Rewrite rule grammars with multitape automata

Mans Hulden
Department of Linguistics
University of Colorado, USA

ABSTRACT

The majority of computational implementations of phonological and morphophonological alternations rely on composing together individual finite state transducers that represent sound changes. Standard composition algorithms do not maintain the intermediate representations between the ultimate input and output forms. These intermediate strings, however, can be very helpful for various tasks: enriching information (indispensable for models of historical linguistics), providing new avenues to debugging complex grammars, and offering explicit alignment information between morphemes, sound segments, and tags. This paper describes a multitape automaton approach to creating full models of sequences of sound alternation that implement phonological and morphological grammars. A model and a practical implementation of multitape automata is provided together with a multitape composition algorithm tailored to the representation used in this paper. Practical use cases of the approach are illustrated through two common examples: a phonological example of a complex rewrite rule grammar where multiple rules interact and a diachronic example of modeling sound change over time.

Keywords: grammar design, multitape automata, morphology, phonology, finite-state phonology

1 INTRODUCTION

Finite-state transducer based phonological and morphological models tend to be built by the composition of individual transducers that encode morphotactics and morphophonological alternations (Beesley and Karttunen 2003). Apart from cases such as nonconcatenative morphologies where augmented techniques tend to be favored (Beesley and Karttunen 2000; Habash *et al.* 2005; Hulden 2009d; Kiraz 2001), this well-established approach is indeed quite successful and streamlined in the domain of morphophonology if the goal is to produce a single transducer that maps underlying forms (parses) to surface forms and vice versa.

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 occur in, and what they were conditioned on. This is particularly important in developing finite-state models of historical sound change, where it is imperative to retain intermediate alignment information so that the model may indicate what sound laws proto-segments are subject to and in what order changes occur. In some respect, the "intermediate representations" in diachronic derivations are more crucial to the linguist than their counterparts in synchronic models since in the latter case they are bound to a particular model of phonology. The ability to model such sequences 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, I show that a multitape model constructed by composition of individual multitape 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 multitape grammar to a simple underlying-to-surface transducer, if desired – as may be the case if the multitape representation is only used for obtaining debugging information. Indeed, debugging the alternation rules and lexicon description involved in drafting a morphological grammar becomes much less burdensome under the multitape model, since information about each step in the process of mapping from underlying to surface form is retained and is available for inspection.[1]

The methods described in this paper are implemented as a standalone library in Python. The library itself is built on top of the *foma* library (Hulden 2009b) which provides a backbone implementation of standard transducer algorithms. The implementation allows users to develop multitape grammars in standard regular expression and rewrite-rule notation, automatically and transparently converting the compiled transducers to multitape equivalents and performing multitape composition on the components. This enables a relatively linguist-friendly grammar design procedure that relies on well-known formalisms and offers the possibility of quick conversion of existing grammars into a multitape representation where word-forms can be parsed and generated with a rich intermediate structure.

This paper is structured as follows: first, some background on rewrite-rule grammars is presented, motivating the need for more richly structured representations; this is followed by a description of the multitape encoding with special focus on the composition algorithm for multitape automata; following this, a system for augmenting the multitape automata with extra annotation (such as rule names) is presented; two case studies are then provided to illustrate in concrete terms the possibilities of the multitape formalism.

2 TRADITIONAL REWRITE-RULE GRAMMARS

A significant portion of morphological analysis tools are written with the design described 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 (Koskenniemi 1983), the former being arguably the more popular choice at present due to simplicity of debugging complex rule interactions (Alegria *et al.* 2010). The result of composing the lexicon transducer and the mor-

[1] The code and the examples in this article are available at `https://fomafst.github.io/`.

Table 1: Interaction of multiple phonological processes in Lardil

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

phophonological 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 2009c) or software tools designed around this paradigm (such as *lexc/xfst/twol* by Xerox (Beesley and Karttunen 2003), *foma* (Hulden 2009b), or *Kleene* (Beesley 2012)). In the following, I shall assume the more common 'rewrite-rule' paradigm.

Table 1 illustrates this standard design using some example words from a grammar of Lardil (iso 639-3: lbz, a Pama-Nyungan language spoken on Mornington Island in Australia). This is an example language often used to illustrate complex rule ordering and word-final phonology with rules that are sensitive to ordering. The table is laid out in a manner often employed by phonologists to quickly give an overview of interacting processes. The original data stems from Hale (1973), and I follow analyses by Kenstowicz and Kisseberth (1979); Hayes (2011); Round (2011). Due to the rich interaction of word-final deletion rules, this is a commonly cited data set that has been a target of many analyses, all of which illustrate the difficulty of marshaling a complex set of phonological alternations. In the language, we find three independently motivated deletion rules (apocope, cluster reduction, non-apical truncation) which interact in complex ways, sometimes conspiring to elide multiple segments word-finally. The rules in question are shown here in traditional phonological notation:

apocope	$V \rightarrow \emptyset$ / $V\, C_0\, V\, C_0$ __ #
cluster reduction	$C \rightarrow \emptyset$ / C __ #
non-apical truncation	$C \rightarrow \emptyset$ / __ # (unless C = [-distr.])

To explain the workings of the grammar, the table shows all the intermediate steps in mapping from lemma-and-inflection forms to actual surface realizations. In actuality, however, 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 any of the columns shown 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. 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. As will be seen below, a combination of a multitape design together with a rule-decoration mechanism allows us to automatically produce rich analyses very much like the ones given in Table 1.

3 PREVIOUS WORK

Multitape automata in general have been proposed as viable models for morphology and phonology, particular when addressing nonconcatenative phenomena abundant in Semitic languages such as Arabic, Hebrew, and Syriac (Altantawy *et al.* 2010; Kay 1987; Habash *et al.*

2005; Habash and Rambow 2006; Hulden 2009d; Kiraz 2000, 2001). In these approaches, different phonological tiers are represented by different tapes in a multitape model. Most of these earlier models could in fact be called multitape transducer models, since they typically work akin to transducers, although with an extended symbol representation where instead of manipulating symbol pairs, as in the transducer case, transitions are labeled with n-tuples of symbols. Specialized algorithms are then used to handle this representation and to enforce symbol correspondences across tapes – Kiraz (2000), for example, works with a constraint formalism similar to that of two-level morphology (Koskenniemi 1983), extended to operate in a multitape transducer scenario.

By contrast, the current work assumes as a starting point that regularities across multiple levels of representation will be captured not by constraints across multiple tapes, but that adjacent tapes will be constrained by (morpho)phonological *rewrite rules*. To make this feasible, the compilation of rewrite rules must be extended to a multitape scenario, and a composition algorithm is required that is able to join multitape representations together, preserving intermediate information.

The importance of the preservation of intermediate results in composition has been noted and partly addressed in Kempe *et al.* (2004), among others. The formulation presented below differs from earlier work in both representation and algorithms, and also in that it is intended to be simple and easily implementable without special algorithms for multitape automata, i.e. using only established algorithms for single-tape automata and transducers. The same representation (without a composition design) has been used earlier for the construction of Arabic multitape grammars (Hulden 2009a). In that work, conversion from transducers is not considered, and no composition algorithm is given, as the assumption is that multitape automata are constructed through intersections of constraints on co-occurrence of symbols on the various tapes, analogously to two-level grammars (Koskenniemi 1983). The multitape representation in this paper uses the encoding from (Hulden 2009d) and builds upon extensions to it given in Hulden (2015).

4 NOTATION

In discussing algorithmic aspects, familiarity with standard regular expression notation to construct automata and transducers is assumed. For regular languages or automata X and Y, the description below will make use of the operations union ($X \cup Y$), concatenation (XY), Kleene closure (X^*), Kleene plus (X^+), intersection ($X \cap Y$), complement ($\neg X$), and difference ($X - Y$). The n-ary concatenation of a language X with itself is denoted X^n. From two languages represented as automata, their string-wise cross-product and resulting regular relation (representable as a transducer) is denoted with $X:Y$. If X and Y are transducers, their composition is ($X \circ Y$). The input and output projections of a relation/transducer X are denoted domain(X) and range(X). Whenever a regular language (or automaton) X appears in a transducer context, it is assumed to represent the identity relation, i.e. a transducer that simply repeats the set of words accepted by X. In some algorithms subtraction is performed in a transducer context ($X - Y$); in such cases the subtraction refers to transducer path subtraction and not relation subtraction which regular relations are not closed under, i.e. the result represents the set of valid sequences of symbol pairs in X but not in Y. We use the special symbol ? to represent any single symbol.

When describing linguistic grammars, the well-known Xerox regular expression notation (Beesley and Karttunen 2003) is used in this paper to define and manipulate automata and transducers, rewrite rule transducers in particular; the examples should be directly compilable with the *foma* library. The formalism used is summarized in Table 2. Multitape additions are implemented through a Python interface discussed in Section 9.

5 A MULTITAPE ENCODING

In the implementations below, a multitape representation is assumed to be a simple single-tape automaton that either accepts or rejects a string s in the standard way. However, the strings in question are intended to represent valid computations of a multitape automaton where certain positions in s pertain to certain tapes. Which symbol in the linear string s belongs to which tape is modeled by a simple "interleaving" encoding where the length of any accepted string s is

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

`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

always an even multiple of the number of tapes in the multitape model it is intended to represent. Informally, the string first encodes the first column of the legal contents of an n-tape multitape automaton, top-down, then the second column, etc. etc. Every symbol in position k in the linear string representation corresponds to – in the case of n tapes – position $\lfloor k/n \rfloor$ on tape ($k \mod n$). A special representation for empty symbols (ϵ-symbols) in the single-tape model is assumed whereby they are represented with the symbol $\square$ – a so-called "hard zero". A string of length $l \times n$ in the single-tape string would correspond to the multitape representation as follows, where, in parentheses, the position within a tape is shown first, followed by the tape number in the multitape 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 is assumed to correspond to a valid configuration

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

seen from the multitape point-of-view (a 3-tape configuration); i.e. a multitape 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).

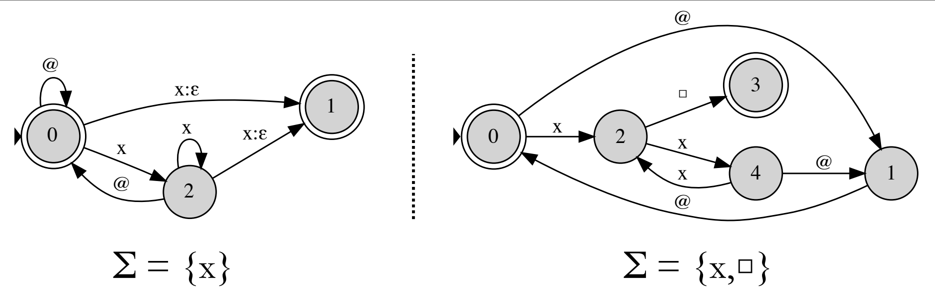
6 CONVERSION FROM TRANSDUCERS

It is evident that an existing transducer can be converted to this multitape representation – that is, to a 2-tape representation – without much effort. To convert a standard transducer, where transitions are encoded as symbol pairs, one simply expands each transition with a symbol pair $x : y$ to a two-symbol sequence xy in the corresponding n-tape automaton. This operation will be referred to as "flattening."[2] If the original transducer T maps a string $x_1 \ldots x_n$ to $y_1 \ldots y_n$ by a sequence of transitions with labels $((x_1, y_1), \ldots, (x_n, y_n))$, then the automaton flatten(T) accepts a string $(x_1 y_1 \ldots x_n y_n)$. In the result, ϵ-symbols are replaced with the $\square$-symbol. This $\square$-symbol is only used to mark the alignment of epsilons and need not be specified by the user in any way, as will be discussed below.

So-called UNKNOWN symbols – placeholders for future alphabet expansion in incremental construction of automata – are denoted by @. These are symbols that match any symbol outside the alphabet of an automaton. Note that this is different from the semantics of the ?-symbol in regular expressions which represent any single symbol at all with no reference to an alphabet (Beesley and Karttunen 2003).

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 that deletes x-symbols at the end of a string compiled into a transducer, and the result of subsequently converting that transducer to

[2] A symmetrical unflattening operation can also be defined.



`x -> ϵ || _ .#.`   `flatten(x -> ϵ || _ .#.)`

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

a standard automaton representing a 2-tape layout in the encoding used here. In other words, we can rely on existing algorithms to build phonological transducers, and only convert them to 2-tape automata before multitape composition.

7 MULTITAPE COMPOSITION

The overall usefulness of converting transducers to 2-tape automata, and then combining a number of individual such 2-tape automata by composition, is illustrated in Figure 2. By combining the individual 2-tape representations into a monolithic *n*-tape represen-

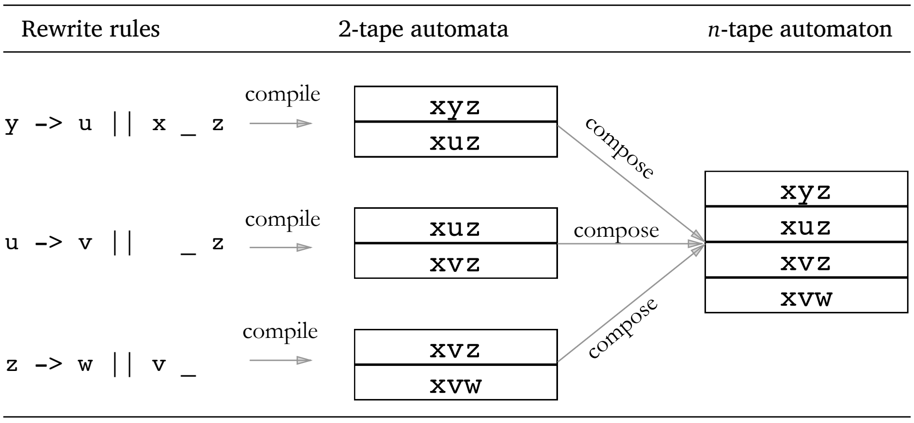


Figure 2: Workflow for converting rewrite rule specifications to transducers, then 2-tape automata, then *n*-tape automata

tation, all the intermediate representations which would normally be destroyed in a series of compositions can be preserved. As will be seen below, if such a strategy is augmented with the possibility of adding decoration and comment symbols to the individual tapes, very user-friendly grammars for parsing and generation can be developed.

Interestingly, a generic multitape composition algorithm in this representation can be encoded entirely algebraically, which is to say, as regular expressions. Given two multitape 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 a two-step process, which yields an $m+n-1$ tape representation of the composite. Informally, this multitape composition process for any m and n-tape automata in the representation at hand can be described as follows:

1. Force automata A and B to be of the same number of tapes ($m+n-1$) by alternatively inserting columns of empty (□) 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 multitape point of view (with intermediate steps retained).

An illustration of the main logic behind the padding and column insertion mechanisms is given in Figure 3. The exact algorithm is given in Algorithm 1.

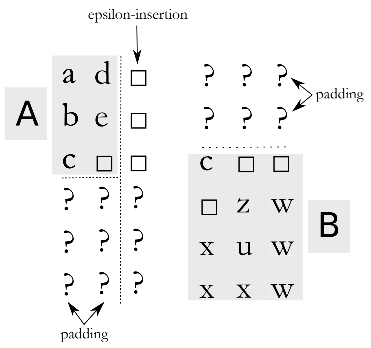


Figure 3:
Illustration of multitape composition: the shaded areas show possible contents of the original multitape automata A and B, while the remaining areas show the result of insertions to coerce the automata to have the same dimensions and epsilon-behavior before intersection of A and B

Algorithm 1: Multitape Composition

```
Input: $A =$ FSM with $m$ tapes, $B =$ FSM with $n$ tapes
1  eInsertA $\leftarrow (\epsilon:\Box)^m \big((\epsilon:?)^{n-1} - (\epsilon:\Box)^{n-1}\big)$
2  padA $\leftarrow ?^m\ (\epsilon:?)^{n-1}$
3  eInsertB $\leftarrow \big((\epsilon:?)^{m-1} - (\epsilon:\Box)^{m-1}\big)(\epsilon:\Box)^n$
4  padB $\leftarrow (\epsilon:?)^{m-1}\ ?^n$
5  ExtendA $\leftarrow$ range$\big(A \circ (\text{eInsertA} \cup \text{padA})^*\big)$
6  ExtendB $\leftarrow$ range$\big(B \circ (\text{eInsertB} \cup \text{padB})^*\big)$
7  Sync $\leftarrow ?^{m+n-1}$
8  X0 $\leftarrow ?^{m-1}\ \Box^n$
9  XY $\leftarrow \big(?^{m-1} - \Box^{m-1}\big)\ \Box\ \big(?^{n-1} - \Box^{n-1}\big)$
10 0Y $\leftarrow \Box^m\ ?^{n-1}$
11 Filter $\leftarrow \neg\big(\text{Sync}^*\ (0Y(X0 \cup XY) \cup X0(0Y \cup XY)\ ?^*\big)$
12 Result $\leftarrow$ ExtendA $\cap$ ExtendB $\cap$ Filter
```

7.1 *Path filtering*

A well known problem of standard composition algorithms for transducers also carries over to the multitape 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 multitape 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 transducer that serves to prefer some specific order of epsilon-interleaving (Mohri *et al.* 2002) or to incorporate this filter mechanism directly into the composition algorithm (Hulden 2009a). In the multitape case, however, this filtering mechanism can be encoded entirely as a regular language filter which disallows certain interleavings of epsilon-symbols in the string representation, in

A	**B**
$\begin{bmatrix} a & d \\ b & e \\ c & \square \end{bmatrix}$	$\begin{bmatrix} c & \square & \square \\ \square & z & w \\ x & u & w \\ x & x & w \end{bmatrix}$

A $\circ_{MT}$ **B**

$$\begin{bmatrix} a & d & \square \\ b & e & \square \\ c & \square & \square \\ \square & z & w \\ x & u & w \\ x & x & w \end{bmatrix} \quad \begin{bmatrix} a & \square & d \\ b & \square & e \\ c & \square & \square \\ \square & z & w \\ x & u & w \\ x & x & w \end{bmatrix} \quad \begin{bmatrix} a & d & \square & \square \\ b & e & \square & \square \\ c & \square & \square & \square \\ \square & \square & z & w \\ x & \square & u & w \\ x & \square & x & w \end{bmatrix}$$

$$\begin{bmatrix} a & \square & d & \square \\ b & \square & e & \square \\ c & \square & \square & \square \\ \square & z & \square & w \\ x & u & \square & w \\ x & x & \square & w \end{bmatrix} \quad \begin{bmatrix} a & \square & \square & d \\ b & \square & \square & e \\ c & \square & \square & \square \\ \square & z & w & \square \\ x & u & w & \square \\ x & x & w & \square \end{bmatrix}$$

Figure 4: Composition of automata **A** and **B**, illustrating different alignments of epsilon-symbols. This shows composition behavior with respect to two particular configurations in **A** and **B**. A subsequent filter, expressed as an automaton, removes all the solutions except the upper leftmost one

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 7-11 in the algorithm).

7.2 *Algorithm details*

The algorithm in 1 essentially reiterates the above, with a few details worth mentioning. In lines 1–4, constants that perform the insertion and padding are declared. Lines 5 and 6 create the transducers A_{extend} and B_{extend}. Lines 7–11 create the filter automaton which is independent of A and B, and the three-element intersection at line 12 yields the result of the final composition.

8 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 multitape representation and proceed with the composition, yielding a multitape representation of the same grammar. Parsing and generation of a string s can be performed by creating a padded multitape automaton where either the underlying representation or the surface representation is in place, with arbitrary symbols present on the other tapes. This multitape 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.

That is to say, if we have an n-tape automaton grammar G and want to parse a string s, we can convert the string to an automaton that accepts that string (ignoring possible intervening blanks $\square$), pad the automaton to match the number of tapes in G (making sure s is on the last tape), and then intersect with G. The padding operation may be performed by the standard method of composing with a transducer that inserts the right amount of arbitrary symbols, and then extracting the range of the transducer.

$$\text{Parse}(s,G) \overset{\text{def}}{=} \text{range}\big(s/\square \circ ((\epsilon{:}?)^{n-1}\ ?)^*\big) \cap G \tag{1}$$

Likewise, to generate, we may perform the same calculation with the padding done in such a manner that s is on the first tape:

$$\text{Generate}(s,G) \overset{\text{def}}{=} \text{range}\big(s/\square \circ ((?\ \epsilon{:}?)^{n-1})^*\big) \cap G \tag{2}$$

Again, these functions are intended to make the system transparent to the user so that no knowledge of the actual multitape representation is needed to design and apply grammars.

8.1 *Adding intermediate information*

It was hinted above that annotating the effect of various transducers is a very useful feature (as seen in Table 1) for debugging or phonological analysis. Incorporating such information can be done separately from the multitape encoding; that is, one can first incorporate the desired decorative information in a standard transducer and then perform the conversion to a multitape representation, retaining the decoration. 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 added separately to each rule transducer before converting it to a 2-tape representation.

8.2 *Decoration example*

In the examples below, each alternation rule transducer is augmented with a textual description of that rule. This allows us to pair up rule descriptions with rules, so that when parsing or generating with a multitape automaton, informative descriptions will appear for each rule in a chain of compositions. In essence, this allows for the inclusion of comments whenever a phonological alternation rule fires, similar to those given in Table 1.

For example, a rule that deletes the latter of consecutive vowels can be encoded as follows as a rule-description pair:

```
('V -> 0 || V _ ', 'Vowel Del')
```

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

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

making it clear to the user that this particular rule applies at that point in the derivation.

9 IMPLEMENTATION

As the *foma* tool has existing Python bindings that can be used to call the underlying standard algorithms for manipulating automata and transducers, providing an extension to that library becomes a matter of implementing the above algorithms. The multitape encoding has been implemented as a standard Python-class that (1) provides a multitape automaton data type `MTFSM` and (2) can perform composition together with rule decoration on arbitrary transducers. 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 multitape encoding does not inherently contain this information. The interface to the *foma* formalism allows for automatic conversion of transducers to 2-tape automata, which may then be incrementally composed to yield representations with multiple tapes. In effect, designing a complete grammar does not require the user to possess knowledge about or keep track of the underlying machinery, such as the number of tapes used, the padding performed, etc. Even the padding symbols – though helpful for debugging individual rules – can be omitted from the output as they are only used internally to produce a consistent alignment of different-length strings.

For example, to simply compose two rules, without any decoration, the user may enter arbitrary regular expressions (in this example rewrite rules) which automatically convert to two-tape representations that can be composed and inspected:

```
>>> r1 = MTFSM(”x -> y || c _ ”)
>>> r2 = MTFSM(”y -> z || _ d”)
>>> composed = r1 + r2

>>> print composed
    States: 35
    Transitions: 126
    Final states: 7
    Deterministic: 1
    Minimized: 1
    Numtapes: 3
```

Entire grammars can be compiled through a separate and more involved `mtgrammar` module. This module allows for the type of rule decoration described above, and provides for a method of composing the different multitape automata in order, as well parsing and generation functionality:

```
from mtgrammar import *

G = compilemt([(’b -> x || a _ c’, ’Rule 1’), (’x -> 0 || a _ c’, ’Rule 2’)])
printparses(’ac’, G, dir=’up’)
```

Here, two rewrite rules are compiled, converted automatically to multitape automata through the `compilemt` statement and composed in the order given. After this, the resulting 3-tape automaton is used to parse the word ac in the “upward” direction, that is, assuming that the string is on the output tape. This produces the three aligned outputs:

```
abc□□          axc□□          ac□□
axc#Rule 1     axc#Rule 1     ac#Rule 1
a□c#Rule 2     a□c#Rule 2     ac#Rule 2
a□c□□          a□c□□          ac□□
```

Here, we see that there are three ways the two phonological rules in question could produce the output ac – by starting from the underlying forms abc, axc, and ac, respectively. The blanks are automatically positioned in their correct positions without the user having to specify anything except the input string to be parsed and the direction of parsing (`up` = from surface form to underlying form, `down` = from underlying form to surface form).

9.1 *Illustrative example 1: phonology (Lardil)*

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 multitape composition mechanism allows us to essentially automatically replicate the linguist-friendly representation given in Table 1. The following snippet illustrates some key points in the design of such grammars:

```
from foma import *
from mtgrammar import *

# Definitions #
FST.define(u'{jilijili}|{kiʈikiʈi}|{muŋkumuŋku}', u'Stems')
FST.define(u'[a | æ | i | u]', 'Vow')
FST.define(u'[m | n | ɳ | ŋ | ŋ | n̪ | nʲ]', 'Nasal')
...
# Rules #
kEpenthesis =     (u'[..] -> k || Nasal _ u ɻ ', 'k-Epenthesis')
wEpenthesis =     (u'[..] -> w || i _ u' , 'w-Epenthesis')
...
G = compilemt((Lex, kEpenthesis, wEpenthesis, VowelDeletion, FinalLowering,
               Apocope, ClusterRed, NonApicalDel, Sonorantization))
```

That is, we may write grammars in much the same way as in established formalisms, defining regular expression constants such as `Vow` and `Nasal` which are later used in building more complex rewrite rules such as `k-Epenthesis` and `w-Epenthesis`, etc. These decorations are automatically added to the right end of each tier, as illustrated in two different parses below:

```
>>> printparses(u’muŋkumu’, G)

muŋkumuŋku[Uninflected]□□
muŋkumuŋku□#Lexicon Output
muŋkumuŋku□#k-Epenthesis
muŋkumuŋku□#w-Epenthesis
muŋkumuŋku□#Vowel Deletion
muŋkumuŋka□#Final Lowering
muŋkumuŋk□□#Apocope
muŋkumuŋ□□□#Cluster Reduction
muŋkumu□□□□#Non-Apical Deletion
muŋkumu□□□□#Sonorantization
muŋkumu□□□□□□
```

```
>>> printparses(u’putu’, G)

putuka[Uninflected]□□
putuka□#Lexicon Output
putuka□#k-Epenthesis
putuka□#w-Epenthesis
putuka□#Vowel Deletion
putuka□#Final Lowering
putuk□□#Apocope
putuk□□#Cluster Reduction
putu□□□#Non-Apical Deletion
putu□□□#Sonorantization
putu□□□□□
```

In the generation direction, the same procedure applies, and the library offers an up/down parameter to control for the direction of operation; a command `printparses(u’putuka[Uninflected]’, G, dir=’down’)` in the above would have produced the same output as the example on the right hand side.

9.2 *Illustrative example 2: Historical Linguistics (Proto-Indo-European)*

As alluded to above, another scenario where intermediate, possibly annotated strings provide important information is in the modeling of historical sound change by finite-state means. In the development of models of diachronic sound change, this provides the possibility of providing annotated parses from modern variants to proto-language forms given hypothesized chronological sound changes. The following parses show the behavior of an ordered set of rewrite rules in multitape form that model the path of sound changes from Proto-Indo-European (PIE) to German and Latin. The relevant rules are implemented as rewrite transducers as in the Lardil example above.

```
>>> printparses(u’pátēr’, Latin)

ph2tḗrs□□
ph2tḗrs#*PIE
ph2tḗr□#Szemerényi’s law
pa□tḗr□#*H > a between consonants
pá□tēr□#Proto-Italic stress
pá□tēr□□□
```

```
>>> printparses(u’fā́tɐr’, German)

ph2tḗrs□□
ph2tḗrs#*PIE
ph2tḗr□#Szemerényi’s law
pa□tḗr□#*H > a between consonants
fa□tḗr□#Grimm’s Law
fa□dḗr□#Verner’s Law
fá□dēr□#Stress Shift
fá□tēr□#High Germanic Consonant Shift
fā́tēr□#Lengthening
fā́tɐr□#Reduction
fā́tɐr□□□
```

Here, we see the parsing of the Latin form for the word "father", *pátēr* as well as the German form *fā́tʊr*, using two different grammars that share part of the rewrite rules (the early sound changes affecting both). Both correspond to the underlying, hypothesized PIE form *ph$_2$tḗrs*. The relevant sound changes in this grammar were modeled following Beekes (2011); Trask (1996). As opposed to synchronic phonological grammars, the chains of sound changes over long periods can grow quite extensive. For example, the German surface form is subject to a number of them: first, a sound change called *Szemerényi's law* deleting coda fricatives takes place, followed by a process of laryngeal vocalization,[3] Grimm's and Verner's Laws, a stress shift, as well as a number of processes that affect vowels. The multitape parse in this case illustrates the value of such a design in checking correctness of very complex sequences of sound changes. Such sequences could plausibly be generated in the chronological direction through non-finite-state means, but the direction of interest for the linguist is generally the inverse one – parsing from surface form to underlying form, which is what is calculated here.

More advanced usage scenarios can also be explored with the method through more complex intersections of individual tapes in multitape representations for different languages. For example, having postulated a sequence of sound changes that two modern languages have undergone from the proto-language, we can calculate the set of *possible* proto-forms for some modern cognates x and y in two languages. In the above parses of "father", only a single parse per cognate is given, since we have included the postulated proto-form in the grammar. There might, however, exist other plausible PIE-forms that fit the sequence of sound changes. For example, removing the proto-form from the grammar yields two plausible parses in the intersection of Latin and German, *patḗr* and *patḗrs*. Such techniques can be extended to a larger scale to support the endeavor of verifying consistency of postulated sound changes with the possibility of immediate feedback when minor changes are made in the various sound laws.

[3] Laryngeals are abstract segments proposed to have been present in Proto-Indo-European (De Saussure 1879) but later disappeared, leaving behind different vowel qualities and a compensatory lengthening. The laryngeals are commonly labeled $*h_1$, $*h_2$, and $*h_3$, and $*H$ is used as a cover symbol for all three.

10 CONCLUSION

This paper has presented a general, automatic method for extending finite-state grammars in the composed rewrite-rule tradition. The method in effect replaces the use of transducers with multitape automata, which are shown to have the capacity to provide rich parses and to support elaborate annotation of intermediate forms. Existing algorithms for constructing transducers from rewrite-rule specifications can still be used, once converted to multitape representations. We can also take advantage of specialized string-rewriting and constraint systems to handle syllabification (Hulden 2006), Semitic interdigitation (Beesley and Karttunen 2000), and, with some caution, unification features such as flag diacritics to model long-distance dependencies (Beesley 1998). Potentially, steps in candidate removal in Optimality Theoretic grammars could also be implemented by incorporating proposals to model such processes by finite-state composition (Karttunen 1998; Gerdemann and van Noord 2000; Gerdemann and Hulden 2012).

The model itself assumes little machinery beyond the ability to compose the resulting multitape automata, but offers a way to produce rich representations of grammars constructed in this vein. If desired (for memory efficiency reasons), the resulting multitape automata can still be re-converted to transducers by eliminating the intermediate representations. This offers the possibility to only use the multitape 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 applications outside standard designs of morphophonological grammars. 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.

APPENDIX A – LARDIL GRAMMAR

```
# -*- coding: utf-8 -*-
from foma import *
from mtgrammar import *

# Definitions needed for rules
FST.define(u'{papi}|{wiʈæ}|{ŋuku}|{wanka}|{kaɾikaɾi}|{jukaɾpa}|{putuka}|
          {jilijili}|{kiʈikiʈi}|{muŋkumuŋku}', u'Stems')
FST.define(u'[a | æ | i | u]', u'Vow')
FST.define(u'[p | t | ʈ | t̪ | tʲ |k | m | n | ɳ | ŋ | ŋ | n̪ | nʲ| ɾ | l |
              w | ɻ | j ]', u'Cons')
FST.define(u'[t | ʈ | n | ɳ | ɾ | l | ɻ ]', u'Apical')
FST.define(u'[m | n | ɳ | ŋ | ŋ | n̪ | nʲ]', u'Nasal')
FST.define(u'"#"|.#.', u'E') # Word edge

# Grammar Lexicon + Rules
Lex =             (u'Stems ["[Acc. Nonfuture]":{in} |
                            "[Acc. Future]":{uɻ} |
                            "[Uninflected]":0 ]', u'Lexicon Output')
kEpenthesis =     (u'[..] -> k || Nasal _ u ɻ ', u'k-Epenthesis')
wEpenthesis =     (u'[..] -> w || i _ u' , u'w-Epenthesis')
VowelDeletion =   (u'Vow -> 0 || Vow _ ', u'Vowel Deletion')
FinalLowering =   (u'i -> æ, u -> a || _ E', u'Final Lowering')
Apocope =         (u'Vow -> 0 || Vow Cons* Vow Cons* _ E', u'Apocope')
ClusterRed =      (u'Cons -> 0 || Cons _ E', u'Cluster Reduction')
NonApicalDel =    (u'Cons - Apical -> 0 || _ E', u'Non-Apical Deletion')
Sonorantization = (u'ʈ -> ɻ || _ E', u'Sonorantization')

Grammar = compilemt((Lex, kEpenthesis, wEpenthesis, VowelDeletion,
     FinalLowering, Apocope, ClusterRed, NonApicalDel, Sonorantization))

## Parse ##
mtgrammar.printparses(u'muŋkumu', Grammar)
mtgrammar.printparses(u'putu', Grammar)
mtgrammar.printparses(u'ŋukuɻ', Grammar)
```

REFERENCES

Iñaki ALEGRIA, Izaskun ETXEBERRIA, Mans HULDEN, and Montserrat MARITXALAR (2010), Porting Basque morphological grammars to foma, an open-source tool, 6062:105–113.

Mohamed ALTANTAWY, Nizar HABASH, Owen RAMBOW, and Ibrahim SALEH (2010), Morphological Analysis and Generation of Arabic Nouns: A Morphemic Functional Approach, in *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta.

Robert S. P. BEEKES (2011), *Comparative Indo-European Linguistics: an Introduction*, John Benjamins Publishing.

Kenneth R BEESLEY (1998), Constraining separated morphotactic dependencies in finite-state grammars, in *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pp. 118–127, Association for Computational Linguistics.

Kenneth R BEESLEY (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)*, pp. 50–54.

Kenneth R BEESLEY and Lauri KARTTUNEN (2000), Finite-state Non-Concatenative Morphotactics, in *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*.

Kenneth R. BEESLEY and Lauri KARTTUNEN (2003), *Finite State Morphology*, CSLI Publications, Stanford, CA.

Noam CHOMSKY and Morris HALLE (1968), *The Sound Pattern of English*, Harper & Row.

Ferdinand DE SAUSSURE (1879), *Mémoire sur le système primitif des voyelles dans les langues indo-européennes*, B.G. Teubner.

Dale GERDEMANN and Mans HULDEN (2012), Practical Finite State Optimality Theory, in *Proceedings of the 10th International Workshop on Finite State Methods and Natural Language Processing*, pp. 10–19, Association for Computational Linguistics, Donostia–San Sebastián.

Dale GERDEMANN and Gertjan VAN NOORD (2000), Approximation and exactness in finite state optimality theory, in *Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology*.

Nizar HABASH and Owen RAMBOW (2006), MAGEAD: A Morphological Analyzer and Generator for the Arabic Dialects, in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pp. 681–688, Association for Computational Linguistics, Sydney, Australia, doi:10.3115/1220175.1220261, `http://www.aclweb.org/anthology/P06-1086`.

Nizar HABASH, Owen RAMBOW, and George KIRAZ (2005), Morphological analysis and generation for Arabic dialects, in *Proceedings of the ACL Workshop on Computational Approaches to Semitic Languages*, pp. 17–24, Association for Computational Linguistics.

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

Bruce HAYES (2011), *Introductory Phonology*, John Wiley & Sons.

Mans HULDEN (2006), Finite-State Syllabification, *Lecture Notes in Artificial Intelligence*, 4002:86–96.

Mans HULDEN (2009a), *Finite-state Machine Construction Methods and Algorithms for Phonology and Morphology*, Ph.D. thesis, University of Arizona.

Mans HULDEN (2009b), Foma: a finite-state compiler and library, in *Proceedings of the 12th conference of the European Chapter of the Association for Computational Linguistics,*, pp. 29–32.

Mans HULDEN (2009c), Regular Expressions and Predicate Logic in Finite-State Language Processing, in Jakub PISKORSKI, Bruce WATSON, and Anssi YLI-JYRÄ, editors, *Finite-State Methods and Natural Language Processing—Post-proceedings of the 7th International Workshop FSMNLP 2008*, volume 191 of *Frontiers in Artificial Intelligence and Applications*, pp. 82–97, IOS Press.

Mans HULDEN (2009d), Revisiting multi-tape automata for Semitic morphological analysis and generation, *Proceedings of the EACL 2009 Workshop on Computational Approaches to Semitic Languages*, pp. 19–26.

Mans HULDEN (2015), Grammar design with multi-tape automata and composition, in *Proceedings of the The 12th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP)*, Association for Computational Linguistics.

Ronald M. KAPLAN and Martin KAY (1994), Regular models of phonological rule systems, *Computational Linguistics*, 20(3):331–378.

Lauri KARTTUNEN (1998), The proper treatment of optimality theory in computational phonology, in *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing (FSMNLP)*.

Lauri KARTTUNEN (2003), Computing with realizational morphology, in *Computational Linguistics and Intelligent Text Processing*, pp. 203–214, Springer.

Martin KAY (1987), Nonconcatenative Finite-State Morphology, in *Proceedings of EACL 1987*.

André KEMPE, Franck GUINGNE, and Florent NICART (2004), Algorithms for weighted multi-tape automata, *XRCE Research Report 2004/031*.

André KEMPE and Lauri KARTTUNEN (1996), Parallel replacement in finite state calculus, in *Proceedings of the 34th annual meeting of the Association for Computational Linguistics*.

Michael KENSTOWICZ and Charles KISSEBERTH (1979), *Generative phonology*, Academic Press.

George Anton KIRAZ (2000), Multitiered nonlinear morphology using multitape finite automata: a case study on Syriac and Arabic, *Computational Linguistics*, 26(1):77–105.

George Anton KIRAZ (2001), *Computational nonlinear morphology: with emphasis on Semitic languages*, Cambridge University Press, Cambridge.

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

Mehryar MOHRI, Fernando PEREIRA, and Michael RILEY (2002), Weighted finite-state transducers in speech recognition, *Computer Speech & Language*, 16(1):69–88.

Mehryar MOHRI and Richard SPROAT (1996), An efficient compiler for weighted rewrite rules, in *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pp. 231–238, Association for Computational Linguistics.

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

Robert Lawrence TRASK (1996), *Historical Linguistics*, Oxford University Press.