EXTENDING KIMMO'S TWO-LEVEL MODEL OF MORPHOLOGY *

Anoop Sarkar
Centre for Development of Advanced Computing
Pune University Campus, Pune 411007, India
anoop@parcom.ernet.in

Abstract

This paper describes the problems faced while using Kimmo's two-level model to describe certain Indian languages such as Tamil and Hindi. The two-level model is shown to be descriptively inadequate to address these problems. A simple extension to the basic two-level model is introduced which allows conflicting phonological rules to coexist. The computational complexity of the extension is the same as Kimmo's two-level model.

INTRODUCTION

Kimmo Koskenniemi's two-level model (Koskenniemi, 1983, Koskenniemi, 1984) uses finite-state transducers to implement phonological rules. This paper presents the experience of attempting a two-level phonology for certain Indian languages; the problems faced in this attempt and their resolution. The languages we consider are Tamil and Hindi. For the languages considered we want to show that practical descriptions of their morphology can be achieved by a simple generalization of the two-level model. Although the basic two-level model has been generalized in this paper, the extensions do not affect the complexity or the basic tenets of the two-level model.

SOME PROBLEMS FOR THE TWO-LEVEL MODEL

The two-level model is descriptively adequate for most morphological processes occuring in Indian languages. However, there are some cases where the basic two-level fails to give an *adequate* description. One problem is caused by the large number of words imported from Sanskrit in languages such as Hindi, Tamil and Tibetan. The other problem occurs in Tamil where phonology disambiguates between different senses of a morpheme. The cases where these occur is common and productive. They cannot be considered as exceptional.

For example, in Tamil the verb `tulai` (to be similar) is derived from the Sanskrit base word `tula` (similarity). The past participle of `tulai` exhibits the following property. (`LR` and `SR` refer to the lexical and surface environments respectively).

(1) `LR: tulai+0ta`
`SR: tolai0tta`
(adj. who resembles [something])

In this example, the consonant insertion at the morpheme boundary is consistent with Tamil phonology, but the realization of `u` as `o` in the environment of `tu` follows a morphology that originates in Sanskrit and which causes inconsistency when used as a general rule in Tamil. The following example illustrates how regular Tamil phonology works.

(2) `LR: kudi+0ta`
`SR: kudi0tta`
(adj. drunk)

(3) `LR: tolai+0ta`
`SR: tolai0tta`
(adj. who has lost [something])

From examples (1) through (3) we see that the same environment gives differing surface realizations. Phonological rules formulated within the two-level model to describe this data have to be mutually exclusive. As all phonological rules are applied simultaneously, the two-level model can describe the above data only with the use of arbitrary diacritics in the lexical representation. The same problem occurs in Hindi. In Table 1 (6) and (7) follow regular Hindi phonology, while (4) and (5) which have descended from Sanskrit display the use of Sanskrit phonology. All these examples show that any model of this phonological behaviour will have to allow access for a certain class of words to the phonology of another language whose rules might conflict with its own.

*I would like to thank P. Ramanujan and R. Doctor for their help, and Dr. Darbari for his support.

	Nom. Sing.	*Ob. Sing.*	*Nom. Plu.*	*Ob. Plu.*
(4)	pita	pita	pita	pitao
(5)	data	data	data	datao
(6)	phita	phite	phite	phito
(7)	ladka	ladke	ladke	ladko

Table 1: Behaviour of certain Hindi words that use Sanskrit phonology

There is one other problem that comes up in Tamil where the phonology disambiguates between two senses of a stem. For instance, for the word padi which means either, 1. to read, or 2. to settle; differing phonological rules apply to the two senses of the word. If, as in (8) gemination is applied the continuous participial of padi means *reading*, whereas, if nasalized, in (9), it means *settling* (e.g. of dust).

(8) LR: padi+0tu+0kondu
SR: padi0ttu0kkondu
(reading)

(9) LR: padi+0tu+kondu
SR: padi0ntu0kondu
(settling)

The two-level model could be conceivably used to handle the cases given above by positing arbitrary lexical environments for classes of words that do not follow the regular phonology of the language, e.g. in (1) we could have the lexical representation as tUlai with rules transforming it to the surface form. To handle (8) and (9) we could have lexical forms padiI and padiY tagged with the appropriate sense and with duplicated phonological rules. But introducing artificial lexical representations has the disadvantage that two-level rules that assume the same lexical environment across classes of words have to be duplicated, leading to an inefficient set of rules. A more adequate method, which increases notational felicity without affecting the computational complexity of the two-level model is described in the next section.

EXTENDING THE TWO-LEVEL MODEL

The extended two-level model presented allows each lexical entity to choose a set of phonological rules that can be applied for its recognition and generation.

Consider the two level rules[1] that apply to example (1). Rule 1 transforms u to o in the proper environment while Rule 2 geminates t.[2]

R1a: u:o ⇐ ___CV* +:0 t:t
R1b: 0:t ⇒ [B | NAS]C +:0___t:t

where, C - consonants
V - vowels
B - voiced stops
NAS - nasals

We cannot allow the rule R1 to apply to (2) and so we need some method to restrict its application to a certain set (in this case all words like (1) borrowed from Sanskrit). To overcome this, each lexical entry is associated with a subset of two-level rules chosen from the complete set of possible rules. Each morpheme applies its respective subset in word recognition and generation.

Consider a fictional example—(11) below—to illustrate how the extended model works.

(11) LR: $\overbrace{\text{haX}}^{1} + \overbrace{\text{mel}}^{2} + \overbrace{\text{lek}}^{3}$
SR: hom 0 mel 0 0ek

R11a: a:o ⇐ C___X: (+:0)
R11b: X:{m,0} ⇒ a:___(+:0) {m,~m}
R11c: l:0 ⇒ l:l (+:0)___

R11a transforms a to o in the proper environment, R11b geminates m and R11c degeminates l.[3] Assume rule R11a that is applied to a in morpheme 1—haX—cannot be used in a general way without conflicts with the complete set of two-level rules applicable. To avoid conflict we assign a subset of two-level rules, say P1, to morpheme 1 which it applies between its morpheme boundaries. Morphemes 2 and 3 both apply rule subset P2 between their respective boundaries. For instance, P1 here will be the rule set {R11a, R11b, R11c} and P2 will be {R11b, R11c}. Note that we have to sup-

[1] The notations used are: * indicates zero or more instances of an element, parentheses are optional elements, ~ stands for negation and curly braces indicate sets of elements that match respectively. 0 stands for the null character in both the lexical and surface representations.

[2] The description presented here is simplified somewhat as the purpose of presenting it is illustrative rather than exhaustive.

[3] In rule R11b a: means lexical a can be realized as any surface character.

ply each morpheme enough rules within its subset to allow for the left-context and right-context of the rules that realize other surrounding morphemes. All the rules are still applied in parallel. At any time in the recognition or generation process there is still only one complete set of two-level rules being used. Any rule (finite state transducer) that fails and which does not belong to the subset claimed by a morpheme being realized is set back to the start state. This mechanism allows mutually conflicting phonological rules to co-exist in the two-level rulebase and allow them to apply in their appropriate environments.

For instance, if we have a lexical entry `laX` in addition to the morphemes introduced in (11), then we can have realizations such as (12) by adding R12 to the above rules.

(12) `LR: laX+mel+lek`
`SR: lim0mel00ek`

R12: `a:i` $\Leftarrow$ `C___X: (+:0)`

Thus `laX` uses a rule subset `P3` which consists of rules {R12, R11b, R11c}. Notice R12 and R11a are potentially in conflict with each other.

In the method detailed above we ignore certain rule failures by resetting it to its start state. Can this be justified within the two-level model? Each rule has a lexical to surface realization which it applies when it finds that the left context and the right context specified in the rule is satisfied. In the extended model, if a rule fails and it does not belong to the rule set associated with the current morpheme, then by resetting it to its start state we are assuming that the rule's left context has not yet begun. The left context of the rule can begin with the next character in the same morpheme. This property means that we can have conflicting rules that apply within the same word.

In practice it is better to use an equivalent method where a set of two-level rules that *cannot* apply between its boundaries is stored with a morpheme. If one or more of these rules fail and they belong to the set associated with that morpheme then the rule is simply reset to the start state else we try another path towards the analysis of the word.

The model presented handles both additive and mutually exclusive rules, whereas in a system in which a few morphs specify additional rules and inherit the rest, mutually exclusive rules have to be handled with the additional complexity of the defeasible inheritance of two-level rules.

It is easy to see that the extensions do not increase the computational complexity of the basic two-level model. We have one additional lexical tag per morpheme and one check for set membership at every failure of a rule.

CONCLUSION

We have shown that some examples from languages such as Tamil and Hindi cannot be effectively described under Kimmo's two-level model. An extension to the basic two-level model is discussed which allows morphemes to associate with them rule subsets which correspond to a certain phonology which gives the morpheme a valid description. The extension to Kimmo's two-level model gives us the following advantages:

- rules that conflict in surface realization can be used,
- it gives more descriptive power,
- the number of rules are reduced,
- no increase in computational complexity over Kimmo's two-level model.

We have implemented the extended two-level model using the standard method of representing phonological rules by deterministic finite state automata (Antworth, 1990, Karttunen, 1983) and using PATRICIA (Knuth, 1973) for the storage of lexical entries.

REFERENCES

Antworth, Evan L., 1990. *PC-KIMMO: a two-level processor for morphological analysis.* Occasional Publications in Academic Computing No. 16. Dallas, TX: Summer Institute of Linguistics.

Karttunen, Lauri, 1983. KIMMO: a general morphological processor. *Texas Linguistic Forum* **22**:163-186.

Knuth, Donald E., 1973. *The Art of Computer Programming. Vol. 3/Sorting and Searching.* Addison Wesley, Reading, MA.

Koskenniemi, Kimmo, 1983. A Two Level model for Morphological Analysis. In *Proc. 8th Int'l Joint Conf. of AI (IJCAI'83)*, Karlsruhe.

Koskenniemi, Kimmo, 1984. A General Computational Model for Word-Form Recognition and Production. In *Proc. 10th Int'l Conf. on Comp. Ling. (COLING'84)*, pp. 178-181, Stanford University.