Proceedings of the Society for Computation in Linguistics

Volume 3 Article 28

2020

Multi-Input Strictly Local Functions for Templatic Morphology

Hossep Dolatian
Stony Brook University, hossep.dolatian@stonybrook.edu

Jonathan Rawski
Stony Brook University, jonathan.rawski@stonybrook.edu

Recommended Citation
Dolatian, Hossep and Rawski, Jonathan (2020) "Multi-Input Strictly Local Functions for Templatic Morphology," *Proceedings of the Society for Computation in Linguistics*: Vol. 3 , Article 28.
Available at: https://scholarworks.umass.edu/scil/vol3/iss1/28

Multi-Input Strictly Local Functions for Templatic Morphology

Hossep Dolatian and Jonathan Rawski
Dept. of Linguistics
Institute for Advanced Computational Science
Stony Brook University
{hossep.dolatian,jonathan.rawski}@stonybrook.edu

Abstract

This paper presents an automata-theoretic characterization of templatic morphology. We generalize the Input Strictly Local class of functions, which characterize a majority of concatenative morphology, to consider multiple lexical inputs. We show that strictly local asynchronous multi-tape transducers successfully capture this typology of nonconcatenative template filling. This characterization and restriction uniquely opens up representational issues in morphological computation.

1 Introduction

Recent work in mathematical phonology connects phonological mappings to subclasses of the regular functions (McNaughton and Papert, 1971; Rogers and Pullum, 2011; Rogers et al., 2013; Heinz and Lai, 2013; Chandlee, 2014). One of the simplest subclasses is the Input Strictly Local (ISL) functions, which take as input a single string and generate an output based on local information. Despite their reduced expressivity, ISL functions capture an overwhelming majority of phonological and morphological maps (Chandlee, 2017; Chandlee and Heinz, 2018). In addition, ISL functions are provably easier and faster to learn than full regular functions (Chandlee et al., 2015a).

In this paper, we generalize this notion of locality from the above single-input functions to functions which take *multiple* strings as input. Such functions are called *Multi-Input Strictly Local* (MISL). MISL functions are computed by deterministic asynchronous Multi-tape Finite State Transducers (MT-FSTs). Natural language has processes which are understood in terms of enriched multi-string input structures, i.e. autosegmental structure. We focus on root-and-pattern (RPM) morphology or template-filling in Semitic. This paper shows that when formalized as a multi-input function, most RPM patterns are MISL.

Semitic RPM has has often been computed using different types of of MT-FSTs. By showing that that the bulk of Semitic RPM can be computed with *only* MISL MT-FSTs, this can act as a stepping stone to determining the learnability of RPM. It likewise acts as a benchmark to examine the typology of attested and unattested RPM processes. Furthermore, by using multi-input functions with MT-FSTs instead single-input functions with FSTs, we can more iconically compute the fact that 1) RPM consists of separate tiers for roots, inflection, and templates, and that 2) this separation makes certain RPM processes be local.

Single-input functions are a special case of multi-input functions. With finite-state calculus, single-input functions correspond to rational functions when they are modeled with 1-way single-tape FSTs, and to regular functions when modeled by 2-way single-tape FSTs (Filiot and Reynier, 2016).[1] Multi-input functions correspond to the class of functions modeled by 1-way or 2-way MT-FSTs. Although there is work on the expressivity of MT-FSTs (Furia, 2012), little is known on multi-input *functions* and their algebra, expressivity, and hierarchy (Frougny and Sakarovitch, 1993). We show that a locally defined subclass, MISL, carves a substantial chunk of Semitic RPM.

2 Preliminaries

2.1 Preliminaries for single-input functions

Let $\rtimes$,$\ltimes$be the start and end boundaries respectively. Let Σ be a finite alphabet of symbols (excluding $\rtimes$,$\ltimes$). Let $\Sigma_{\rtimes} = \Sigma \cup \{\rtimes, \ltimes\}$. Let Σ^* the set of all strings over Σ. Let $|w|$ indicate the length of $w \in \Sigma^*$. For two strings w and v let wv be their

[1] By single-tape FST, we mean a two-tape FST with one input tape and one output tape. Note that the functions computed by 1-way FSTs are called 'regular functions' in American computer science. In this paper, we follow French conventions which call this class the 'rational functions' (Filiot and Reynier, 2016).

Proceedings of the Society for Computation in Linguistics (SCiL) 2020, pages 282-296.
New Orleans, Louisiana, January 2-5, 2020

concatenation, and for a set $L \subset \Sigma^*$ of strings and a string w, by wL we denote $\{wv | v \in L\}$. Let λ denote the empty string.

Given some string u and a natural number k, the *k-suffix* of u is the last k symbols of u: $\texttt{suff}(u, k) = v$ s.t. $|v| = k$ and $xv = u$ for some $x \in \Sigma^*$. For an alphabet Σ, the *k-factors* of Σ are the set of strings $w \in \Sigma*$ such that $|w| \leq k$.

Informally, a single-input function f is k-ISL if for all $u_1, u_2 \in \Sigma^*$, if $\texttt{suff}(u_1, k-1) = \texttt{suff}(u_2, k-1)$ then the two strings have the output extensions w.r.t f (Chandlee, 2014; Chandlee et al., 2015b). For any k-ISL function f over domain Σ^*, there exists a *canonical* deterministic single-tape finite-state transducer (1T-FST) M such that $|M| = f$ (meaning M computes f), and every state $q \in Q$ in M is labelled with one of the $k-1$ suffixes of Σ^*. Transitions are function tuples $\Delta : Q \times \Sigma \rightarrow Q \times \Gamma^*$. For a state $q \in Q$ and input symbol $a \in \Sigma$, $\delta(q, a) = (p, B)$ such that $B \in \Gamma^*$ and $p = \text{suff}(qa, k-1)$.

2.2 Preliminaries for multi-input functions

We introduce notation for functions which take multiple strings as input. To do so, we use tuples demarcated by brackets. In the formalization here, we only consider functions which produce one output string, not a tuple of output strings. But extending the formalization is trivial; such a function is illustrated in another paper of ours in the same volume.

A function f is an n-input function if it takes as input a tuple of n strings: $[w_1, \ldots, w_n]$, which we represent as $\vec{w}$, where each word w_i is made up of symbols from some alphabet Σ_i such that $w_i \in \Sigma_i^*$. Each alphabet Σ_i may be disjoint or intersecting, so two input strings w_i, w_j may be part of the same language Σ_i^*. These n alphabets form a tuple $\vec{\Sigma}$. Tuples can be concatenated: if $\vec{w} = [ab, c], \vec{x} = [d, ef]$, then $\vec{w}\vec{x} = [abd, cef]$.

To generalize the notion of suffixes into multiple strings, we define a tuple of n natural numbers as $\vec{k} = [k_1, \ldots, k_n]$. Given some tuple of n strings $\vec{w}$ and tuple of n numbers $\vec{k}$, *$\vec{k}$-suffix* of $\vec{w}$ is a tuple $\vec{v}$ of n strings v_i, made up of the last k_i symbols on w_i: $\texttt{suff}(\vec{w}, \vec{k}) = V$ s.t. $\vec{v} = [v_1, \ldots, v_n]$ and $|v_i| = k_i$ and $x_i v_i = w_i$ for $x_i \in \Sigma_i^*$. E.g. for $\vec{w}$=[abc,def] and $\vec{k} = [2, 1]$, $\texttt{suff}(\vec{w}, \vec{k}) = [bc, f]$. Given a tuple $\vec{k}$, the operation $\vec{k} - x$ subtracts x from each of k_i. E.g., for $\vec{k} = [2, 3, 6]$, $\vec{k} - 1 = [1, 2, 5]$. For a tuple of alphabets $\vec{\Sigma}$, the $\vec{k} - factors$ of $\vec{\Sigma}$ is the set of tuples $\vec{w} \in \vec{\Sigma}$ such that $|w_i| \leq k_i$. For example with

Let f be an n-input function defined over an $n-$tuple $\vec{w}$ of input strings $\vec{w} = [w_1, \ldots, w_n]$ taken from the tuple of n alphabets $\vec{\Sigma}$. As an *informal* and intuitive abstraction from ISL functions, f is Multi-Input Strictly Local (MISL) for $k = [k_1, \ldots, k_n]$ if the function operates over a bounded window of size k_i for w_i. Formally,

Definition 1: A function f is $\vec{k}$-MISL iff there exists a deterministic asynchronous Multi-tape FST such that i) $|M| = f$, and ii) the MT-FST is canonically $\vec{k}$-MISL

We explain $\vec{k}$-MISL Multi-tape FSTs in the next section.

Definition 1 is a automata-based definition of an MT-FST. We are currently working on finding a language-theoretic-based definition of an MISL function. Possible definitions for ISL functions, such as the use of tails or output extensions, cannot be easily extended to MISL functions. This is because are functions which have an MISL MT-FST, *but* the function has an infinite set of tails. We are currently investigating whether a monoidal definition of MISL functions is useful.

For an ISL function, it does not matter if the input string is read left-to-right or right-to-left. But for an MISL function, it does. A function may be left-to-right MISL but not right-to-left MISL. We leave out a proof but an illustration is given in another paper of ours in the same volume.

2.3 Multi-tape finite-state transducers

Multi-input functions can be modeled by multi-tape FSTs (MT-FST). An MT-FST is conceptually the same as single-tape FSTs, but over *multiple* input tapes (Rabin and Scott, 1959; Elgot and Mezei, 1965; Fischer, 1965; Fischer and Rosenberg, 1968; Furia, 2012). MT-FSAs and MT-FSTs are equivalent, and single-tape FSTs correspond to an MT-FSA with two tapes.

Informally, a MT-FST reads n multiple input strings as n input tapes, and it writes on a single output tape. Each of the n input strings is drawn from its own alphabet Σ_i. The output string is taken from the output alphabet Γ. For an input tuple of n strings $\vec{w} = [w_1, \ldots, w_n] = [\sigma_{1,1} \ldots \sigma_{1,|w_1|}, \ldots, \sigma_{n,1} \ldots \sigma_{n,|w_n|}]$, the initial configuration is that the MT-FST is in the initial state q_0, the read head. The FST begins at the first position of each of the n input tapes $\sigma_{i,1}$, and the writing head of the FST is positioned at the beginning of an empty output tape. After the FST reads the symbol under the read head, three things occur: 1) the state changes; 2) the FST writes some string; 3) the read head may advance to the right (+1) or stay put (0) on different tapes: either move on all tapes, no tapes, or some subset of the tapes.

This process repeats until the read head "falls off" the end of each input tape. If for some input $\vec{w}$, the MT-FST falls off the right edge of the n input tapes when the FST is in an accepting state after writing u on the output tape, we say the MT-FST transduces, transforms, or maps, $\vec{w}$ to u or $f_T\vec{w} = u$.[2] Otherwise, the MT-FST is undefined at $\vec{w}$. We illustrate MT-FSTs in §4.

Formally, a $n-$MT-FST for some natural number n is a 6-tuple $(Q, \vec{\Sigma_{\rtimes}}, \Gamma, q_0, F, \Delta)$ where:

- n is the number of input tapes
- Q is the set of states
- $\vec{\Sigma_{\rtimes}} = [\Sigma_{1\rtimes}, \ldots, \Sigma_{n\rtimes}]$ is a tuple of n input alphabets Σ_i which include the end boundaries $\Sigma_{i\rtimes}$
- Γ is the output alphabet
- $q_0 \in Q$ is the initial state
- $F \subset Q$ is the set of final states
- $\delta : Q \times \vec{\Sigma_{\rtimes}} \rightarrow Q \times \vec{D} \times \Gamma^*$ is the transition function where
  - $D = \{0, +1\}$ is the set of possible directions,[3]
  - $\vec{D} = [D^n]$ is an n-tuple of possible directions to take on each tape

The above definition can be generalized for MT-FSTs which use multiple output tapes. As parameters, an MT-FST can be deterministic or non-deterministic, synchronous or asynchronous. We only use *deterministic* MT-FSTs which are weaker than non-deterministic MT-FSTs. An MT-FST is synchronous if all the input tapes are advanced at the same time, otherwise it is asynchronous. We use asynchronous MT-FSTs which are more powerful than synchronous MT-FSTs. Synchronous MT-FSTs are equivalent to multi-track FSAs which are equivalent to single-tape FSAs, making them no more expressive than regular languages. For a survey of the properties of MT-FSAs and MT-FSTs, see Furia (2012).

A configuration c of a $n-$MT-FST M is an element of $(\vec{\Sigma_{\rtimes}}^* Q \vec{\Sigma_{\rtimes}}^* \times \Gamma^*)$, short for $([\Sigma_{1\rtimes}^* q \Sigma_{1\rtimes}^*, \ldots, \Sigma_{n\rtimes}^* q \Sigma_{n\rtimes}^*] \times \Gamma^*)$. The meaning of the configuration $c = ([w_1 q x_1, \ldots, w_n q x_n], u)$ is the following. The input to M is the tuple $\vec{w}\vec{x} = [w_1 x_1, \ldots, w_n x_n]$. The machine is currently in state q. The read head is on each of the n-input tapes on the first symbol of x_i (or has fallen off the right edge of the input tape if $x_i = \lambda$). u is currently written on the output tape.

Let the current configuration be $([w_1 q a_1 x_1, \ldots, w_n q a_n x_n], u)$ and let the current transition arc be $\delta(q, [a_1, \ldots, a_n]) = (r, \vec{D}, v)$. If $\vec{D} = [0^n]$, then the next configuration is $([w_1 r a_1 x_1, \ldots, w_n r a_n x_n], uv)$ in which case we write $([w_1 q a_1 x_1, \ldots, w_n q a_n x_n], u) \rightarrow ([w_1 r a_1 x_1, \ldots, w_n r a_n x_n], uv)$ (= none of the tapes are advanced) . If $\vec{D} = [+1^n]$, then the next configuration is $([w_1 a_1 r x_1, \ldots, w_n a_n r x_n], uv)$ in which case we write $([w_1 q a_1 x_1, \ldots, w_n q a_n x_n], u) \rightarrow ([w_1 a_1 r x_1, \ldots, w_n a_n r x_n], uv)$ (= all the tapes are advanced). Otherwise, the next configuration is $([w_i C_1 x_1 \ldots, w_n C_n x_n, \ldots], uv)$ where $C_i = r a_i$ if $D_i = 0$ and $C_i = a_i r$ if $D_i = +1$ in which case we write $([w_1 q a_1 x_1, \ldots, w_n q a_n x_n], u) \rightarrow ([w_i C_1 x_1 \ldots, w_n C x_n, \ldots], uv)$ (= a subset of the tapes are advanced).[4]

The transitive closure of $\rightarrow$ is denoted with $\rightarrow^+$. Thus, if $c \rightarrow^+ c'$ then there exists a finite sequence of configurations $c_1, c_2 \ldots, c_n$ with $n > 1$ such that $c = c_1 \rightarrow c_2 \rightarrow \ldots \rightarrow c_n = c'$.

As for the function that a MT-FST M computes, for each $n-$tuple $\vec{w} \in \vec{\Sigma}^*$ where $\vec{w} = [w_1, \ldots, w_n]$, $f_M(\vec{w}) = u \in \Gamma^*$ (where $f_M = |M|$) provided there exists $q_f \in F$ such that $([q_0 \rtimes w_1 \ltimes, \ldots, q_0 \rtimes w_n \ltimes], \lambda) \rightarrow^+ ([\rtimes w_1 \ltimes q_f, \ldots, \rtimes w_n \ltimes q_f], u)$. Otherwise, if the configuration is $([\rtimes w_1 \ltimes q, \ldots, \rtimes w_n \ltimes q], u)$ and $q \notin F$ then the transducer crashes and the transduction f_T is undefined on input $\vec{w}$. Note that if a MT-FST is deterministic, it follows that if $f_T(\vec{w})$ is defined then u is unique.

As explained in §2.2, we define a function as $\vec{k}$-MISL iff there exists a corresponding deterministic asynchronous $\vec{k}$-MISL Multi-tape FST.

Definition 2: A deterministic asynchronous MT-FST M with alphabet $\vec{\Sigma}$ is a canonical MT-

[2] If the MT-FST generates tuples instead of single strings, then the MT-ST maps $\vec{w}$ to $\vec{u}$.

[3] If the MT-FST reads from right to left, then it uses the -1 direction parameter

[4] Note that the interpretation of the third type of configuration subsumes the first two. We explicitly show the first two for illustrative reasons.

FST for an $\vec{k}$-MISL function f if the states of M are labelled with the $\vec{k}-1$ suffixes of $\vec{\Sigma}$.

In Definition 2, the restriction on state labels does not apply to the unique initial state and unique final state. In other words, except for the initial and final states q_0 and q_f, every state corresponds to a possible $\vec{k}-1$ *factor* of f

.

3 Root-and-pattern morphology in template filling

Semitic root-and-pattern morphology (RPM) involves segmenting a word into multiple discontinuous morphemes or morphs: a consonantal root **C**, inflectional vocalism **V**, and prosodic template **T**.[5] A partial paradigm of Standard Arabic verbs is in Table 1, amassed from McCarthy (1981). To illustrate, the verb *kutib* (Table 1a) is morphologically composed of a root **C**=*ktb*, vocalism **V**=*ui*, and template **T**=*CVCVC* which marks locations for consonants and vowels. Its autosegmental structure is provided in Table 1a.[6]

The bulk of theoretical and psycholinguistic results show that Semitic RPM *does* involve template-filling (Prunet, 2006; Aronoff, 2013; Kastner, 2016), but the formulation of templates is controversial (Ussishkin, 2011; Bat-El, 2011). One hypothesis is that the template is composed of CV slots (McCarthy, 1981). Alternatives are that the template is made of prosodic units like moras, syllables, and feet (McCarthy and Prince, 1990a,b), is derived from other templates via affixation (McCarthy, 1993), or is a set of optimized prosodic constraints (Tucker, 2010; Kastner, 2016; Zukoff, 2017). Alternatively, the job of the template is done by deriving words from other words via *overwriting* or changing the vowels and consonants (Ussishkin, 2005), e.g. *katab+ui→kutib*.

We take a theory-neutral position and focus on the *mathematical* function behind RPM. Mathematically, RPM is a 3-input function that takes as input a 3-tuple $\vec{w} = [w_1, w_2, w_3]$ where w_1 is the root **C**, w_2 is the vocalism **V**, w_3 is the template **T**. The input alphabets are $\Sigma_1 = \Sigma_C$ of consonants, $\Sigma_2 = \Sigma_V$ of vowels, and $\Sigma_3 = \Sigma_T$ of prosodic slots {C,V} and other elements (moras, affixes). Each alphabet includes the start and end boundaries $\rtimes, \ltimes$: $\Sigma_{i\rtimes} = \Sigma_i \cup \{\rtimes, \ltimes\}$. The output alphabet is the output segments.

Thus mathematically, many of the formalizaitons of templates are equivalent. Whether the template or **T**-string is made from CV units or moras is a *notational* difference (Kiraz, 2001) and does not affect locality. The use of derivational affixation is analogous to function composition; it does not affect locality and is discussed in §4.1.3,§4.2. For prosodic optimization, the function still needs to be well-defined over multiple inputs and this makes a template be implicitly present in the function. This is discussed in (Dolatian and Rawski, 2019). As for an overwriting approach, it still requires a mechanism for placing the new segments that references discontinuity. That is, the function *katab+ui→kutib* implicitly assumes that the vowels can be separated: *kVtVb+ui→kutib*. The fact that one of the inputs is a template with filled consonants *kVtVb* can be equally well broken down to a root and template *ktb+CVCVC*.

Computationally, different models have been used to compute the above *mathematical* function *behind* Semitic RPM: single-tape FSTs (Bird and Ellison, 1994; Beesley and Karttunen, 2000, 2003; Cohen-Sygal and Wintner, 2006; Roark and Sproat, 2007), synchronous MT-FSAs (Kiraz, 2000, 2001; Hulden, 2009), and non-deterministic asynchronous MT-FSTs (Kay, 1987; Wiebe, 1992). For a review, see Kiraz (2000, 92), Kiraz (2001, Ch4),and Wintner (2014, 47). We model RPM with asynchronous deterministic MT-FSTs in order to capture its locality properties, which we explain next.

4 Multi-Input Locality in Semitic

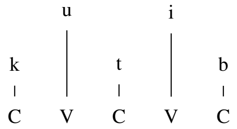
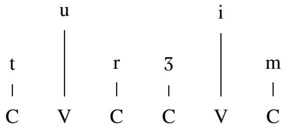
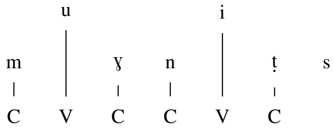
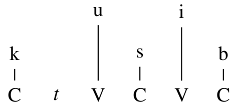
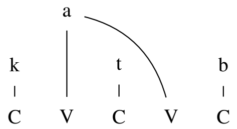
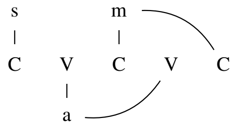
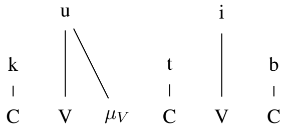
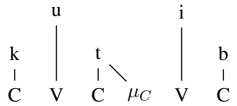
Mathematically, there is little discussion on the locality or non-locality of RPM. Chandlee (2017) shows that template-filling cannot be easily modeled with single-tape FSTs without sacrificing locality. Although not ISL, we show that the majority of RPM processes in Table 1 are MISL.

Arabic roots are generally at most 5 segments, vocalisms at most 2 segments, and the template is at most 12 slots (McCarthy, 1981). With this

[5] In Hebrew, some roots consists of consonants *and* vowels (Kastner, 2016). This difference is computationally trivial as long the template still treats Cs and Vs differently.

[6] We do not formalize RPM functions in broken plurals (Hammond, 1988; McCarthy and Prince, 1990b). Kiraz (2001, 106) formalizes it as a MT-FSA which use two inputs tapes: the singular and the vocalism. The singular tape can be annotated with prosodic information. We conjecture that broken plural formation is also MISL because there are no long-distance dependencies. We leave out a full formalization for space.

Table 1: Partial paradigm of Arabic root-and-pattern morphology with stable $\vec{k}$-values.

	Slot-filling pattern	Binyan	Gloss	Output	Root	Vowels	Template	k-value
a	1-to-1	Measure I Passive	*kutib*	'was written'	*ktb*	*ui*	*CVCVC* 	[1,1,1]
b	... four consonants	Measure QI Passive	*turʒim*	'was translated'	*trʒm*	*ui*	*CVCCVC* 	[1,1,1]
c	... with final deletion	Borrowed verb	*maɣnaṭ*	'be magnetized'	*mɣnṭs*	*ui*	*CVCCVC* 	[1,1,1]
d	... with pre-association	Measure VIII Passive	*k<t>usib*	'was gained'	*ksb*	*ui*	*CtVCVC* 	[1,1,1]
e	1-to-many... ... final spread of... ... vowels	Measure I Active	*katab*	'it wrote'	*ktb*	*a*	*CVCVC* 	[1,2,1]
f	... consonants	Measure I Active	*samam*	'he poisoned'	*sm*	*a*	*CVCVC* 	[2,1,1]
g	... medial spread of... ... (long) vowels	Measure III Passive	*kuutib*	'be corresponded'	*ktb*	*ui*	$CV\mu_V CVC$ 	[1,2,1]
h	... (geminate) consonants	Measure II Passive	*kuttib*	'be caused to write'	*ktb*	*ui*	$CVC\mu_C VC$ 	[2,1,1]

bound, RPM is reducible to modeling a function over a finite domain and range, i.e., a *finite* list of input-output pairs. Throughout this section, we abstract away from this. Our functions assume that there is no bound on the size of the root **C**, vocalism **V**, or template **T**. This allows us to treat RPM as a function over an infinitely sized domain. Doing so allows us to better capture the underlying function's generative capacity (Savitch, 1993). See (Dolatian and Rawski, 2019) for details on the role of infinity in computing Semitic RPM.

4.1 1-to-1 slot-filling

4.1.1 Simple 1-to-1 slot-filling

For *kutib* (Table 1a), RPM shows 1-to-1 slot-filling, meaning the e association of segments on any two strings is 1-to-1. The number of vowels in the vocalism **V** match the number of *V* slots in the template **T**. The same applies for the number of consonants in the root **C** and the *C* slots in **T**.

1-to-1 slot-filling is [1,1,1]-MISL or MISL for $\vec{k} = [1, 1, 1]$. The function is modeled by the deterministic asynchronous MT-FST in Figure 1 using three input tapes: **C**-tape, **V**-tape, and **T**-tape. The transition arcs in the MT-FST in Figure are in shorthand. In a transition arc like $[c, \Sigma_{\rtimes}, C]$: $[+1, 0, +1]$: c, lower case letters are interpreted as variables. A derivation is provided in Table 2. Each row keeps track of the:

1. current state
2. location of the read heads on the 3 input tapes
3. transition arc used on each 3 input tapes
4. outputted symbol
5. current output string

We use a deterministic asynchronous MT-FST because it can *iconically* model MISL functions, while a synchronous MT-FST cannot without sacrificing locality. The reason is because synchronous MT-FSTs are equivalent to single-tape FSAs, thus making RPM computed non-locally. To illustrate, Figure 2 is the derivation for *kutib* using a synchronous 4-tape MT-FSA. To avoid asynchrony, the 3 'input' tapes are aligned with the corresponding symbols on the 'output' tape by using the special symbol □ as a padding symbol.

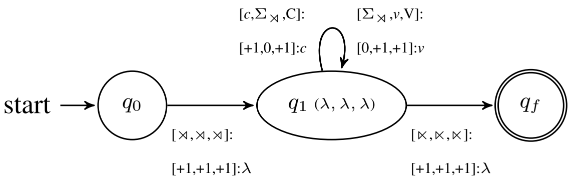


Figure 1: MT-FST for 1-to-1 slot-filling.

Input Tapes	**C**:	k	□	t	□	b
	V:	□	u	□	i	□
	T:	C	V	C	V	C
Output Tape:		k	u	t	i	b

Figure 2: Alignment of *kutib* with a synchronous MT-FSA (cf. Kiraz, 2001; Hulden, 2009).

To understand why the function is [1,1,1]-MISL, consider its MT-FST in Figure 1. Besides the initial and final state, there is only one state q_1. q_1 keeps track of the last $\vec{k}-1$ suffix on each of the three input-strings. Because $\vec{k}-1 = [1, 1, 1]-1 = [0, 0, 0]$, the state q_1 does not keep track of any previous input-symbol seen. When deciding on what to output and which state to go to, only the current input symbols on the 3 tapes were needed.

4.1.2 1-to-1 slot-filling with four or more consonants

Extensions of 1-to-1 slot-filling are also [1,1,1]-MISL. If the root contains four consonants **C**=*trʒm* and the template has four consonant slots **T**=*CVCCVC* (Table 1b), then the output *turʒim* is generated with the same [1,1,1]-MISL function that's modeled by the MT-FST in Figure 1. A sample derivation is provided in the appendix.

If the root contains more consonants **C**=*mɣnṭs* than the template has consonant slots **T**=*CVCCVC* (Table 1c), the output shows deletion of the additional consonant: *muɣniṭ* not **muɣniṭs*. This is [1,1,1]-MISL. It is modeled by the same MT-FST in Figure 1 but with the additional transition arc: $[c, \Sigma_{\rtimes}, \ltimes]$: $[+1, 0, 0]$: λ between q_1, q_1. A sample FST and derivation are provided in the appendix.

4.1.3 1-to-1 slot-filling and pre-associated affixes

Given a root **C**=*ksb*, some outputs show an additional affix, e.g. the infix <*t*> in *k*<*t*>*usib*. The affix <t> is pre-associated to a slot after the first consonant. Pre-associated templates can be computed either representationally or derivationally. Both are local.[7]

[7] A third alternative is to treat the infix <*t*> as part of a separate input-string or input-tape. The template is *CCVCVC* where **C** is pre-associated to <*t*>. This is analogous to giving each morpheme its own autosegmental tier (McCarthy,

	Current State	**C**-tape		**V**-tape		**T**-tape		Output Symbol	Output String
1.	q_0	$\underline{\rtimes}\mathrm{ktb}\ltimes$		$\underline{\rtimes}\mathrm{ui}\ltimes$		$\underline{\rtimes}\mathrm{CVCVC}\ltimes$			
2.	q_1	$\rtimes\underline{\mathrm{k}}\mathrm{tb}\ltimes$	$\mathbf{C}{:}\rtimes{:}{+1}$	$\rtimes\underline{\mathrm{u}}\mathrm{i}\ltimes$	$\mathbf{V}{:}\rtimes{:}{+1}$	$\rtimes\underline{\mathrm{C}}\mathrm{VCVC}\ltimes$	$\mathbf{T}{:}\rtimes{:}{+1}$	λ	
3.	q_1	$\rtimes\mathrm{k}\underline{\mathrm{t}}\mathrm{b}\ltimes$	$\mathbf{C}{:}k{:}{+1}$	$\rtimes\underline{\mathrm{u}}\mathrm{i}\ltimes$	$\mathbf{V}{:}u{:}0$	$\rtimes\mathrm{C}\underline{\mathrm{V}}\mathrm{CVC}\ltimes$	$\mathbf{T}{:}c{:}{+1}$	*k*	*k*
4.	q_1	$\rtimes\mathrm{k}\underline{\mathrm{t}}\mathrm{b}\ltimes$	$\mathbf{C}{:}t{:}0$	$\rtimes\mathrm{u}\underline{\mathrm{i}}\ltimes$	$\mathbf{V}{:}u{:}{+1}$	$\rtimes\mathrm{CV}\underline{\mathrm{C}}\mathrm{VC}\ltimes$	$\mathbf{T}{:}v{:}{+1}$	*u*	*ku*
5.	q_1	$\rtimes\mathrm{kt}\underline{\mathrm{b}}\ltimes$	$\mathbf{C}{:}t{:}{+1}$	$\rtimes\mathrm{u}\underline{\mathrm{i}}\ltimes$	$\mathbf{V}{:}i{:}0$	$\rtimes\mathrm{CVC}\underline{\mathrm{V}}\mathrm{C}\ltimes$	$\mathbf{T}{:}c{:}{+1}$	*t*	*kut*
6.	q_1	$\rtimes\mathrm{kt}\underline{\mathrm{b}}\ltimes$	$\mathbf{C}{:}b{:}0$	$\rtimes\mathrm{ui}\underline{\ltimes}$,	$\mathbf{V}{:}i{:}{+1}$	$\rtimes\mathrm{CVCV}\underline{\mathrm{C}}\ltimes$	$\mathbf{T}{:}v{:}{+1}$	*i*	*kuti*
7.	q_1	$\rtimes\mathrm{ktb}\underline{\ltimes}$	$\mathbf{C}{:}b{:}{+1}$	$\rtimes\mathrm{ui}\underline{\ltimes}$	$\mathbf{V}{:}\ltimes{:}0$	$\rtimes\mathrm{CVCVC}\underline{\ltimes}$	$\mathbf{T}{:}c{:}{+1}$	*b*	*kutib*
8.	q_f	$\rtimes\mathrm{ktb}\ltimes$	$\mathbf{C}{:}\ltimes{:}{+1}$	$\rtimes\mathrm{ui}\ltimes$	$\mathbf{C}{:}\ltimes{:}{+1}$	$\rtimes\mathrm{CVCVC}\ltimes$	$\mathbf{T}{:}\ltimes{:}{+1}$	λ	*kutib*

Table 2: Derivation of *kutib* using the MT-FST in Figure 1.

The representational route is to enrich the template with the affix itself: **T**=*CtVCVC* (Hudson, 1986). The root and template are then combined to generate *k<t>usib*. This function is [1,1,1]-MISL. It is computed by the same MT-FST in Figure 1 but with the additional transition arc: $[\Sigma_{\rtimes}, \Sigma_{\rtimes}, t] : [0, 0, t] : \lambda$ between q_1, q_1. A sample FST and derivation are provided in the appendix.

A derivational alternative is to derive *k<t>usib* from an un-affixed base *kusib* by infixing *<t>* (McCarthy, 1993). Generating *kusib* from [*ksb, ui, CVCVC*] is [1,1,1]-MISL. Infixing *<t>* onto *kusib* is 2-ISL. The representational route can be interpreted as the composition of the derivational approach.

4.2 1-to-many slot filling

4.2.1 Final spread

Final spread in *katab* has 1-to-many slot-filling (Table 1e). The word consists of the following input strings: **C**=*ktb*, **V**=*a*, **T**=*CVCVC*. The vocalism **V** consists of only one vowel *a* because of the Obligatory Contour Principle (McCarthy, 1981). The vowel *a* undergoes final spread by being associated with multiple *V* slots in the **T**-string.

Computing final vowel spread is [1,2,1]-MISL with $k_2 = 2$ on the **V**-string, not $k_2 = 1$. Knowing to spread the final vowel requires a window of size 2 on the **V**-string. The locality window stays at 1 for the **C**,**T**-strings because they do not play a role. For illustration, we provide an MT-FST for final vowel spread in the appendix. The states keep track of the last 1-suffix on the **V**-tape and last 0-suffix on **C**,**T**-tapes. A sample FST and derivation are provided in the appendix.

Consonants can also undergo final spread: f([*sm, a, CVCVC*] = *samam* (Table 1f).[8] This is [2,1,1]-MISL, analogous to final spread of vowels except that the locality window is now larger over the **C**-string instead of the **V**-string.

4.2.2 Medial spread

In contrast to final spread, medial spread involves associating a string-medial vowel or consonant to multiple slots on the **T**-string: *kuutib* with a long-vowel *u* (Table 1g) or *kuttib* with a geminate *t* (Table 1h). Like pre-associated affixes (§4.1.3), medial spread can be analyzed either representationally or derivationally. An alternative edge-in analysis is discussed in §5.2.

For gemination, the representational route involves enriching the template with a special symbol, i.e., a consonant mora μ_C in **T**=$CVC\mu_V VC$ (Kay, 1987; McCarthy, 1993; Beesley, 1998). With this template, generating *kuttib* is [2,1,1]-MISL with k_1=2 over the **C**-string. A corresponding MT-FST and derivation is in the appendix using $\Sigma_T = \{C, V, \mu_C\}$, and $\Sigma_C = \{k, t\}$ for illustration. Long vowels have the same computational treatment but with μ_V as a special symbol.

A derivational alternative is to derive *kuttib* from *kutib* by infixing a consonant mora μ_C followed by consonant spreading. Generating the base *kutib* is [1,1,1]-MISL. Infixing the mora *kut*μ_C*ib* is 4-ISL and spreading the consonant *kuttib* is 2-ISL. As with preassociation (§4.1.3), the

1981). But computing this type of input-structure cannot be modeled in an MT-FST because MT-FSTs work over multiple linear strings, not over graphs.

[8] Since McCarthy (1981), the analysis of final consonant spread has been controversial (Hudson, 1986; Hoberman, 1988; Yip, 1988; McCarthy, 1993; Gafos, 1998; Bat-El, 2006). Alternative analyses involving reduplication, preference for local spreading, or right-to-left association can be potentially non-local and are discussed in §5. Computationally, Beesley (1998) formalizes consonant spread with a special symbol X as an equivalent treatment for medial spread. This formalization is [2,1,1]-MISL, just like (§4.2.2.

representational solution is a composition of the derivational solution; both are local functions.

5 Possible non-locality in Semitic

Certain templatic processes in Semitic are not local: reduplication and loanword adaptation in Table 3, amassed from many sources (McCarthy, 1981; Broselow and McCarthy, 1983; Bat-El, 2011).

5.1 Reduplication

Semitic RPM shows intensive reduplication which varies on root size (Broselow and McCarthy, 1983): root doubling in for biconsonantal roots in *laflaf* (Table 3i) and first-C copying for triconsonantal roots in *barbad* (Table 3j). Root-doubling is analogous to total reduplication. Initial-C copying involves copying the first consonant of the root and placing it in a prespecified spot on the template.[9]

Reduplication is computationally challenging. Cross-linguistically, partial reduplication patterns can range from being ISL to subsequential (Chandlee and Heinz, 2012). Total reduplication is above the subsequential threshold and cannot be modeled by 1-way FSTs but requires deterministic 2-way FSTs (Dolatian and Heinz, 2018). If we assume that there's no bound on the size of the root, then root-doubling cannot be computed by a MISL function for any $\vec{k}$. The function would need a 2-way MT-FST which could go back and forth on the **C**-tape. Similarly, if we assume that there's no bound on the number n of consonants between the two copies of the root-initial consonant, then the function is not MISL for any $\vec{k}$. Analogously to subsequential functions over single-input FSTs, root-initial copying would be Multi-Subsequential. However, the assumption on root size is not correct. All roots which undergo the above reduplication processes have a bounded size (2 or 3). If we discard this assumption, then both reduplicative processes are MISL for a large value of $\vec{k}$.[10]

5.2 Local spreading in loanword adaptation

In loanword adaptation of verbs in Arabic, the most productive template is *CVCCVC* with the vocalism *a*: *CaCCaC* (Bat-El, 2011). When a borrowed consonantal root has four consonants, the template is filled with 1-to-1 slot filling of consonants: *telephone* [telefon] and *talfan* (Table 3k). But when a borrowed root has three consonants, then the input undergoes medial gemination: *SMS* and *sammas*, not final spread **samsas* (Table 3l).

There are many ways to analyze this difference between three vs. four-consonant roots. One is suppletive allomorphy: four-consonant roots use the template *CVCCVC*, three-consonant roots use the template $CVC\mu_C VC$. Choosing the template is ISL-4. Once chosen, the root, vocalism, and template can then be submitted to an MISL function. This analysis is plausible because, outside of loanword adaptation, Semitic templates *do* have suppletion conditioned by root-size: the comparative in Egyptian Arabic is *VCCVC* for triconsonantal roots: *kbr* → *akbar*, but *VCVCC* for biconsonantal roots: *ʃd* → *aʃadd* (Davis and Tsujimura, 2018).

An alternative is to use a template *CVCCVC* without any representational markup for gemination. The correct outputs are generated based on avoiding non-local spreading. For a three-consonant root, medial gemination is generated because the grammar (in OT-parlance) prefers outputs with local spreading of consonants *sa***mm***as* instead of outputs with non-local spreading *sam***s***a***s**. An analogous anti-long-distance spreading mechanism has been proposed for medial gemination (§4.2.2) and for the fact that *i* cannot spread (§4.2.1) (Hudson, 1986; Hoberman, 1988; Yip, 1988).[11] Computationally, the choice of local spreading depends on the following information:

1. Having the context CCV on the template: $k = 3$ on **T**-string
2. Being the final consonant in the root or not: $k = 2$ on **C**-string
3. The existence of an additional C slot on the template: $XCCV_yC\ltimes$ vs. $\mathrm{XCCV_y}\ltimes$: $k = |V_{\mathrm{x}}| + 1$ on **T**-string

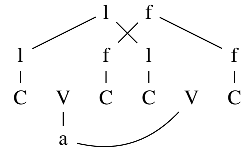
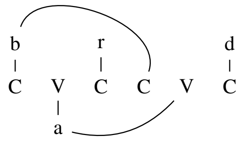
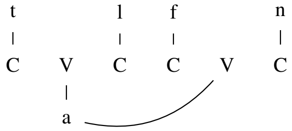
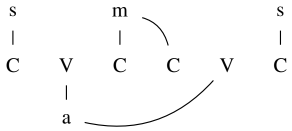
The last condition is important. Consider the contrast in *kuttib* and *kutba* 'writers' derived from the templates $C_1V_xC_2C_3V_yC_4$ and$C_1V_xC_2C_3V_y$.

[9] Technically, the relevant inputs need to be annotated to trigger reduplication, e.g. initial-C copying with **T**=*CVCFVC* and root doubling with **C**=*zl*-RED. We abstract away from this for clarity.

[10] The value of the k is [3,1,1] for initial-C copying, but [3,1,3] for root-doubling because the function keeps track of the root size and the current C-slot.

[11] These have also been analyzed with edge-in association. Instead of association operating from left-to-right, Yip (1988) argues that these templates are simultaneously or consecutively right-to-left and left-to-right. Such an analysis though has unclear computational expressivity; we conjecture that it may be analogous to Weak Determinism (Heinz and Lai, 2013) over multiple inputs.

Table 3: Partial paradigm of Arabic root-and-pattern morphology with variable MISL $\vec{k}$-values.

	Slot-filling pattern	Binyan	Gloss	Output	Root	Vowels	Template	k-value
	Reduplication of							
i	... root		*laflaf*	'wrapped intensely'	*lf*	*a*	*CVCCVC*	varies
								
j	... first C		*barbad*	'shaved unevenly'	*brd*	*a*	*CVCFVC*	varies
								
	Loanword adaptation of...	Source noun	Adapted Verb					
k	... four-consonant root	*telephone*	*talfan*	'he phoned'	*tlfn*	*a*	*CVCCVC*	varies
								
l	... three-consonant root	*SMS*	*sammas*	'he SMS-ed'	*sms*	*a*	*CVCCVC*	varies
								

The C_2C_3 substring in $C_1V_xC_2C_3V_yC_4$ maps to gemination: *kuttib*, while the *CC* substring in *CVCCV* maps to 1-to-1 spreading: *kutba*. The choice depends on if the C_1C_2 substrings precedes an extra consonant slot C_4 on the template or not. If there is no bound on the number of intervening vowels V_x, then the function is not MISL for any $\vec{k}$. If there is a bound, then it is MISL for a k which is sufficiently large enough to encode these contexts. In Arabic, V_y can be at most two vowels slots in order to encode long vowels: *kuttaab* 'writers'. This makes the function MISL with $k = 5$ on the **T**-string, $k = 3$ on the **C**-string.

6 Conclusion

This paper examined the computational expressivity of non-concatenative morphology, in particular, Semitic root-and-pattern morphology (RPM). Generalizing Input Strictly Local (ISL) functions to handle multiple inputs, we showed that the class of Multiple-Input Strictly Local (MISL) functions can compute almost all Semitic RPM. These MISL functions are computed by deterministic asynchronous multi-tape finite-state transducers. This computational result looks beyond various points of theoretical contention in Semitic. The result also narrows the gap in mathematical results between concatenative and non-concatenative morphology.

References

Mark Aronoff. 2013. The roots of language. In Silvio Cruschina, Martin Maiden, , and John Charles Smith, editors, The boundaries of pure morphology, pages 161–180.

Outi Bat-El. 2006. Consonant identity and consonant copy: The segmental and prosodic structure of hebrew reduplication. Linguistic Inquiry, 37(2):179–210.

Outi Bat-El. 2011. Semitic templates. In (van Oostendorp et al., 2011), pages 2586–2609.

Kenneth Beesley and Lauri Karttunen. 2003. Finite-state morphology: Xerox tools and techniques. CSLI Publications, Stanford, CA.

Kenneth R Beesley. 1998. Consonant spreading in arabic stems. In Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on

Computational Linguistics-Volume 1, pages 117–123. Association for Computational Linguistics.

Kenneth R. Beesley and Lauri Karttunen. 2000. Finite-state non-concatenative morphotactics. In Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, ACL '00, pages 191–198, Hong Kong. Association for Computational Linguistics.

Steven Bird and T Mark Ellison. 1994. One-level phonology: Autosegmental representations and rules as finite automata. Computational Linguistics, 20(1):55–90.

Ellen Broselow and John McCarthy. 1983. A theory of internal reduplication. The Linguistic Review, 3(1):25–88.

Jane Chandlee. 2014. Strictly Local Phonological Processes. Ph.D. thesis, University of Delaware, Newark, DE.

Jane Chandlee. 2017. Computational locality in morphological maps. Morphology, pages 1–43.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015a. Output strictly local functions. In 14th Meeting on the Mathematics of Language, pages 112–125.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2015b. Output strictly local functions. In Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015), pages 112–125, Chicago, USA.

Jane Chandlee and Jeffrey Heinz. 2012. Bounded copying is subsequential: Implications for metathesis and reduplication. In Proceedings of the 12th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology, SIGMORPHON '12, pages 42–51, Montreal, Canada. Association for Computational Linguistics.

Jane Chandlee and Jeffrey Heinz. 2018. Strict locality and phonological maps. Linguistic Inquiry, 49(1):23–60.

Yael Cohen-Sygal and Shuly Wintner. 2006. Finite-state registered automata for non-concatenative morphology. Computational Linguistics, 32(1):49–82.

Stuart Davis and Natsuko Tsujimura. 2018. Arabic nonconcatenative morphology in construction morphology. In Geert Booij, editor, The Construction of Words: Advances in Construction Morphology, volume 4. Springer.

Hossep Dolatian and Jeffrey Heinz. 2018. Modeling reduplication with 2-way finite-state transducers. In Proceedings of the 15th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, Brussells, Belgium. Association for Computational Linguistics.

Hossep Dolatian and Jonathan Rawski. 2019. Finite-state locality in semitic root-and-pattern morphology. In University of Pennsylvania Working Papers in Linguistics.

C. C. Elgot and J. E. Mezei. 1965. On relations defined by generalized finite automata. IBM Journal of Research and Development, 9(1):47–68.

Emmanuel Filiot and Pierre-Alain Reynier. 2016. Transducers, logic and algebra for functions of finite words. ACM SIGLOG News, 3(3):4–19.

Patrick C Fischer. 1965. Multi-tape and infinite-state automataa survey. Communications of the ACM, 8(12):799–805.

Patrick C Fischer and Arnold L Rosenberg. 1968. Multitape one-way nonwriting automata. Journal of Computer and System Sciences, 2(1):88–101.

Christiane Frougny and Jacques Sakarovitch. 1993. Synchronized rational relations of finite and infinite words. Theoretical Computer Science, 108(1):45–82.

Carlo A. Furia. 2012. A survey of multi-tape automata. http://arxiv.org/abs/1205.0178. Latest revision: November 2013.

Diamandis Gafos. 1998. Eliminating long-distance consonantal spreading. Natural Language & Linguistic Theory, 16(2):223–278.

Michael Hammond. 1988. Templatic transfer in arabic broken plurals. Natural Language & Linguistic Theory, 6(2):247–270.

Jeffrey Heinz and Regine Lai. 2013. Vowel harmony and subsequentiality. In Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13), pages 52–63, Sofia, Bulgaria. Association for Computational Linguistics.

Robert D Hoberman. 1988. Local and long-distance spreading in semitic morphology. Natural Language & Linguistic Theory, 6(4):541–549.

Grover Hudson. 1986. Arabic root and pattern morphology without tiers. Journal of Linguistics, 22(1):85–122.

Mans Hulden. 2009. Revisiting multi-tape automata for semitic morphological analysis and generation. In Proceedings of the EACL 2009 Workshop on Computational Approaches to Semitic Languages, pages 19–26. Association for Computational Linguistics.

Itamar Kastner. 2016. Form and meaning in the Hebrew verb. Ph.D. thesis, New York University.

Martin Kay. 1987. Nonconcatenative finite-state morphology. In Third Conference of the European Chapter of the Association for Computational Linguistics.

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.

John McCarthy and Alan Prince. 1990a. Prosodic morphology and templatic morphology. In Perspectives on Arabic linguistics II: papers from the second annual symposium on Arabic linguistics, pages 1–54. John Benjamins Amsterdam.

John J McCarthy. 1981. A prosodic theory of nonconcatenative morphology. Linguistic inquiry, 12(3):373–418.

John J McCarthy. 1993. Template form in prosodic morphology. In Proceedings of the Formal Linguistics Society of Mid-America, volume 3, pages 187–218.

John J McCarthy and Alan S Prince. 1990b. Foot and word in prosodic morphology: The Arabic broken plural. Natural Language & Linguistic Theory, 8(2):209–283.

Robert McNaughton and Seymour A Papert. 1971. Counter-Free Automata (MIT research monograph no. 65). The MIT Press.

Marc van Oostendorp, Colin Ewen, Elizabeth Hume, and Keren Rice, editors. 2011. The Blackwell companion to phonology. Wiley-Blackwell, Malden, MA.

Jean-François Prunet. 2006. External evidence and the semitic root. Morphology, 16(1):41.

Michael O Rabin and Dana Scott. 1959. Finite automata and their decision problems. IBM journal of research and development, 3(2):114–125.

Brian Roark and Richard Sproat. 2007. Computational Approaches to Morphology and Syntax. Oxford University Press, Oxford.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In Formal Grammar, volume 8036 of Lecture Notes in Computer Science, pages 90–108. Springer.

James Rogers and Geoffrey Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. Journal of Logic, Language and Information, 20:329–342.

Walter J Savitch. 1993. Why it might pay to assume that languages are infinite. Annals of Mathematics and Artificial Intelligence, 8(1-2):17–25.

Matthew A Tucker. 2010. Roots and prosody: the iraqi arabic derivational verb. Recherches linguistiques de Vincennes, (39):31–68.

Adam Ussishkin. 2005. A fixed prosodic theory of nonconcatenative templaticmorphology. Natural Language & Linguistic Theory, 23(1):169–218.

Adam Ussishkin. 2011. Tier segregation. In (van Oostendorp et al., 2011), pages 2516–2535.

Bruce Wiebe. 1992. Modelling autosegmental phonology with multi-tape finite state transducers. Master's thesis, Simon Fraser University.

Shuly Wintner. 2014. Morphological processing of semitic languages. In Imed Zitouni, editor, Natural language processing of Semitic languages, pages 43–66. Springer.

Moira Yip. 1988. Template morphology and the direction of association. Natural Language & Linguistic Theory, 6(4):551–577.

Sam Zukoff. 2017. Arabic nonconcatenative morphology and the syntax-phonology interface. In NELS 47: Proceedings of the Forty-Seventh Annual Meeting of the North East Linguistic Society, volume 3, page 295314, Amherst, MA. Graduate Linguistics Student Association.

A Appendix

Below are MT-FSTs and derivation tables for some of the described Semitic processes.

A.1 1-to-1 slot-filling with four consonants

In Table 1b, the input root **C** has 4 consonants *trʒm* and the template **T** has enough consonantal slots *CVCCVC*. The vocalism **V** is *ui*. The output is *turʒim*. A derivation table is provided in Table 4 using the [1,1,1]-MISL MT-FST from Figure 1.

A.2 1-to-1 slot-filling with larger roots

In Table 1c, the root **C**==*mɣnts* contains more consonants than the template **T**=*CVCCVC*. With a vocalism **V**=*ui*, the output is *muɣnit* with final consonant deletion. This function is modeled by the [1,1,1]-MISL MT-FST in Figure 3, illustrated with the derivation in Table 5.

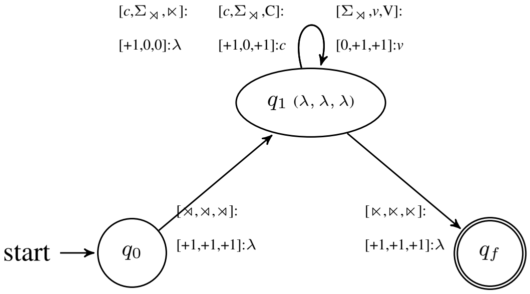


Figure 3: MT-FST for 1-to-1 slot-filling with final consonant deletion

A.3 1-to-1 slot-filling and pre-associated affixes

The template **T**=*CtVCVC* has a preassociated affix ⟨*t*⟩. With a root **C**=*ksb* and vocalism **V**=*ui*, the output is *ktusib*. A [1,1,1]-MISL MT-FST is provided in Figure 4 along with a sample derivation in Table 6. The symbol *A* represents any input symbol from the input alphabet of segments {t,n,m} which are possible segmental affixes in McCarthy (1981).

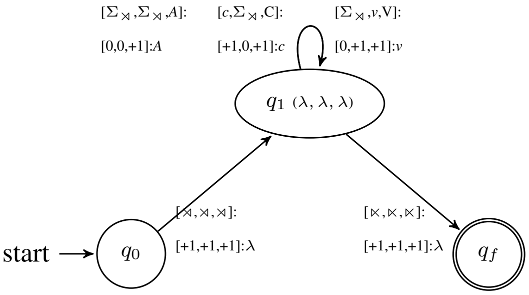


Figure 4: MT-FST for 1-to-1 slot-filling with pre-associated affixes

A.4 1-to-many slot-filling with final spread of vowels

In Table 1e, the vocalism **V**=*a* has fewer vowels than the template **T**=*CVCVC*. This triggers final spread of vowels. With a root **C**=*ktb*, the output is *katab*. This function is modeled with the [1,2,1]-MISL MT-FST in Figure 5, illustrated with a sample derivation in Table 7. The vowel alphabet is only {a,u}. In Standard Arabic, only the vowels *a,u* spread; the vowel *i* does not. This is discussed in §5.2. The FST does not visually represent the dedicated final state q_f. Instead, all non-initial states are marked as accepting states. A state is accepting if upon reading [⋉,⋉,⋉], it advances [+1,+1,+1] to state q_f.

A.5 1-to-many slot filling with medial spread of consonants

In Table 1g, the template **T**=*CVC*μ_C*VC* contains a marker for gemination. With root **C**=*ktb* and vocalism **V**=*ui*, the output is *kuttib*. This is modeled by the [2,1,1]-MISL MT-FST in Figure 6. with a sample derivation in Table 8 for a nonce word *kuttik* with root **C**=*ktk*. For illustrative reasons, the consonant alphabet is only {k,t}. The final state q_f is not visualized for space reasons.

	Current State	**C**-tape		**V**-tape		**T**-tape		Output Symbol	Output String
1.	q_0	⋊̲trʒm⋉		⋊̲ui⋉		⋊̲CVCCVC⋉			
2.	q_1	⋊t̲rʒɱ⋉	**C**:⋊:+1	⋊u̲i⋉	**V**:⋊:+1	⋊C̲VCCVC⋉	**T**:⋊:+1	λ	
3.	q_1	⋊tr̲ʒɱ⋉	**C**:*t*:+1	⋊u̲i⋉	**V**:*u*:0	⋊CV̲CCVC⋉	**T**:*C*:+1	*t*	*t*
4.	q_1	⋊tr̲ʒɱ⋉	**C**:*r*:0	⋊ui̲⋉	**V**:*u*:+1	⋊CVC̲CVC⋉	**T**:*V*:+1	*u*	*tu*
5.	q_1	⋊trʒ̲ɱ⋉	**C**:*r*:+1	⋊ui̲⋉	**V**:*i*:0	⋊CVCC̲VC⋉	**T**:*C*:+1	*r*	*tur*
6.	q_1	⋊trʒm̲⋉	**C**:*ʒ*:+1	⋊ui̲⋉	**V**:*i*:0	⋊CVCCV̲C⋉	**T**:*C*:+1	*ʒ*	*turʒ*
7.	q_1	⋊trʒm̲⋉	**C**:*m*:0	⋊ui⋉̲	**V**:*i*:+1	⋊CVCCVC̲⋉	**T**:*V*:+1	*i*	*turʒi*
8.	q_1	⋊trʒm⋉̲	**C**:*m*:+1	⋊ui⋉̲	**V**:⋉:0	⋊CVCCVC⋉̲	**T**:*C*:+1	*m*	*turʒim*
9.	q_1	⋊trʒm⋉	**C**:⋉:+1	⋊ui⋉	**V**:⋉:+1	⋊CVCCVC⋉	**T**:⋉:+1	λ	*turʒim*

Table 4: Derivation of *turʒim* using the MT-FST in Figure 1.

	Current State	**C**-tape		**V**-tape		**T**-tape		Output Symbol	Output String
1.	q_0	⋊̲mɣnts⋉		⋊̲ui⋉		⋊̲CVCCVC⋉			
2.	q_1	⋊m̲ɣnts⋉	**C**:⋊:+1	⋊u̲i⋉	**V**:⋊:+1	⋊C̲VCCVC⋉	**T**:⋊:+1	λ	
3.	q_1	⋊mɣ̲nts⋉	**C**:*m*:+1	⋊u̲i⋉	**V**:*u*:0	⋊CV̲CCVC⋉	**T**:*C*:+1	*m*	*m*
4.	q_1	⋊mɣ̲nts⋉	**C**:*ɣ*:0	⋊ui̲⋉	**V**:*u*:+1	⋊CVC̲CVC⋉	**T**:*V*:+1	*u*	*mu*
5.	q_1	⋊mɣn̲ts⋉	**C**:*ɣ*:+1	⋊ui̲⋉	**V**:*i*:0	⋊CVCC̲VC⋉	**T**:*C*:+1	*G*	*muG*
6.	q_1	⋊mɣnt̲s⋉	**C**:*n*:+1	⋊ui̲⋉	**V**:*i*:0	⋊CVCCV̲C⋉	**T**:*C*:+1	*n*	*muGn*
7.	q_1	⋊mɣnt̲s⋉	**C**:*t*:0	⋊ui⋉̲	**V**:*i*:+1	⋊CVCCVC̲⋉	**T**:*V*:+1	*i*	*muGni*
8.	q_1	⋊mɣnts̲⋉	**C**:*t*:+1	⋊ui⋉̲	**V**:⋉:0	⋊CVCCVC⋉̲	**T**:*C*:+1	*t*	*muGnit*
9.	q_1	⋊mɣnts⋉̲	**C**:*s*:+1	⋊ui⋉̲	**V**:⋉:0	⋊CVCCVC⋉̲	**T**:⋉:0	λ	*muGnit*
10.	q_1	⋊mɣnts⋉	**C**:⋉:+1	⋊ui⋉	**V**:⋉:+1	⋊CVCCVC⋉	**T**:⋉:+1	λ	*muGnit*

Table 5: Derivation of *muɣnit* using the MT-FST in Figure 3

	Current State	**C**-tape		**V**-tape		**T**-tape		Output Symbol	Output String
1.	q_0	⋊̲ksb⋉		⋊̲ui⋉		⋊̲CtVCVC⋉			
2.	q_1	⋊k̲sb⋉	**C**:⋊:+1	⋊u̲i⋉	**V**:⋊:+1	⋊C̲tVCVC⋉	**T**:⋊:+1	λ	
3.	q_1	⋊ks̲b⋉	**C**:*k*:+1	⋊u̲i⋉	**V**:*u*:0	⋊Ct̲VCVC⋉	**T**:*C*:+1	*k*	*k*
4.	q_1	⋊ks̲b⋉	**C**:*s*:0	⋊u̲i⋉	**V**:*u*:0	⋊CtV̲CVC⋉	**T**:*t*:+1	*t*	*kt*
5.	q_1	⋊ks̲b⋉	**C**:*s*:0	⋊ui̲⋉	**V**:*u*:+1	⋊CtVC̲VC⋉	**T**:*V*:+1	*u*	*ktu*
6.	q_1	⋊ksb̲⋉	**C**:*s*:+1	⋊ui̲⋉	**V**:*i*:0	⋊CtVCV̲C⋉	**T**:*C*:+1	*s*	*ktus*
7.	q_1	⋊ksb̲⋉	**C**:*b*:0	⋊ui⋉̲	**V**:*i*:+1	⋊CtVCVC̲⋉	**T**:*V*:+1	*i*	*ktusi*
8.	q_1	⋊ksb⋉̲	**C**:*b*:+1	⋊ui⋉̲	**V**:⋉:0	⋊CtVCVC⋉̲	**T**:*C*:+1	*b*	*ktusib*
9.	q_1	⋊ksb⋉	**C**:⋉:+1	⋊ui⋉	**V**:⋉:+1	⋊CtVCVC⋉	**T**:⋉:+1	λ	*ktusib*

Table 6: Derivation of *k*⟨*t*⟩*usib* using the MT-FST in Figure 4

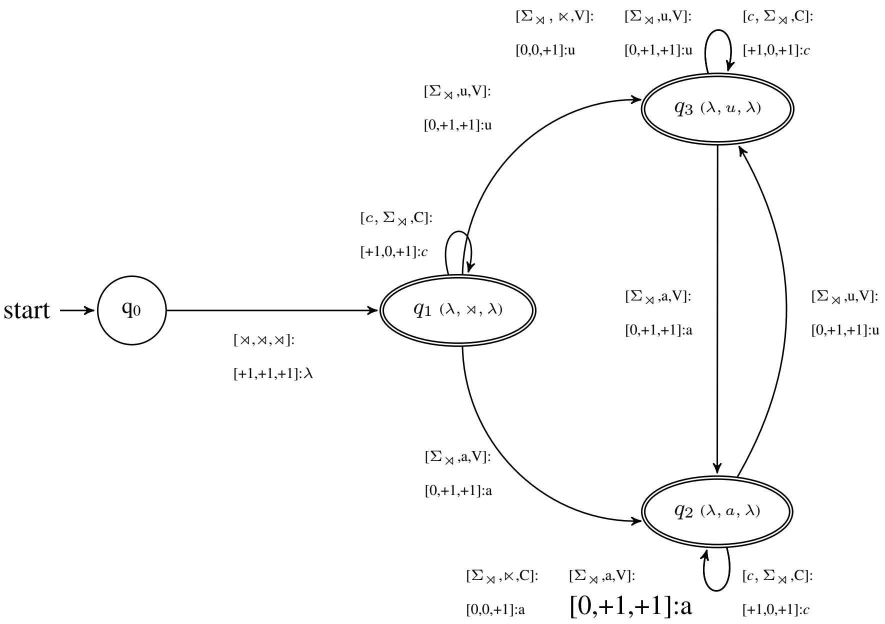


Figure 5: MT-FST for 1-to-many slot-filling with final spread of vowels

	Current State	C-tape		V-tape		T-tape		Output Symbol	Output String
1.	q_0	$\underline{\rtimes}\mathrm{ktb}\ltimes$		$\underline{\rtimes}\mathrm{a}\ltimes$		$\underline{\rtimes}\mathrm{CVCVC}\ltimes$			
2.	q_1	$\rtimes\underline{\mathrm{k}}\mathrm{tb}\ltimes$	$\mathbf{C}{:}\rtimes{:}{+}1$	$\rtimes\underline{\mathrm{a}}\ltimes$	$\mathbf{V}{:}\rtimes{:}{+}1$	$\rtimes\underline{\mathrm{C}}\mathrm{VCVC}\ltimes$	$\mathbf{T}{:}\rtimes{:}{+}1$	λ	
3.	q_1	$\rtimes\mathrm{k}\underline{\mathrm{t}}\mathrm{b}\ltimes$	$\mathbf{C}{:}k{:}{+}1$	$\rtimes\underline{\mathrm{a}}\ltimes$	$\mathbf{V}{:}a{:}0$	$\rtimes\mathrm{C}\underline{\mathrm{V}}\mathrm{CVC}\ltimes$	$\mathbf{T}{:}C{:}{+}1$	k	*k*
4.	q_2	$\rtimes\mathrm{k}\underline{\mathrm{t}}\mathrm{b}\ltimes$	$\mathbf{C}{:}t{:}0$	$\rtimes\mathrm{a}\underline{\ltimes}$	$\mathbf{V}{:}a{:}{+}1$	$\rtimes\mathrm{CV}\underline{\mathrm{C}}\mathrm{VC}\ltimes$	$\mathbf{T}{:}V{:}{+}1$	a	*ka*
5.	q_2	$\rtimes\mathrm{kt}\underline{\mathrm{b}}\ltimes$	$\mathbf{C}{:}t{:}{+}1$	$\rtimes\mathrm{a}\underline{\ltimes}$	$\mathbf{V}{:}\ltimes{:}0$	$\rtimes\mathrm{CVC}\underline{\mathrm{V}}\mathrm{C}\ltimes$	$\mathbf{T}{:}t{:}{+}1$	t	*kat*
6.	q_2	$\rtimes\mathrm{kt}\underline{\mathrm{b}}\ltimes$	$\mathbf{C}{:}b{:}0$	$\rtimes\mathrm{a}\underline{\ltimes},$	$\mathbf{V}{:}\ltimes{:}0$	$\rtimes\mathrm{CVCV}\underline{\mathrm{C}}\ltimes$	$\mathbf{T}{:}V{:}{+}1$	a	*kata*
7.	q_2	$\rtimes\mathrm{ktb}\underline{\ltimes}$	$\mathbf{C}{:}b{:}{+}1$	$\rtimes\mathrm{a}\underline{\ltimes}$	$\mathbf{V}{:}\ltimes{:}0$	$\rtimes\mathrm{CVCVC}\underline{\ltimes}$	$\mathbf{T}{:}C{:}{+}1$	b	*katab*
8.	q_f	$\rtimes\mathrm{ktb}\ltimes$	$\mathbf{C}{:}\ltimes{:}{+}1$	$\rtimes\mathrm{a}\ltimes$	$\mathbf{C}{:}\ltimes{:}{+}1$	$\rtimes\mathrm{CVCVC}\ltimes$	$\mathbf{T}{:}\ltimes{:}{+}1$	λ	*katab*

Table 7: Derivation of *katab* using the MT-FST in Figure 5

	Current State	C-tape		V-tape		T-tape		Output Symbol	Output String
1.	q_0	$\underline{\rtimes}\mathrm{ktk}\ltimes$		$\underline{\rtimes}\mathrm{ui}\ltimes$		$\underline{\rtimes}\mathrm{CVC}\mu_C\mathrm{VC}\ltimes$			
2.	q_1	$\rtimes\underline{\mathrm{k}}\mathrm{tk}\ltimes$	$\mathbf{C}{:}\rtimes{:}{+}1$	$\rtimes\underline{\mathrm{u}}\mathrm{i}\ltimes$	$\mathbf{V}{:}\rtimes{:}{+}1$	$\rtimes\underline{\mathrm{C}}\mathrm{VC}\mu_C\mathrm{VC}\ltimes$	$\mathbf{T}{:}\rtimes{:}{+}1$	λ	
3.	q_2	$\rtimes\mathrm{k}\underline{\mathrm{t}}\mathrm{k}\ltimes$	$\mathbf{C}{:}k{:}{+}1$	$\rtimes\underline{\mathrm{u}}\mathrm{i}\ltimes$	$\mathbf{V}{:}u{:}0$	$\rtimes\mathrm{C}\underline{\mathrm{V}}\mathrm{C}\mu_C\mathrm{VC}\ltimes$	$\mathbf{T}{:}C{:}{+}1$	k	*k*
4.	q_2	$\rtimes\mathrm{k}\underline{\mathrm{t}}\mathrm{k}\ltimes$	$\mathbf{C}{:}k{:}0$	$\rtimes\mathrm{u}\underline{\mathrm{i}}\ltimes$	$\mathbf{V}{:}u{:}{+}1$	$\rtimes\mathrm{CV}\underline{\mathrm{C}}\mu_C\mathrm{VC}\ltimes$	$\mathbf{T}{:}V{:}{+}1$	u	*ku*
5.	q_3	$\rtimes\mathrm{kt}\underline{\mathrm{k}}\ltimes$	$\mathbf{C}{:}t{:}{+}1$	$\rtimes\mathrm{u}\underline{\mathrm{i}}\ltimes$	$\mathbf{V}{:}i{:}0$	$\rtimes\mathrm{CVC}\underline{\mu_C}\mathrm{VC}\ltimes$	$\mathbf{T}{:}C{:}{+}1$	t	*kut*
6.	q_3	$\rtimes\mathrm{kt}\underline{\mathrm{k}}\ltimes$	$\mathbf{C}{:}k{:}0$	$\rtimes\mathrm{u}\underline{\mathrm{i}}\ltimes$	$\mathbf{V}{:}i{:}0$	$\rtimes\mathrm{CVC}\mu_C\underline{\mathrm{V}}\mathrm{C}\ltimes$	$\mathbf{T}{:}\mu_C{:}{+}1$	t	*kutt*
7.	q_3	$\rtimes\mathrm{kt}\underline{\mathrm{k}}\ltimes$	$\mathbf{C}{:}k{:}0$	$\rtimes\mathrm{ui}\underline{\ltimes}$	$\mathbf{V}{:}i{:}{+}1$	$\rtimes\mathrm{CVC}\mu_C\mathrm{V}\underline{\mathrm{C}}\ltimes$	$\mathbf{T}{:}V{:}{+}1$	i	*kutti*
8.	q_3	$\rtimes\mathrm{ktk}\underline{\ltimes}$	$\mathbf{C}{:}k{:}{+}1$	$\rtimes\mathrm{ui}\underline{\ltimes}$	$\mathbf{V}{:}\ltimes{:}0$	$\rtimes\mathrm{CVC}\mu_C\mathrm{VC}\underline{\ltimes}$	$\mathbf{T}{:}C{:}{+}1$	k	*kuttik*
9.	q_f	$\rtimes\mathrm{ktk}\ltimes$	$\mathbf{C}{:}\ltimes{:}{+}1$	$\rtimes\mathrm{ui}\ltimes$	$\mathbf{V}{:}\ltimes{:}{+}1$	$\rtimes\mathrm{CVC}\mu_C\mathrm{VC}\ltimes$	$\mathbf{T}{:}\ltimes{:}{+}1$	λ	*kuttik*

Table 8: Derivation of *kuttik* using the MT-FST in Figure 6

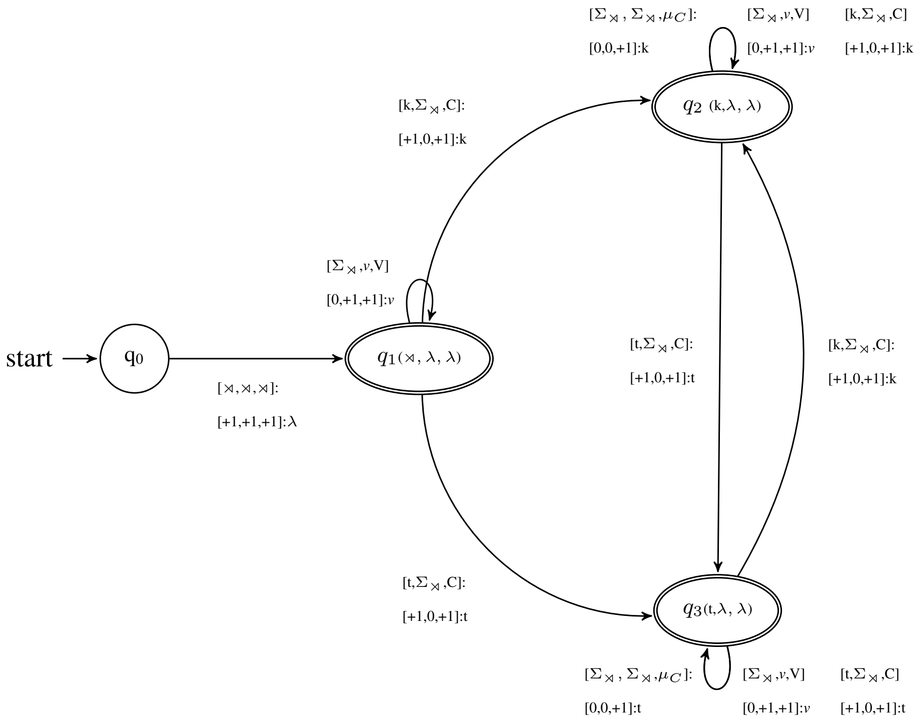

$[\Sigma_{\rtimes}, \Sigma_{\rtimes}, \mu_C]$:
$[0,0,+1]$:k
$[\Sigma_{\rtimes}, v, \mathrm{V}]$
$[0,+1,+1]$:v
$[\mathrm{k}, \Sigma_{\rtimes}, \mathrm{C}]$
$[+1,0,+1]$:k
$[\mathrm{k}, \Sigma_{\rtimes}, \mathrm{C}]$:
$[+1,0,+1]$:k
q_2 (k,λ, λ)
$[\Sigma_{\rtimes}, v, \mathrm{V}]$
$[0,+1,+1]$:v
start
q_0
q_1($\rtimes$, λ, λ)
$[\rtimes, \rtimes, \rtimes]$:
$[+1,+1,+1]$:λ
$[\mathrm{t}, \Sigma_{\rtimes}, \mathrm{C}]$:
$[+1,0,+1]$:t
$[\mathrm{k}, \Sigma_{\rtimes}, \mathrm{C}]$:
$[+1,0,+1]$:k
$[\mathrm{t}, \Sigma_{\rtimes}, \mathrm{C}]$:
$[+1,0,+1]$:t
q_3(t,λ, λ)
$[\Sigma_{\rtimes}, \Sigma_{\rtimes}, \mu_C]$:
$[0,0,+1]$:t
$[\Sigma_{\rtimes}, v, \mathrm{V}]$
$[0,+1,+1]$:v
$[\mathrm{t}, \Sigma_{\rtimes}, \mathrm{C}]$
$[+1,0,+1]$:t

Figure 6: MT-FST for 1-to-many slot-filling with medial spread of consonants