Multi-Input Strictly Local Functions for Templatic Morphology

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

In mathematical phonology, there has been work on discovering how phonological processes map to subclasses of rational functions (McNaughton and Papert, 1971; Rogers and Pullum, 2011; Rogers et al., 2013; Heinz and Lai, 2013; Chandelle, 2014). One of the simplest subclasses is the class of Input Strictly Local (ISL) 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.

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

In mathematical phonology, there has been work on discovering how phonological processes map to subclasses of rational functions (McNaughton and Papert, 1971; Rogers and Pullum, 2011; Rogers et al., 2013; Heinz and Lai, 2013; Chandelle, 2014). One of the simplest subclasses is the class of Input Strictly Local (ISL) 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.

2 Preliminaries

2.1 Preliminaries for single-input functions

Let $\times, \cdot$ be the start and end boundaries respectively. Let $\Sigma$ be a finite alphabet of symbols (excluding $\times, \cdot$). Let $\Sigma^* = \Sigma \cup \{ \times, \cdot \}$. Let $\Sigma^*$ be the set of all strings over $\Sigma$. Let $|w|$ denote the length of $w \in \Sigma^*$. For two strings $w$ and $v$ let $wv$ be their concatenation, and for a set $L \subseteq \Sigma^*$ of strings and a string $w$, by $wL$ we denote $\{wv | v \in L \}$. Let $\lambda$ denote the empty string.

Given some string $u$ and a natural number $k$, the $k$-suffix or $suff_k$ of $u$ is the last $k$ symbols of $u$: \[ suff_k(u, k) = v \text{ s.t. } |v| = k \text{ and } xv = u \text{ for } x \in \Sigma^*. \]

The prefixes of a string $w \in \Sigma^*$, $\text{prfs}(w) = \{u | uv = w \in L \text{ for some } v \in \Sigma^* \}$. The common prefixes of a set $L \subseteq \Sigma^*$, \[ \text{cmpfrs}(L) \triangleq \bigcap \text{prfs}(w). \]

The longest common prefix of a set $L \subseteq \Sigma^*$, $\text{lcp}(L) = u$ such that $u \in \text{cmpfrs}(L)$ and $\forall v \in \text{cmpfrs}(L), |u| \geq |v|$. The following is taken from Chandelle (2014); Chandelle et al. (2015). Given a function $f$ and a string $x$, the tails of a string is possible input-
output extensions of \( x \) w.r.t. the function \( f \). It is a set of pairs of strings \((y, v)\) where \( y \) is a possible input extension of \( x \) and \( v \) is the contribution to the output string that \( y \) itself is necessarily responsible for: \( \text{tails}_f(x) = \{(y, v) | f(xy) = uv \wedge u = 1\mathrm{cp}(f(x\Sigma^*))\} \). A function is Input Strictly \( k \)-Local iff there is a \( k \) such that for all \( u_1, u_2 \in \Sigma^* \), if \( \text{suffix}_{k-1}(u_1) = \text{suffix}_{k-1}(u_2) \) then \( \text{tails}_f(u_1) = \text{tails}_f(u_2) \).

For any \( k \)-ISL function \( f \) over domain \( \Sigma^* \), there exists a canonical deterministic single-tape finite-state transducer (IT-FST) \( M \) such that \( f(M) = f \), and every state \( q \in Q \) in \( M \) is labelled with one of the \( k - 1 \) suffixes of \( \Sigma^* \). Transitions are function \( \Delta : Q \times \Sigma \rightarrow Q \times \Gamma^* \). For a state \( q \in Q \) and input symbol \( \sigma \in \Sigma \), \( \delta(q, \sigma) = (p, B) \) such that \( B \in \Gamma^* \) and \( p = \text{suffix}_{k-1}(qa) \).

### 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. We only consider functions which produce one output string, not more.

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 call an \( n \)-STuple, where each word \( w_i \) is made up of symbols from some alphabet \( \Sigma_i \) such that \( w_i \in \Sigma_i^* \). Each alphabet \( \Sigma_i \) may be disjoint or intersecting, so two input strings \( w_i, w_j \) may be part of the same language \( \Sigma_i^* \).

To generalize the notion of suffixes into multiple strings, we define another type of tuple, called an \( n \)-NTuple \( K = [k_1, \ldots, k_n] \) be a tuple of \( n \) natural numbers marked by brackets. Given some \( n \)-STuple \( W \) and a \( n \)-NTuple \( K \), the \( \text{suffix}_k \) or \( k \)-suffix of \( W \) is an \( n \)-NTuple \( V \) made up of the last \( k_i \) symbols on \( w_i \): \( \text{suffix}_k(W, K) = V \) s.t. \( V = [v_1, \ldots, v_n] \) and \( |v_i| = k_i \) and \( x_i, v_i = w_i \) for \( x_i \in \Sigma_i^* \). E.g. for \( W=\text{abc,def} \) and \( K=[2,1] \), \( \text{suffix}_k(W, K) = [bc, f] \). In the rest of this paper, we do not capitalize the name of \( n \)-NTuples as \( K \) but use lowercase \( k \). Contexts makes the distinction clear.

Let \( f \) be an \( n \)-input function defined over an \( n \)-STuple \( I \) of input strings \( I = [I_1, \ldots, I_n] \) taken from the tuple of alphabets \( \Sigma_i \). As an 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 \( I_i \). For any \( k \)-MISL function \( f \), we conjecture that there exists a canonical asynchronous deterministic multi-tape FST \( M \) such that i) \( f(M) = f \), and ii) the states of \( M \) are labelled with the \( k - 1 \) suffixes of \( \Sigma_i^* \).

As of now we are still developing a proof of this conjecture. However, if some function can be computed by an MISL MT-FST, then that is sufficient evidence for the function being MISL.²

### 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 \( \Sigma_i \). The output string is taken from the output alphabet \( \Gamma \). For an input tuple of strings or an \( n \)-STuple \( w = [w_1, \ldots, w_n] = [\sigma_{1,1}, \ldots, \sigma_{1,n}, \ldots, \sigma_{n,1}, \ldots, \sigma_{n,n}] \), 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 \( n \)-STuple \( w \), the MT-FST falls off the right edge

²It is also known what is the exact algebraic definition of an MISL function. The notion of tails from single-input functions can be extended to multi-input functions. Given a \( n \)-input function \( f \) and an \( n \)-STuple \( x = [x_1, \ldots, x_n] \), the tails of the \( n \)-STuple \( x \) is the set of possible input-output extensions of \( x \) w.r.t. the function \( f \), i.e. pairs \((y, v)\) where \( y \) is an \( n \)-STuple and \( v \) is a string. \( y \) is a possible ‘input extension’ of \( x \) and \( v \) is the contribution to the output string that the input-extension \( y \) is necessarily responsible for. \( \text{tails}_f(x) = \{(y, v) | y = [y_1, \ldots, y_n] \wedge f([x_1 y_1, \ldots, x_n y_n]) = uv \wedge u = 1\mathrm{cp}(f([x_1 \Sigma_1^* x_n \Sigma_n^*])) \} \). A MISL function for \( k = [k_1, \ldots, k_n] \) could be defined as function where \( v \wedge n \)-STuples \( a, b \in \Sigma_1^* \times \ldots \times \Sigma_n^* \), if \( \text{suffix}_{k-1}(a, k) = \text{suffix}_{k-1}(b, k) \) then \( \text{tails}_f(a) = \text{tails}_f(b) \). However, because MISL functions involve dependencies across multiple input strings, there are multi-input functions which can be computed by MISL MT-FST’s but have an infinite set of tails. We do not discuss this here for space reasons.
of the n input tape when the FST is in an accepting state after writing u on the output tape, we say the MT-FST transduces, transforms, or maps, w to u. Otherwise, the MT-FST is undefined at w. We illustrate MT-FSTs in §4.

Formally, a n—MT-FST for some natural number n is a 6-tuple (Q, Σ, Γ, q0, F, Δ) where:
- n is the number of input tapes
- Q is the set of states
- Σ = [Σ1, ..., Σn] is a tuple of n input alphabets Σi which include the end boundaries Σi
- Γ is the output alphabet
- q0 ∈ Q is the initial state
- F ⊂ Q is the set of final states
- δ: Q × Σ → Q × D × Γ* is the transition function where D = {0, +1}n is an n-tuple of possible directions to take on each tape
- Δ is the output function

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 T is an element of (Σ* × Σ* × Γ*), short for ([Σ1* × Σ2* × ... × Σn*] × Γ*). The meaning of the configuration c = ([w1q1x1, ..., wnxnxn], u) is the following. The input to T is the n-STuple wx = [w1x1, ..., wnxnxn]. The machine is currently in state q. The read head is on each of the n-input tapes on the first symbol of xi (or has fallen off the right edge of the input tape if xi = λ). u is currently written on the output tape.

Let the current configuration be ([w1q1a1x1, ..., wnxqa nxt], u) and let the current transition arc be δ(q, [a1, ..., an]) = (r, D, v). If D = [0n], then the next configuration is ([w1qa1x1, ..., wnxqa nxn], uw) in which case we write ([w1qa1x1, ..., wnxqa nxn], u) → ([w1qa1x1, ..., wnxqa nxn], u) (= none of the tapes are advanced) . If D = [+1n], then the next configuration is ([w1qa1x1, ..., wnxqa nxn], uw) in which case we write ([w1qa1x1, ..., wnxqa nxn], u) → ([w1qa1x1, ..., wnxqa nxn], uw) (= all the tapes are advanced). Otherwise, the next configuration is ([w1C1x1, ..., wnxCnxn], uw) where C1 = ra if D1 = 0 and C1 = ari if D1 = +1 in which case we write ([w1qa1x1, ..., wnxqa nxn], u) → ([w1C1x1, ..., wnxCnxn], uw) (= a subset of the tapes are advanced).

The transitive closure of → is denoted with →+. Thus, if c → c' then there exists a finite sequence of configurations c1, c2, ..., cn with n > 1 such that c = c1 → c2 → ... → cn = c'.

As for the function that a MT-FST computes, for each n—STuple w ∈ Σ* where w = [w1, ..., wn], fT(w) = u ∈ Γ* provided there exists qf ∈ F such that ([qf × w1 × ..., qn × wn] × λ) →+ ([q1 × qf, ..., qn × qf], u). Otherwise, if the configuration is ([q1 × qf, ..., qn × qf], u) and qf ∉ F then the transducer crashes and the transduction fT is undefined on input w. Note that if a MT-FST is deterministic, it follows that if fT(w) is defined then u is unique.

We conjecture that if a n-input function is MISL, then there exists a corresponding asynchronous Multi-tape FST. Let f be a n-input string function which takes as input the n—STuple w = [w1, ..., wn] defined over the alphabets Σ = [Σ1, ..., Σn]. If f is MISL with k = [k1, ..., kn], then there exists an MT-FST M such that: f(M) = f, M has n input-tapes corresponding to the n input strings; Except for the initial and final states q0 and qf, every state corresponds to a possible k-factor of f.

3 Root-and-pattern morphology in template filling

Semiotic 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. A partial paradigm of Standard Arabic verbs is in Table 1, amassed from many canonical sources (McCarthy, 1981; Broselow and McCarthy, 1983; Bat-El, 2011). To illustrate, the verb *kutub (Table 1a) is morphologically composed of a root *C=kb, vocalism V=ui, and template T=CVCVC which marks locations for consonants and vowels. Its au-
Table 1: Partial paradigm of Arabic root-and-pattern morphology.

| Pattern | Root Vowels | Template | Autosegmental Graph | L-value |
|---------|-------------|----------|---------------------|---------|
| a       | ab          | CVVC     |                     | (1,1,1) |
| b       | ab          | CVVC     |                     | (1,1,1) |
| c       | ab          | CVVC     |                     | (1,1,1) |
| d       | ab          | CVVC     |                     | (1,1,1) |
| e       | ab          | CVVC     |                     | (1,1,1) |
| f       | ab          | CVVC     |                     | (1,1,1) |
| g       | ab          | CVVC     |                     | (1,1,1) |
| h       | ab          | CVVC     |                     | (1,1,1) |
| i       | ab          | CVVC     |                     | varies  |
| j       | ab          | CVVC     |                     | varies  |
| k       | ab          | CVVC     |                     | varies  |
| l       | ab          | CVVC     |                     | varies  |
The bulk of theoretical and psycholinguistic results show that template-filling is a real process (Prunet, 2006; Aronoff, 2013; Kastner, 2016), but the formulation of templates is controversial (Us- sishkin, 2011; Bat-El, 2011). One hypothesis is that the template is composed of CV slots (Mc- Carthy, 1981). Alternatives are that the template is made of prosodic units like moras, syllables, and feet (McCarthy and Prince, 1990a,b), derived from other templates via affixation (McCarthy, 1993), or is a set of optimized prosodic constraints (Kastner, 2010; Kastner, 2016; Zukoff, 2017).

We take a theory-neutral position and focus on the mathematical function behind RPM. Mathematically, RPM is a 3-input function \( I = \{I_1, I_2, I_3\} \) where \( I_1 \) is the root \( C \), \( I_2 \) is the vocalism \( V \), \( I_3 \) 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 \( \{, \times, \} \): \( \Sigma_1 \times = \Sigma_i \cup \{, \times, \} \). The output alphabet is the output segments. Thus mathematically, 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. As 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.

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 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).

#### 4.1 1-to-1 slot-filling

##### 4.1.1 Simple 1-to-1 slot-filling

For \textit{kutib} (Table 1a), RPM shows 1-to-1 slot-filling, meaning the \( c \) 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 \( k = [1, 1, 1] \). 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, we interpret \( [c, \Sigma_x, C] : [+1, 0, +1] : c, \) lowercase 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.
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.4

The representational route is to enrich the template with the affix itself: T=CVVCVC (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: $\lambda : (0, 0, t) : \lambda$ 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, ut, CVVCVC] 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=kab, V=a, T=CVVCVC. 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: $f((sm, a, CVVCVC) = samam$ (Table 1f).5 This is

4A third alternative is to treat the infix <t> as part of a separate input-string or input-tape. The template is CVVCVC where C is pre-associated to <t>. This is analogous to giving each morpheme its own autosegmental tier (McCarthy, 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.

5Since McCarthy (1981), the analysis of final conso-
(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 associated a string-medial vowel or consonant with multiple slots on the T-string: *kautib* with a long-vowel *u* (Table 1g) or *kutib* 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 \( \mu_C \) in \( T=VC\mu_VC \) (Kay, 1987; McCarthy, 1993; Beesley, 1998). With this template, generating *kutib* is (2,1,1)-MISL with \( h_1=2 \) 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 \( \mu_V \) as a special symbol.

A derivational alternative is to derive *kutib* from *kutib* by infixing a consonant mora \( \mu_C \) followed by consonant spreading. Generating the base *kutib* is (1,1,1)-MISL. Infixing the mora \( k\mu_Ctib \) is 4-ISL and spreading the consonant *kutib* is 2-ISL. As with preassociation (§4.1.3), the representational solution is a composition of the derivational solution; both are local functions.

### 5 Possible non-locality in Semitic

Not every Semitic template formation is local, such as: reduplication and loanword adaptation.

#### 5.1 Reduplication

Semitic templates show intensive reduplication which varies on root size (Broselow and McCarthy, 1983): root doubling in for biconsonantal roots in *laflaf* (Table 1i) and first-C copying for triconsonantal roots in *barbad* (Table 1j). 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.\(^6\)

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 \( k \). 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 \( k \). Analogously to subsequential functions over single-input FSTs, root-initial copying would be M-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).

---

\(^6\)Technically, the relevant inputs need to be annotated to trigger reduplication, e.g. initial-C copying with \( T=CVCFVC \) and root doubling with \( C=I-RED \). We abstract away from this for clarity.
we discard this assumption, then both reduplicative processes are MISL for a large value of $k$.

### 5.2 Local spreading in loanword adaptation

In loanword adaptation of verbs in Arabic, the most productive template is CVCCVC with the vocalism a: CaCcCaC (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 1k). But when a borrowed root has three consonants, then the input undergoes medial gemination: SMS and sammas, not final spread *sansas (Table 1l).

There are many ways to analyze this difference between three vs. four-consonantal roots. One is suppletive allomorphy: four-consonantal roots use the template CVCCVC, three-consonantal roots use the template CVCCV. 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: khr → akbar, but VCVC for biconsonantal roots: fd → afadd (Davis and Tsujimura, 2018).

An alternative is to use a template CVCCV without any representational markup for gemination. The correct outputs are generated based on avoiding non-local spreading. For a three-consonantal root, medial gemination is generated because the grammar (in OT-parlance) prefers outputs with local spreading of consonants sansas instead of outputs with non-local spreading sansas. 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).\(^3\)\(^4\) 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: $XCCV,C^k \times$ vs. $XCCV,C^k = |V_s| + 1$ on T-string

The last condition is important. Consider the contrast in kuttib and kutba ‘writers’ derived from the templates $C_1V_1C_1C_2V_4$ and $C_1V_1C_1C_1V_1$. The $C_2C_3$ substring in $C_1V_1C_2C_2V_2C_4$ 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 precede an extra consonant slot $C_4$ on the template or not. If there is no bound on the number of intervening vowels $V_s$, then the function is not MISL for any $k$. If there is a bound, then it is MISL for a $k$ which is sufficiently large enough to encode these contexts. In Arabic, $V_s$ can be at most two vowels slots in order to encode long vowels: kuttaaab ‘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 Oosten-dorp 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. 2015. 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 Tsuji. 2014. Arabic nonconcatenative morphology in construction morphology. In Geert Booij, editor, The Construction of Words: Advances in Construction Morphology, volume 4. Springer.

Hossein 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, Brussels, Belgium. Association for Computational 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 automata survey. Communications of the ACM, 8(12):799–805.

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

Christian 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.

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.