Grammar design with multi-tape automata and composition

Mans Hulden
University of Colorado Boulder
mans.hulden@colorado.edu

Abstract

In this paper we show how traditional composition-based finite-state grammars can be augmented to preserve intermediate results in a chain of compositions. These intermediate strings can be very helpful for various tasks: enriching information while parsing or generating, providing accurate information for debugging purposes as well as offering explicit alignment information between morphemes and tags in morphological grammars. The implementation strategies discussed in the paper hinge on a representation of multi-tape automata as a single-tape automaton. A simple composition algorithm for such multi-tape automata is provided.

1 Introduction

Applications for morphological and phonological analysis using finite-state techniques tend to follow established design patterns based on the composition of transducers that encode morphotactics and morphophonological alternations. Not counting a few exceptions that concern nonconcatenative morphologies, this well-established approach is indeed quite successful and streamlined in the domain of morphophonology if the goal is to produce a monolithic transducer that provides mappings between lemmas/tags and actual surface forms of words.

Some types of grammatical information are difficult to include in such a design, however. In morphological modeling, one may want to recover the alignment of morphological tags to the actual morphemes; in phonological modeling, one may want to recover intermediate representations that show how a particular phonological alternation targets specific segments in a word, what order phonological alternations occurred in, and what they were conditioned on. The ability to do so would make finite-state devices more attractive for linguistic research, where computational methods could help streamline the work of lining up large amounts of data and testing hypothetical generalizations; it might therefore increase linguists' use of finite-state methods, whose potential has to date been underexploited in the linguistics literature (Karttunen, 2003).

In this paper, we argue that a multi-tape model constructed by composition of individual multi-tape lexicon or alternation transducers offers a simple framework that addresses the problem of intermediate forms, while at the same time retaining the straightforward design of morphology and morphophonology. Apart from expanding the expressive power of the grammar, the method also offers the grammar designer the option to re-convert the multi-tape grammar to a simple underlying-to-surface transducer, if desired—as may be the case if the multi-tape representation is only used for obtaining debugging information.

When drafting a morphological grammar, debugging the alternation rules and lexicon description becomes much less burdensome under the multi-tape model, since information about each step in the process of mapping from underlying to surface form is retained and is available for inspection.[1]

2 Traditional rewrite-rule grammars

A significant portion of morphological analysis tools are written with the design alluded to above: (1) a transducer that encodes morphotactics and tag sequences, and (2) a series of transducers that model morphophonological/orthographic alternation. The latter may be expressed as Sound Pattern of English-inspired 'rewrite rules' (Chomsky and Halle, 1968) or as two-level parallel constraints

[1] The code and the examples in this paper are available at http://foma.googlecode.com

tupalan-ɯ	papi-ɯ	pulpu-un	pulpu	kiʈikiʈi	muŋkumuŋku	**Underlying form**
tupalankɯ						/k/-epenthesis
	papiwɯ					/w/-epenthesis
		pulpun				Vowel Deletion
			pulpa	kiʈikiʈæ	muŋkumuŋka	Final Lowering
				kiʈikiʈ	muŋkumuŋk	Apocope
					muŋkumuŋ	Cluster Reduction
					muŋkumu	Non-apical Truncation
				kiʈikiɻ		Sonorantization
tupalankɯ	papiwɯ	pulpun	pulpa	kiʈikiɻ	muŋkumu	**Surface form**

Table 1: Interaction of multiple phonological processes in Lardil.

(Koskenniemi, 1983), the former being the arguably more popular choice at present. The result of composing the lexicon transducer and the morphophonological transducers is one monolithic transducer that directly performs the bidirectional mapping from underlying-to-surface forms (generation) and vice versa (parsing). The prevalence of this design is probably partly due to known algorithms (Kaplan and Kay, 1994; Kempe and Karttunen, 1996; Mohri and Sproat, 1996; Hulden, 2009a) or software tools designed around this paradigm (such as Xerox's *lexc/xfst/twol* (Beesley and Karttunen, 2003), *foma* (Hulden, 2009b), or *Kleene* (Beesley, 2012)). In the following, we shall assume the more common 'rewrite-rule' paradigm.

Table 1 illustrates this standard design using some example words from a grammar of Lardil—an example language often used to illustrate complex rule ordering and word-final phonology with rules that are sensitive to ordering. The original data stems from Hale (1973), and we follow analyses by Kenstowicz and Kisseberth (1979); Hayes (2011); Round (2011). Due to the rich interaction of word-final deletion rules, this is a widely used data set that has been a target of many analyses, all of which illustrate the difficulty of marshaling a complex set of phonological alternations. To explain the workings of the grammar, we show all the intermediate steps in mapping from lemma-and-inflection forms to actual surface realizations. In actuality, if modeled by transducer composition, all the intermediate forms are lost through the composition process, which is one of the shortcomings addressed below. That is, a final composite transducer simply provides mappings between parse and surface. For phonological analysis, possible grammar debugging, and perhaps language documentation purposes, it would be very desirable to be able to produce a rich representation such as the one in table 1 from either an underlying form (morphological information) or the surface form showing all the processes that the word undergoes step-by-step.

Under the standard composition model, there is no easy way to do this, save by applying an underlying form to each of the individual transducers representing the alternation rules in order, saving the results, and passing them on as input to the next transducer. However, in the inverse direction, such a strategy is not directly feasible, in addition to the fact that not composing the transducers partly defeats the purpose of using a finite-state model in the first place.

There is no principled reason, however, why the composition algorithm should destroy the intermediate representations if they are desired later. In other words, when creating a composite transducer modeling $x{:}z$ from transducers $x{:}y$ and $y{:}z$, one can in principle expand the composition algorithm to yield $x{:}y{:}z$ in some representation, retaining all the intermediate information.

3 Previous work

The importance of the preservation of intermediate results in composition has been noted and partly addressed in Kempe et al. (2004), among others. Our formulation below differs in representation and algorithms, and also in that it is intended to be simple and easily implementable without special algorithms for multi-tape automata, i.e. only using established algorithms for single-tape automata and transducers. We use the representation of Hulden (2009a) for multi-tape automata. In that work, conversion from transducers is not considered, and no composition algorithm is given, as the assumption is that multi-tape automata are constructed through intersections of constraints on co-

AB	Concatenation
A|B	Union
A*	Kleene Star
~A	Complement
?	Any symbol in alphabet
0	The empty string (epsilon)
A^k	k-ary concatenation
%	Escape symbol
[and]	Grouping brackets
A:B	Cross product
A/B	A ignoring intervening B
T.2	Output projection of T
A -> B	Rewrite A as B
|| C _ D	Context specifier
.#.	End or beginning of string
def F(X1,...,Xn)	definition of macro
def X	definition of language constant

Table 2: Regular expression notation in *xfst/foma*.

occurrence of symbols on the various tapes, akin to two-level grammars.

4 Notation

We assume familiarity with regular expression notation to construct automata and transducers. For ease of replication, we employ the Xerox regular expression notation (Beesley and Karttunen, 2003) to define and manipulate automata and transducers in this paper; the examples should be directly compilable with the *xfst* or *foma* tools. The formalism used is summarized in table 2. Multi-tape additions are implemented through a Python interface discussed in section 8.

5 A multi-tape encoding

In the following we will assume a relatively simple interleaving encoding of a multi-tape automaton and represent one as a single-tape automaton where the length of any accepted string is always an even multiple of the number of tapes. Informally, the automaton first encodes the first column of the legal contents of an n-tape multi-tape automaton, top-down, then the second column, etc. etc. Every symbol in position k in the single-tape representation corresponds to—in the case of n tapes—position $\lfloor k/n \rfloor$ on tape $(k \mod n)$. We assume a special representation for empty symbols (ϵ-symbols) in the single-tape model, and represent them with the symbol $\square$. A string of length $l \times n$ in the single-tape representation would correspond to the multi-tape representation as follows, where, in parentheses, the tape number is shown first, followed by the symbol position in the multi-tape representation.

T_0	(0,0)	(1,0)	...	(l,0)
		...		
T_{n-1}	(0,$n-1$)	(1,$n-1$)	...	($l, n-1$)
T_n	(0,n)	(1,n)	...	(l, n)

For example, if a single-tape representation contains in its language the string $abcde\square$, this corresponds to a valid configuration

$$\begin{bmatrix} a & d \\ b & e \\ c & \square \end{bmatrix}$$

seen from the multi-tape point-of-view (a 3-tape configuration); i.e. a multi-tape automaton that accepts the string ad as input, translates it into be, and then translates this into c (the $\square$-symbol representing the empty string).

5.1 Conversion from transducers

An existing transducer can evidently be converted to this multi-tape representation— that is, to a 2-tape representation—without much effort. To convert a transducer where transitions are encoded as symbol pairs, one simply expands each symbol pair $x{:}y$ to a two-symbol sequence $x\ y$ in the corresponding n-tape automaton. We call this operation "flattening". If the original transducer T maps a string $x_1 \dots x_n$ to $y_1 \dots y_n$ by a sequence of transitions with labels $((x_1, y_1), \dots, (x_n, y_n))$, then the automaton flatten(T) accepts a string $(x_1 y_1 \dots x_n y_n)$. In the result, ϵ-symbols are replaced with the $\square$-symbol. So-called UNKNOWN symbols—in the Xerox formalism, placeholders for future alphabet expansion in incremental construction of automata (Beesley and Karttunen, 2003), which we denote by ? in regular expressions and @ in automata—can be retained as is.

Conversion of transducers is particularly convenient since we can take advantage of existing algorithms for building complex transducers for NLP use. This includes replacement-rule transducers available in many toolkits, as well as lexicon transducers constructed through essentially right-linear grammars. Figure 1 shows a replacement rule compiled into a transducer, and the result of subsequently converting that transducer to a 2-tape automaton in the encoding used here.

6 Multi-tape composition

Interestingly, a multi-tape composition algorithm in this representation can be encoded entirely al-

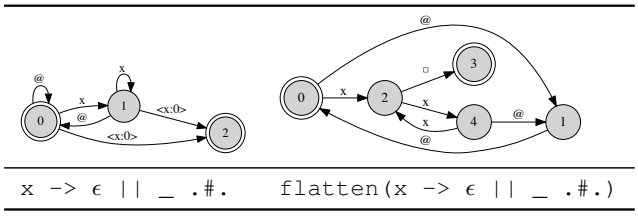

Figure 1: Illustration of a replacement-rule encoded as a transducer (left) and subsequently converted to a 2-tape automaton using the encoding presented here.

multi-tape composition

```
### Assume m-tape automaton A       ###
### and n-tape automaton B as input ###

def eInsertA [0:□^m [0:?^(n-1)-0:□^(n-1)]];
def PadA     [?^m 0:?^(n-1)];

def eInsertB [[0:?^(m-1) - 0:□^(m-1)] 0:□^n];
def PadB     [0:?^(m-1) ?^n];

def ExtendA [A .o. [eInsertA|PadA]*].2;
def ExtendB [B .o. [eInsertB|PadB]*].2;

def Sync ?^(m+n-1);

### Define composition filter ###
def X0 ?^(m-1) □^n ;
def XY [?^(m-1) - □^(m-1)] □ [?^(n-1) - □^(n-1)];
def 0Y □^m ?^(n-1) ;

def Filter ~[Sync* [0Y [X0|XY] | X0 [0Y|XY]] ?*];
def Composition  ExtendA & ExtendB & Filter ;
```

Figure 2: The multi-tape composition algorithm in Xerox notation. The variables m and n interspersed in the regular expressions denote the number of tapes assumed to be present in the two multi-tape automata to be composed.

gebraically, which is to say, as regular expressions. Given two multi-tape automata A and B encoded as above, each representing some specified number of tapes m and n, the core idea is to break down their composed representation as the following process, which returns an $m + n - 1$ tape representation of the composite.

1. Force automata A and B to be of the same number of tapes ($m + n - 1$) by alternatively inserting columns of empty ($\square$) symbols followed (in A) or preceded (in B) by arbitrary symbols, or retaining the original columns in A an B but inserting arbitrary symbols after each column (in A) or before each column (in B).

2. Call the new automata A_{extend} and B_{extend}: now, the result of intersecting the two $A_{\text{extend}} \cap B_{\text{extend}}$ (using standard automaton intersection) represents their composition $A \circ_{MT} B$, seen from a multi-tape point of view (with intermediate steps retained).

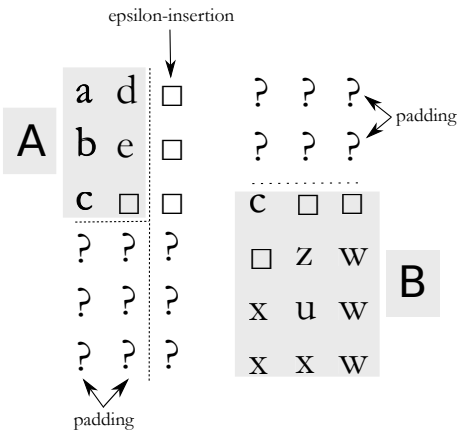

Figure 3: Illustration of multi-tape composition: the shaded areas show possible contents of the original multi-tape automata A and B, while the remaining areas show the result of insertions to coerce the automata to have the same dimensions and epsilon-behavior before intersection of A and B.

An illustration of the main logic behind the padding and column insertion mechanisms is given in figure 3.

6.1 Path filtering

A well known problem of standard composition algorithms for transducers also carries over to the multi-tape representation; this is the problem of producing multiple alternate paths in the resulting transducer when epsilon-symbols are present (ϵ-multiplicity). The cause of this is that there exist many equivalent paths that yield the same transduction: e.g. $a{:}\epsilon \circ \epsilon{:}b$ can be represented as $a{:}b$, a sequence $a{:}\epsilon\ \epsilon{:}b$, or a sequence $\epsilon{:}b\ a{:}\epsilon$; figure 4 illustrates different but equivalent outputs for the composition of two multi-tape automata. None of the multiple paths for describing a relation are incorrect, but the inconvenience of handling the possibility of multiple equivalent parses or generations motivates an attempt to provide unambiguous paths for each composition during the process itself. Furthermore, in a weighted automaton/transducer scenario—which we will not specifically deal with here—use of a non-idempotent semiring can yield incorrect results if multiple paths are not filtered out.

The common solution in the classical transducer domain is to either design a separate filter trans-

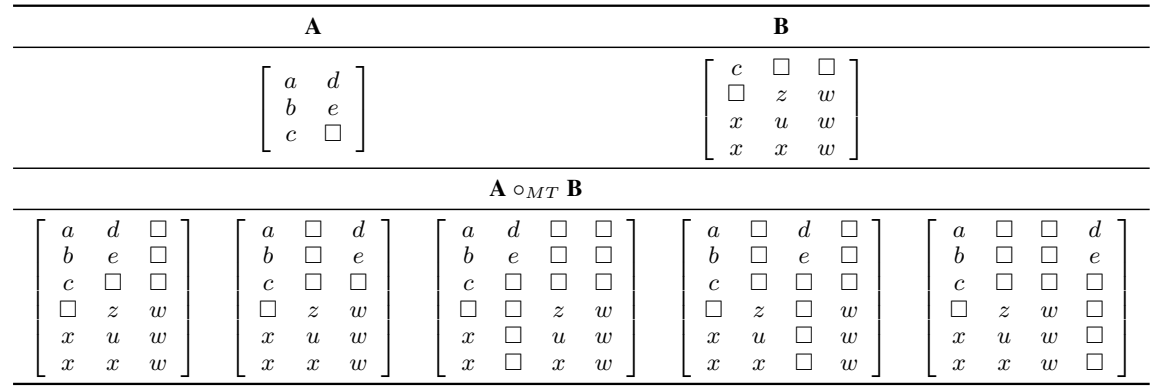

Figure 4: Various solutions to the composition of two multi-tape automata **A** and **B**, illustrating different alignments of epsilon-symbols. Here, what is shown is composition behavior with respect to two particular configurations in **A** and **B**, for the purposes of illustration. A subsequent filter, expressed as an automaton, removes all the solutions except the leftmost one.

ducer that serves to prefer some order of epsilon-interleaving (Mohri et al., 2002) or to incorporate this filter mechanism directly into the composition algorithm (Hulden, 2009a). In the multi-tape case, however, this filtering mechanism can be encoded entirely as a regular language filter which disallows certain interleavings of epsilon-symbols, in particular those where an x:□-transition (when automaton A has an epsilon on the last tape in some position) immediately follows or precedes a □:y-transition (when automaton B inserts a symbol on its first pair of tapes). This filter can then be intersected with the output of the earlier algorithm. As mentioned, this regular expression (`Filter`) can simply be intersected with the earlier result to remove redundant paths in the composition (shown in lines 16-20 in the algorithm). An implementation of the composition algorithm including filtering of multiple paths is given in figure 2 using the Xerox notation.

7 Composition in grammars

The composition algorithm is the only extension needed to retain all the intermediate information in an ordered rewrite-rule grammar. One can simply convert any individual transducers to a multi-tape representation and proceed with the composition, yielding a multi-tape representation of the same grammar. Parsing and generation of a string s can be performed by creating a padded multi-tape automaton where either the underlying representation or the surface representation is in place, with arbitrary symbols present on the other tapes. This multi-tape automaton can then be intersected with the grammar G, yielding a string representation of the set of legal parses or generations, with their intermediate representations intact.

parsing

```
def Parse(s, G) [s .o. [0:?^(n-1) ?]* ].2/□ & G;
```

generation

```
def Gen(s, G) [s .o. [? 0:?^(n-1)]* ].2/□ & G;
```

7.1 Adding intermediate information

It was hinted above that annotating the effect of various transducers is a very useful feature (as seen in Figure 1) for debugging or phonological analysis. Incorporating such information can be done separately from the multi-tape encoding; that is, one can first incorporate the desired information in a standard transducer and subsequently perform the conversion to a multi-tape representation. For morphophonological processes, it suffices to modify the transducers that encode the relevant replacement rules in such a way as to add information about each process. In most cases, this would only entail naming the process in question. Such an annotation mechanism can be encoded in a macro/function in the xfst/foma formalism:

```
def Mark(Rule, L, B)
   [Rule (B:0 ?:0 | B ?:L | 0:B 0:L)] &
   [\B* (B:0 ?:0) | [?-B]* [0:?|?:0|?:?-?]
   [[?-B]:[?-B]|?:0|0:?]* [0:B 0:L | B ?:L] ];
```

Here, each alternation rule transducer is augmented with the following behavior: at the end of the string a label (L) is appended, preceded by a boundary mark (B). If another label is already present (from a previous process in a chain of compositions), that label is simply replaced with the current label. If the rule doesn't fire (doesn't change anything for a particular input

string), nothing is appended and any existing labels are removed.

For example, a rule that deletes the latter of consecutive vowels could be encoded as follows:

```
def VD  Mark([ V -> 0 || V _ ], "Vowel Del", "#");
```

and would have the following effect on input words (a) **papiin** and (b) **papi**, respectively:

```
    (a)                    (b)
p a p i i n              p a p i
p a p i n   # Vowel Del  p a p i
```

8 Implementation

As the tools *xfst*, *foma*, and *OpenFST* have existing Python bindings that can be used to call the underlying algorithms, we have implemented the multi-tape automaton encoding as a separate Python-class (data type) `MTFSM`. This allows for a certain level of transparency in the bookkeeping needed. For example, the information about how many tapes are encoded in an FSM is auxiliary information that it is necessary to store during a composition process, since the multi-tape encoding does not inherently contain this information. A simple interface to the *xfst/foma* formalism allows for transparent conversion of transducers to 2-tape automata, which may then be incrementally composed to yield representations with multiple tapes:

```
>>> r1 = MTFSM("x -> y || c _ ")
>>> r2 = MTFSM("y -> z || _ d")
>>> composed = r1.compose(r2)
>>> print composed
    States: 35
    Transitions: 126
    Final states: 7
    Deterministic: 1
    Minimized: 1
    Numtapes: 3
>>> composed.generate("c x d")
c x d
c y d
c z d
```

8.1 An example

Returning now to the original Lardil example; annotating replacement rules with additional descriptive symbols to be inserted at the ends of strings every time a rule fires in combination with the multi-tape composition mechanism allows us to essentially automatically replicate the linguist-friendly representation given in table 1. Table 3 shows an example parse of the word undergoing the largest number of alternations given earlier, illustrating the output of the multi-tape generation where each intermediate result occupies a tape.

```
lardil.generate("muŋkumuŋku")
m u ŋ k u m u ŋ k u
m u ŋ k u m u ŋ k u
m u ŋ k u m u ŋ k u
m u ŋ k u m u ŋ k u
m u ŋ k u m u ŋ k u
m u ŋ k u m u ŋ k a   # Final Lowering
m u ŋ k u m u ŋ k    # Apocope
m u ŋ k u m u ŋ      # Cluster Reduction
m u ŋ k u m u        # Non-apical truncation
m u ŋ k u m u
m u ŋ k u m u
```

Table 3: Illustration of generating a word with the multi-tape encoding. This is the result of a standard composed transducer-grammar, converted to a multi-tape representation, with replacement rules also appending their own descriptions to an input string in case they fire.

9 Conclusion

We have proposed a general method for constructing finite-state grammars in the composed rewrite-rule tradition. The method in effect replaces the use of transducers with multi-tape automata. Existing algorithms for constructing transducers from rewrite-rule specifications can still be used if converted to multi-tape representations. The model itself assumes little machinery beyond the ability to compose the resulting multi-tape automata, but offers a way to produce rich representations of grammars constructed in this vein. If desired (for memory efficiency reasons) the resulting multi-tape automata can still be re-converted to transducers by eliminating the intermediate representations. This offers the possibility to only use the multi-tape representation for debugging purposes, if the final intent is to produce a simpler underlying-to-surface mapping, or vice versa.

The above techniques may be useful for other applications as well. In modeling historical sound changes, for example, 'debugging' problems similar to those in phonology and morphology tend to arise—much exacerbated by the fact that one is often dealing with multiple languages at the same time. Keeping track of hundreds of proposed sound laws together with their effect on lexical items across languages is a task that is well suited for the type of modeling presented in this paper.

Although the application focus of this paper has been more along the lines of modeling traditional non-probabilistic grammars, the methods presented above—the composition algorithm in particular—are also adaptable to weighted automata.

References

Beesley, K. R. (2012). Kleene, a free and open-source language for finite-state programming. In *10th International Workshop on Finite State Methods and Natural Language Processing (FSMNLP)*, pages 50–54.

Beesley, K. R. and Karttunen, L. (2003). *Finite State Morphology*. CSLI Publications, Stanford, CA.

Chomsky, N. and Halle, M. (1968). *The Sound Pattern of English*. Harper & Row.

Hale, K. (1973). Deep-surface canonical disparities in relation to analysis and change: An Australian example. *Current trends in linguistics*, 11:401–458.

Hayes, B. (2011). *Introductory Phonology*. John Wiley & Sons.

Hulden, M. (2009a). *Finite-state Machine Construction Methods and Algorithms for Phonology and Morphology*. PhD thesis, University of Arizona.

Hulden, M. (2009b). Foma: a finite-state compiler and library. In *Proceedings of the 12th conference of the European Chapter of the Association for Computational Linguistics,*, pages 29–32.

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

Karttunen, L. (2003). Computing with realizational morphology. In *Computational Linguistics and Intelligent Text Processing*, pages 203–214. Springer.

Kempe, A., Guingne, F., and Nicart, F. (2004). Algorithms for weighted multi-tape automata. *XRCE Research Report 2004/031*.

Kempe, A. and Karttunen, L. (1996). Parallel replacement in finite state calculus. In *Proceedings of the 34th annual meeting of the Association for Computational Linguistics*.

Kenstowicz, M. and Kisseberth, C. (1979). *Generative phonology*. Academic Press.

Koskenniemi, K. (1983). *Two-level morphology: A general computational model for word-form recognition and production*. Publication 11, University of Helsinki, Department of General Linguistics, Helsinki.

Mohri, M., Pereira, F., and Riley, M. (2002). Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88.

Mohri, M. and Sproat, R. (1996). An efficient compiler for weighted rewrite rules. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 231–238. Association for Computational Linguistics.

Round, E. (2011). Word final phonology in Lardil: Implications of an expanded data set. *Australian Journal of Linguistics*, 31(3):327–350.

10 Appendix: Lardil grammar

```
def Mark(Rule, Label) [Rule ("#":0 ?:0 | "#" ?:Label | 0:"#" 0:Label)] & [\"#"* ("#":0
?:0) | [?-"#"]* [0:?|?:0|?:?-?] [[?-"#"]:[?-"#"]|?:0|0:?]* [0:"#" 0:Label | "#" ?:Label]
];

def Vow  [a | æ | i | u];
def Cons [p | t | ʈ | ʧ | tʲ |k | m | n | ɳ | ŋ | ŋ | n̪ | nʲ| ɾ | l | w | ɻ | j ];
def Apical [t | ʈ | n | ɳ | ɾ | l | ɻ ];

def Nasal [m | n | ɳ | ŋ | ŋ | n̪ | nʲ];

def kEpenthesis     Mark([ [..] -> k || Nasal _ u ɻ ], "k-Epenthesis");
def wEpenthesis     Mark([ [..] -> w || i _ u ] , "w-Epenthesis");
def VowelDeletion   Mark([ Vow -> 0 || Vow _ ], "Vowel Deletion");
def FinalLowering   Mark([ i -> æ, u -> a || _ [.#.|"#"] ], "Final Lowering");
def Apocope         Mark([ Vow -> 0 || Vow Cons* Vow Cons* _ [.#.| "#"] ], "Apocope");
def ClusterRed      Mark([ Cons -> 0 || Cons _ [.#.|"#"] ], "Cluster Reduction");
def NonApicalDel    Mark([ [Cons - Apical] -> 0 || _ [.#.|"#"] ] , "Non-Apical
truncation");
def Sonorantization Mark([ ʈ -> ɻ || _ [.#.|"#"] ], "Sonorantization");
```