is defined as the surface alphabet (the union of all lower-case letters). The vertical bar is the UNION operator. Following Stump, we ignore tones here.

```
    define Stems {bet} ;
    define L [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];
```

Table 5. Auxiliary Definitions

The next step consist of defining the feature set. To make comparisons easy, we closely follow Stump's notation altough it makes things more cumbersome than they would need to be otherwise. A feature consists of an attribute, such as `Per` for 'person' and a value such as 1. For the sake of legibility, we separate them with a colon (quoted for technical reasons). The definitions for Person, Number, Gender, and Tense are expressed in Table 6. The value of the variable `Person1` for example, is a sequence consisting of three symbols: `Per`, `:`, and `1`.

```
define Person1 [Per ":" 1];
define Person2 [Per ":" 2];
define Person3 [Per ":" 3];
define Number  [Num ":" [Sg | Pl] ];
define Gender3 [Gen ":" [1 "." 2 | 1a "." 2 | 3 "." 4 | 5 "." 6 |
                         7 "." 8 | 9a "." 10a | 10 | 11 "." 6 |
                         14 "." 6 | 15]];
define PastTense [Past ":" [Rec|Hist|MoreRem|MostRem]];
define PresTense [Pres ":" [Cont|Hab1|Hab2]];
define FutTense [Fut ":" [Immed|MostRem]];
```

Table 6. Features with Atomic Values

The next set of feature definitions in Table 7 makes reference to features already defined in Table 6. For example, the `Tense` feature consists of words such as *Tns:Past:Rec* and *Tns:Fut:Immed*. The definition of `Agreement` includes values such as *Per:1 Num:Pl* and *Per:3 Num:Sg Gen:5.6*. These values are to be interpreted as sets containing a number of features separated a space.

The definition of Agreement feature in Table 8 builds on the definition of the Agreement values in Table 7. There is a minor complication here. Gender 15 is not expressed at all as an object marker. As a subject marker it only exists in the singular. For this reason we have to eliminate some otherwise possible strings by subtraction. The dollar sign in Figure 8 is called the CONTAINS operator.

```
define Tense [Tns ":" [PastTense|PresTense|FutTense]];
define Agreement [[[Person1 | Person2] " " Number] |
                   [Person3 " " Number " " Gender3]];
```

Table 7. Features with Set Values

$15 denotes the language of strings that somewhere contain *15*. The & operator represents INTERSECTION. Thus [$Pl & $15] denotes strings that contain both *Pl* (= plural) and gender *15*.

```
define SubjAgr [Sub ":" Agreement]  - [$Pl & $15];
define ObjAgr  [Obj ":" Agreement] - $15 ;
define Agr     [Func ":" Agreement];
```

Table 8. Subject and Object Agreement Features

We now have nearly all the definitions we need to define the lexical forms of Lingala in the style of realizational morphology. The final definitions are in Table 9. The verb lexicon consists of forms such as <*bet,Sub:Per:3 Num:Sg Gen:14,6 Obj:Per:2 Num:Sg*>, <*bet,Sub:Per:3 Num:Pl Gen:5,6 Obj:Per:3 Num:Pl Gen:5,6*>, etc.,.in which the stem *bet* is paired with some valid combination of morphosyntactic features.

```
define Features  [SubjAgr " " ObjAgr " " Tense];
define VerbLex "<" Stems "," Features ">" ;
```

Table 9. Verb Lexicon

4.2 Realization Rules

The rules of exponence can be expressed in PARC/XRCE regular expression notation quite easily using the REPLACE operator ->. We will need two types of rules. Certain rules introduce specific subject or object markers, others introduce markers for both syntactic functions. Table 10 contains the specific rules.

Each rule inserts (= rewrites the empty string as) a particular affix in the beginning of the form derived so far, that is, immediately after the initial < bracket. For example, rule R302 inserts *ko* to the beginning of the stem if the object agreement features of the stem include second person and singular. Thus the rule will apply to an underlying lexical form such as

<*bet,Sub:Per:1 Num:Sg Obj:Per:2 Num:Sg Gen:1.2 Tns:Past:Rec*>

changing *bet* to *kobet* and leaving everything else unmodified. Note that although the features occur in the underlying lexical form in a certain order, none of the rules refer to the order and would apply even if the order was changed, say, by reversing the subject and object agreement features. The naming of the rules indicates what block each rule belongs to. The rules in Block 1 (R101, R102, etc.) realize subject agreement markers; the rules in Block 3 (R301, R02, etc) mark object agreement. In Lingala verbs, the subject markers precede the object

```
define R101 [[. .] -> {na} || "<" _ [$[SubjAgr & $Person1 & $Sg]]] ;
define R102 [[. .] -> o    || "<" _ [$[SubjAgr & $Person2 & $Sg]]] ;
define R103 [[. .] -> a || "<" _ [$[SubjAgr & $Person3 & $Sg & $2]]];
define R104 [[. .] -> e  || "<" _ [$[SubjAgr & $Person3 & $Sg & $7]]];
define R105 [[. .] -> {ei} || "<" _[$[SubjAgr & $Person3 & $Sg & $15]]];
define R111 [[. .] -> {to} || "<" _ [$[SubjAgr & $Person1 & $Pl]]] ;

define R301 [[. .] -> n || "<" _ [$[ObjAgr & $Person1 & $Sg]]] ;
define R302 [[. .] -> {ko} || "<" _ [$[ObjAgr & $Person2 & $Sg]]] ;
define R303 [[. .] -> {mo} || "<" _ [$[ObjAgr & $Person3 & $Sg & $2]]];
define R304 [[. .] -> {ei}  || "<" _ [$[ObjAgr & $Person3 & $Sg & $7]]];
define R310 [[. .] -> {lo} || "<" _ [$[ObjAgr & $Person1 & $Pl]]] ;
```

Table 10. Specific Subject and Object Agreement Rules

markers, and both come before the stem. Because the rules are designed to build the verb forms "from inside out" starting with the stem, the rules in Block 3 have to apply before the rules in Block 1.

With the rules in Table 10 we can already produce some Lingala stems fully marked with subject and object agreement markers. Table 11 illustrates the process with the **xfst** program. The first command composes the input string *<bet,Sub:Per:1 Num:Sg Obj:Per:2 Num:Sg Tns:Past:Rec>* with Rule R302 and the result again with Rule R101. Because angle brackets, commas, and colons have a special meaning in regular expressions, we have to put them in double quotes. The separating space symbols, `" "`, also have to be quoted. The symbol `.o.` is the COMPOSITION operator. The result of the composition is a single transducer containing one path. The **xfst** command 'print upper-words' shows the original input, the 'print lower-words' shows the resulting output. In the finite-state world, the linguistic notion of "rule application" corresponds to the composition of an input string with one or more rules in a cascade. As we see in Table 11, the effect of the two rules of exponence is to change *bet*, to *nakobet*.

```
xfst[0]: regex "<" {bet} "," Sub ":" Per ":" 1 " " Num ":" Sg " "
                             Obj ":" Per ":" 2 " " Num ":" Sg " "
                             Tns ":" Past ":" Rec ">"
               .o.
               R302
               .o.
               R101;
1.5 Kb. 34 states, 33 arcs, 1 path.
fst[1]: print upper-words
<bet,Sub:Per:1 Num:Sg Obj:Per:2 Num:Sg Tns:Past:Rec>
fst[1]: print lower-words
<nakobet,Sub:Per:1 Num:Sg Obj:Per:2 Num:Sg Tns:Past:Rec>
```

Table 11. A Cascade of Compositions

This is almost what we want but it is evident that to produce actual surface forms on Lingala, we need to suppress the morphological features on the output side. For that we need the cleanup rule defined in Table 12. It eliminates everything that is not part of the surface alphabet defined in Table 5, that is, brackets, punctuation, spaces, numerals, and the multi-character symbols for attributes and values. The backslash in Table 12 is the TERM COMPLEMENT operator. The zero represents an epsilon. With `Cleanup` as the last rule of the cascade, the lower-side output string on the single path is reduced to *nakobet*. The upper-side string of the path still has its original form. That is, we now have a minimal lexical transducer that generates and analyzes the string *nakobet*.

```
define Cleanup \L -> 0;
```

Table 12. Elimination Rule for Non-Alphabetic Symbols

4.3 Rules of Referral

In addition to specific subject and object markers, Lingala contains many affixes that are used both for subjects and objects. To be faithful to the principles of Realizational Morphology, we will not define them directly but derive them from a common source by a rule of referral. Table 13 contains the common sources. Note that the rules in this table all contain the symbol `Agr` defined in Table 8. In other words, the rules do not identify themselves as Subject or Object rules. The features are assigned to the place holder attribute *Func*.

```
define RAgr1 [[. .] -> {mo} || "<" _ [$[Agr & $Person3 & $Sg & $4]]];
define RAgr2 [[. .] -> {li} || "<" _ [$[Agr & $Person3 & $Sg & $5]]];
define RAgr3 [[. .] -> e  || "<" _ [$[Agr & $Person3 & $Sg & $[9a"."10a]]]];
define RAgr4 [[. .] -> {lo} || "<" _ [$[Agr & $Person3 & $Sg & $[10|11]]]];
define RAgr5 [[. .] -> {bo} || "<" _ [$[Agr & $Person3 & $Sg & $14]]];

define RAgr6 [[. .] -> {bo} || "<" _ [$[Agr & $Person2 & $Pl]]] ;
define RAgr7 [[. .] -> {ba} || "<" _ [$[Agr & $Person3 & $Pl & $2]]];
define RAgr8 [[. .] -> {mi} || "<" _ [$[Agr & $Person3 & $Pl & $4]]];
define RAgr9 [[. .] -> {ma} || "<" _ [$[Agr & $Person3 & $Pl & $[5|6]]]];
define RAgr10 [[. .] -> {bi}  || "<" _ [$[Agr & $Person3 & $Pl & $7]]];
define RAgr11 [[. .] -> i  || "<" _ [$[Agr & $Person3 & $Pl & $[9a|10]]]];
```

Table 13. Shared Agreement Rules

The rules of referral in Table 14 all use a construct indicated by ‘ in the PARC/XRCE regular expression calculus that modifies a given transducer by systematically replacing all occurrences of a given symbol with some other symbol. Rule R106, for example, derives a subject agreement rule from Rule RAgr1 in Table 13. Rule R305 derives an object agreement rule from the same source.

```
define R106 ‘[RAgr1, Func, Sub];
define R107 ‘[RAgr2, Func, Sub];
define R108 ‘[RAgr3, Func, Sub];
define R109 ‘[RAgr4, Func, Sub];
define R110 ‘[RAgr5, Func, Sub];
define R112 ‘[RAgr6, Func, Sub];
define R113 ‘[RAgr7, Func, Sub];
define R114 ‘[RAgr8, Func, Sub];
define R115 ‘[RAgr9, Func, Sub];
define R116 ‘[RAgr10, Func, Sub];
define R117 ‘[RAgr11, Func, Sub];
define R305 ‘[RAgr1, Func, Obj];
define R306 ‘[RAgr2, Func, Obj];
define R307 ‘[RAgr3, Func, Obj];
define R308 ‘[RAgr4, Func, Obj];
define R309 ‘[RAgr5, Func, Obj];
define R311 ‘[RAgr6, Func, Obj];
define R312 ‘[RAgr7, Func, Obj];
define R313 ‘[RAgr8, Func, Obj];
define R314 ‘[RAgr9, Func, Obj];
define R315 ‘[RAgr10, Func, Obj];
define R316 ‘[RAgr11, Func, Obj];
```

Table 14. Rules of Referral

Alternatively, we could have chosen either the subject or the object agreement rule as the basic one and derived the other by a rule of referral that substitutes the attribute *Sub* for *Obj*, or vice versa. Either way, Rule RAgr1 gets used twice, once as an object agreement rule, once as a subject agreement rule. Whatever theoretical insight there is in insisting that it is the same rule that applies in both cases, this insight is faithfully captured by the implementation.

```
define R201 [[. .] -> {ko} || "<" _ [$[Fut":"Immed]]] ;
define R401 [[. .] -> {ak} || _ "," [$[Pres":"[Hab1 | Hab2]|
                                      Past":"[Hist | MostRem]]]];
define R402 [[. .] -> a    || _ "," [$[Pres":"Cont Fut":"Immed]]];
define R501 [[. .] -> i || _ "," [$[Fut":"MostRem Past":"[Rec|Hist]]]];
```

Table 15. Rules of Tense and Aspect Marking

The final set of rules yet to be discussed involves the realization of Tense and Aspect features. The rules are expressed in Figure 15. Unlike the agreement features that come before the stem, most tense features are realized as suffixes after the stem. Consequently, the context specification of the tense rules refer to the comma, the marker that separates the stem from the feature specification as the right context. The one exception is the immediate future tense that is marked both by a prefix and by a suffix.

All that remains to be done now is to define the cascade of compositions that maps each of the lexical forms to its proper surface realization, and vice versa. Figure 16 gives the explicit definition of Lingala verbs in Realizational Morphology. The text following the hash mark is a comment, not part of the definition.

```
define LingalaVerbs [
     VerbLex
     .o.
     R301 .o. R302 .o. R303 .o. R304 .o. R305 .o.   # singular object
     R306 .o. R307 .o. R308 .o. R309
     .o.
     R310 .o. R311 .o. R312 .o. R313 .o. R314 .o.   # plural object
     R315 .o. R316
     .o.
     R201                                           # future
     .o.
     R101 .o. R102 .o. R103 .o. R104 .o. R105 .o.   # singular subject
     R106 .o. R107 .o. R108 .o. R109 .o. R110
     .o.
     R111 .o. R112 .o. R113 .o. R114 .o. R115 .o.   # plural subject
     R116 .o. R117
     .o.
     R401 .o. R402 .o. R501                         # tense
     .o.
     Cleanup ] ;
```

Table 16. Definition of Lingala Verbal Morphology

Some examples to show how the system is working are given in Table 17 where we are exploring the contents of the lexical transducer resulting from the script in Table 16 and the preceding definitions in **xfst**. The command 'print random-upper 5' print five lexical strings at random; the command 'print random-lower 5' prints five random surface strings. The command 'apply up loibeta' prints out all the possible feature bundles associated with the surface form. The last command generates *loibeta* from one of its lexical interpretations.

5 Conclusion

What we have shown in this paper is that Stump's theory of realizational morphology is yet another incarnation of finite-state morphology, different in notation, but not in substance, from the technology that the successful commercial morphology applications are based on. Moving from Stump's notation to a more standard regular expression calculus does not incur any loss of simplicity or elegance. Rather the opposite.

Computational phonology and morphology have a curious non-relationship with "real" linguistics extending back to more than three decades. Time after

```
fst[1]: print random-upper 5
<bet,Sub:Per:3 Num:Sg Gen:5.6 Obj:Per:3 Num:Sg Gen:14.6 Tns:Pres:Cont>
<bet,Sub:Per:3 Num:Sg Gen:14.6 Obj:Per:3 Num:Pl Gen:11.6 Tns:Pres:Cont>
<bet,Sub:Per:3 Num:Sg Gen:1.2 Obj:Per:3 Num:Sg Gen:5.6 Tns:Pres:Hab1>
<bet,Sub:Per:3 Num:Pl Gen:7.8 Obj:Per:1 Num:Pl Tns:Fut:MostRem>
<bet,Sub:Per:3 Num:Sg Gen:15 Obj:Per:3 Num:Pl Gen:5.6 Tns:Past:MostRem>
xfst[1]: print random-lower 5
mamibeta
loibeta
ikolobeta
mimobetak
ekoeibeta
xfst[1]: apply up loibeta
<bet,Sub:Per:3 Num:Sg Gen:11.6 Obj:Per:3 Num:Pl Gen:9a.10a Tns:Pres:Cont>
<bet,Sub:Per:3 Num:Sg Gen:11.6 Obj:Per:3 Num:Pl Gen:10 Tns:Pres:Cont>
<bet,Sub:Per:3 Num:Sg Gen:10 Obj:Per:3 Num:Pl Gen:9a.10a Tns:Pres:Cont>
<bet,Sub:Per:3 Num:Sg Gen:10 Obj:Per:3 Num:Pl Gen:10 Tns:Pres:Cont>
xfst[1]: apply down
apply down> <bet,Sub:Per:1 Num:Sg Obj:Per:2 Num:Sg Tns:Past:Rec>
nakobeti
apply down> <bet,Sub:Per:3 Num:Pl Gen:7.8 Obj:Per:1 Num:Pl Tns:Fut:MostRem>
bilobeti
```

Table 17. Exploring Lingala Verbal Morphology

time, from Johnson [8] to Eisner [6], including Kaplan and Kay [9, 10], Koskenniemi [14], myself [11, 12], Beesley [2], Kiraz [13], and others, the computational knights have presented themselves at the Royal Court of Linguistics, rushed up to the Princess of Phonology and Morphology in great exitement to deliver the same message "Dear Princess. I have wonderful news for you: You are not like some of your NP-complete sisters. You are regular. You are rational. You are finite-state. Please marry me. Together we can do great things." And time after time, the put-down response from the Princess has been the same: "Not interested. You do not understand Theory. Go away you geek."

This constant rejection of the most suitable suitor is puzzling. The Princess must have a vested interest in making simple things appear more complicated than they really are. The good news that the computational knights are trying to deliver is unwelcome. The Princess prefers the pretense that phonology/morphology is a profoundly complicated subject, shrouded by theories.

If that is the right analysis of the situation, computational linguists should adopt a different strategy. Instead of being the eternal rejected suitor at the Royal Court, they should adopt the role of the innocent boy in the street shouting "The Princess has no clothes. The Princess has no clothes..." That is my conclusion.

Acknowledgements. I thank Louisa Sadler for stimulating discussions of Stump's work, and Jason Eisner, Ronald M. Kaplan, Kemal Oflazer and Annie Zaenen for many helpful suggestions and comments.

References

[1] Stephen R. Anderson. *A-Morphous Morphology.* Cambridge University Press, Cambridge, England, 1992.

[2] Kenneth R. Beesley. Arabic morphology using only finite-state operations. In Michael Rosner, editor, *Computational Approaches to Semitic Languages: Proceedings of the Workshop*, pages 50–57, Montréal, Québec, August 16 1998. Université de Montréal.

[3] Kenneth R. Beesley and Lauri Karttunen. Finite-state non-concatenative morphotactics. In *SIGPHON-200. Fifth Workshop of the ACL Special Interest Group in Computational Phonology.*, pages 1–12, Luxembourg, August 5-6 2000. Association for Computational Linguistics.

[4] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology.* CSLI Publications, Palo Alto, CA, 2003.

[5] Noam Chomsky and Morris Halle. *The Sound Pattern of English.* Harper and Row, New York, 1968.

[6] Jason Eisner. Phonological comprehension and the compilation of optimality theory. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 56–63, Washington, DC, July 2002. Association for Computational Linguistics.

[7] Raphael Finkel and Gregory Stump. Generating hebrew verb morphology by default inheritance hierarchies. In *Proceedings of the Workshop on Computational Approaches to Semitic Languages*, pages 9–18, Washington, DC., 2002. Association for Computational Linguistics.

[8] C. Douglas Johnson. *Formal Aspects of Phonological Description.* Mouton, The Hague, 1972.

[9] Ronald M. Kaplan and Martin Kay. Phonological rules and finite-state transducers. In *Linguistic Society of America Meeting Handbook, Fifty-Sixth Annual Meeting*, New York, December 27-30 1981. Abstract.

[10] Ronald M. Kaplan and Martin Kay. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378, 1994.

[11] Lauri Karttunen. Finite-state constraints. In John Goldsmith, editor, *The Last Phonological Rule.* University of Chicago Press, Chicago, Illinois., 1993.

[12] Lauri Karttunen. The proper treatment of optimality in computational phonology. In *FSMNLP'98. International Workshop on Finite-State Methods in Natural Language Processing*, Bilkent University, Ankara, Turkey, June 29 1998. cmp-lg/9804002.

[13] George Anton Kiraz. Multi-tiered nonlinear morphology using multitape finite automata: A case study on Syriac and Arabic. *Computational Linguistics*, 26(1):77–105, 2000.

[14] Kimmo Koskenniemi. Two-level morphology: A general computational model for word-form recognition and production. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki, 1983.

[15] P. H. Matthews. *Inflectional Morphology: a Theoretical Study Based on Aspects of Latin Verb Conjugation.* Cambridge University Press, Cambridge, England, 1972.

[16] Michael Meeuwis. *Lingala.* Lincom Europa, München, Germany, 1998.

[17] Gregory T. Stump. *Inflectional Morphology. A Theory of Paradigm Structure.* Cambridge University Press, Cambridge, England, 2001.

[18] Arnold Zwicky. How to describe inflection. In *Proceedings of the Eleventh Anuual Meeting of the Berkeley Linguistic Society*, pages 372–86, Berkeley, CA, 1985. Berkeley Linguistic Society.