Long Distance Processes in Tone and Subsequentiality

Tajudeen Mamadou Y.

May 13, 2022

Contents

1 Introduction

Phonological patterns are usually treated as transformations, i.e mappings from some underlying and abstract form to a surface form. Rule based theories of phonology (e.g: SPE, Chomsky and Halle 1968), as well as constraint based theories (Optimality Theory, Prince and Smolensky 1993, 2004) commit to this idea as a cognitively real phenomenon. Formal Language Theoretic approaches to phonology also follow this tradition by studying phonological processes as functions. For instance, phonological processes of metathesis, final devoicing, epenthesis (Chandlee, 2014; Chandlee et al., 2012a; Chandlee and Heinz, 2012; Chandlee et al., 2018; Chandlee and Jardine, 2014), vowel harmony (Heinz, 2018; Heinz and Lai, 2013; McCollum et al., 2017), consonant harmony (Luo, 2017), nasal spreading (Chandlee and Jardine, 2019b), morpho-phonological processes of template filling (Hulden, 2009; Dolatian and Rawski, 2020), tone processes (Gibbon, 1987; Jardine, 2016c; Chandlee and Jardine, 2019a; Koser et al., 2019) have been analyzed as functions of input-output mapping. This raises the question: what kind of functions are phonological processes and what can we hypothesize as a possible phonological function?

Interestingly, an overwelming majority of segmental processes are functions with (at most) the property of subsequentiality;[1] that is, functions that are computable with a deterministic finite state machine that consumes symbols from one end going in one direction toward the opposite end (Mohri, 1997; Chandlee, 2014). However, Jardine (2016b) showed that tone processes, unlike segmental ones, are not subsequential over a finite sequence of symbols, i.e over a *string* or a piece thereof (a *substring*), but rather properly regular. The goals of this paper are: (1) to define exactly the kind of functions tone processes are, (2) the role played by representation in that definition and (3) whether the hypothesis put forth in Heinz and Lai (2013) that phonological patterns belong to the subregular class extends to tone processes.

Of particular empirical interest are the unbounded circumambient (UC) processes in which triggers on either sides of a given span of targets can be unboundedly far away from each other. UC processes are non-subsequential over string. Building on Jardine (2020a)'s Melody Local (ML) grammars, we propose a new class of functions – the Input Melody Local (IML) functions – to characterize UC processes and other less complex processes using autosegmental-like representations and formal language theory (FLT). IML functions reinforce the insights of Autosegmental Representations (ARs) although they derive the stipulations of the latter, namely the association lines, the no-crossing, no-gapping and the one-to-many & many-to-one constraints (Williams, 1976; Goldsmith, 1976; Kornai, 1995) from independently motivated properties of the computational apparatus.

The novel contribution here is to extend Jardine (2020b)'s ML grammars, which only deal with phonotactics, to functions. This extension has the advantage of allowing us to study processes from the ML perspective. Doing so, we are not only able to account for a wider range of phonological phenomena but also to provide a representational solution to the problem of non-subsequentiality of long distance processes. The current account also bridges the complexity gap between segmental and tonal processes, making the distinction between the two, ones of representation rather than of computational complexity proper.

The paper is organized as follows: Section §2 gives a background on the important classes of functions for phonological patterns and how they are related to subsequentiality, §3 introduces and defines IML functions. The empirical coverage of IML functions is presented in §4, §5 discusses the implications of the results and §6 concludes.

2 Background

The concept of locality, whether strict or over a tier, plays a key role in phonology because most (but not all) phonological transformations are conditioned by the immediate environment of the phonological primitive (e.g: segment, autosegment, etc) undergoing the transformation. Kenstowicz (1994)'s argument that phonological rules can only count up to two is in support of this idea of

[1] Exceptions to this generalization include vowel harmony in Tutrugbu (McCollum et al., 2017).

boundedness in phonology. In the section below, we follow Chandlee (2014) in presenting locality as a computational property of phonological processes that are bounded within a fixed bracket/window.

2.1 SL grammars and ISL functions

Chandlee (2014) introduced Strictly Local (SL) grammars and presented them as grammars whose defining property is that they can only see within a bounded window of contiguous segments. SL grammars can thus only characterize phonotactic constraints as long as the constraints are enforceable within a fixed-size window. This size is usually referred to as the k value and represents the number of segments or characters present within the scanning window. The diagram in (1) represents an SL grammar for a final devoicing rule that bans voiced obstruents word-finally (e.g: *g#, *d#, *b#). The underbrace delimits a window of size $k = 2$ that moves along the sequences of characters. This window only sees exactly two characters at a time and determines whether they are forbidden[2] or attested[3].

(1) Schematization of an SL grammar

Output:	a	b	a	**d**	#

Unlike local phonotactic constraints, which are characterized by SL grammars, local input-output mappings are derivable with Input Strictly Local (ISL) *functions* (for details, see Chandlee 2014). ISL functions are important in phonology for the simple reason that phonological rules are typically viewed as processes from an input to a corresponding output (we return to this below). Basically, ISL functions take some input character x and output it as y based on other characters located at some bounded distance from x in the input. An ISL function can be thought of as an SL grammar with mappings to an output. The diagram in (2) recasts the final devoicing rule presented in (1) as an input-output function. The function applies right-to-left and when it sees the word boundary symbol # in the input, it outputs # and if immediately after seeing a word boundary character, it sees a voiced obstruent like *d*, it outputs it as *t*, its voiceless counterpart. After that, any other input character is faithfully mapped to itself in the output.

In sum, a process is ISL, if the output of some input symbol is determined based on other input symbols located at some bound distance from the target of the process.

(2) Schematization of an ISL Function

Input:	a	b	a	d	#
Output:					#

Input:	a	b	a	**d**	#
Output:				**t**	#

Input:	a	b	**a**	d	#
Output:			**a**	t	#

...

Input:	a	b	a	**d**	#
Output:	a	b	a	**t**	#

Although SL grammars and ISL functions derive most of the segmental phonology patterns because of their locality, they fail to capture long distance interactions, particularly in tone. To derive these interactions, Jardine (2020b) extended the notion of locality to the melody tier, creating a class of grammars known as the Melody Local (ML) grammars. We briefly present ML grammars below.

[2] As a phonotactic constraint, final devoicing forbids pairs of symbols that are members of some set F = {g#, d#, b#}.

[3] Attested pairs of symbols are members of some set A = {ab, ba, a#, k#, t#, p#}.

2.2 Melody Local grammars

The introduction of ML grammars is based on the insights from the autosegmental phonology (Goldsmith, 1976) that non-local (i.e long distance) tone interactions are local on the melody tier. So, just like SL grammars, ML grammars enforce phonotactic contraints over output symbols but in addition (and unlike SL grammars) they can enforce phonotactic constraints on the melody of the same output (Jardine, 2020b), assuming the Obligatory Contour Principle (OCP) (Leben, 1978; McCarthy, 1986; Odden, 1986, 1988) on the melody tier. Jardine (2020b) derived the melody in ML grammars with a melody function, briefly described below.

The melody function (defined in (3)) recursively applies to each span of tonally-specified TBUs, and returns a single H or L until no span of H or L toned TBU is left. It then takes the empty string (i.e λ, a string of length 0) as its final input and outputs an empty string (nothing). This function can be thought of as enforcing the OCP, retaining only one tone in a sequence of adjacent identical tones. It is also close in spirit to Heinz et al. (2011)'s '*erasing*' function in that the melody function erases all but one in a sequence of adjacent like-tones on the melody tier.

(3) Adapted from Jardine (2020a)

$$\texttt{mel}(w) \stackrel{\text{def}}{=} \begin{cases} \lambda & \text{if } w = \lambda, \\ \texttt{mel}(v)\sigma & \text{if } w = v\sigma^n,\ v \neq u\sigma \text{ for some } u \in \Sigma^* \end{cases}$$

The function in (3) reads as follows: if mel( ) applies to a *string* w, when w (henceforth, timing tier string) is an empty string (i.e λ), the mel( ) function outputs an empty string; but when w equals $v\sigma^n$, where v is a variable representing a substring of any sequence and combination of tone symbols (e.g: LHH), and σ^n is one with a uniform n sequence of Ls or Hs, the function outputs the σ^n part of the input as a single σ. Applying iteratively, the function breaks up v into another substring of the form $v\sigma^n$ and goes through the steps described above again until there is no substring left to break up into yet smaller substrings. Crucially, σ^n has to contain all and only the like-symbols (either Hs or Ls but not both) in a given unbroken stretch. An example is shown in (4) below, with each bolded symbol representing σ, the output of the melody function applying to the σ^n substring. In the example, w = HHHLLLHHL:

(4)

$$\begin{aligned} \texttt{mel}(HHHLLLHHL) &= \texttt{mel}(HHHLLLHH)\mathbf{L} \\ &= \texttt{mel}(HHHLLL)\mathbf{H}L \\ &= \texttt{mel}(HHH)\mathbf{L}HL \\ &= \texttt{mel}(\lambda)\mathbf{H}LHL \\ &= \mathbf{HLHL} \end{aligned}$$

As in the Autosegmental Theory (Goldsmith, 1976), the outputs of the melody function are on a different tier, the *melody* tier. The timing tier strings *and* the newly derived melody tier strings are conjunctively monitored by ML grammars. The melody function finds its motivation in the fact that in traditional Autosegmental Phonology, the tone melodies and sequences of TBUs start out each on a tier of their own, leading to the theory having to make a lot of additional necessary but unmotivated assumptions. One such assumption is that there is a tone-TBU association algorithm, which in turn lead to the controversial phonological directionality of tone-TBU association (Leben 1973; Williams 1976; Goldsmith 1976; Pulleyblank 1986; McCarthy and Prince 1993a; Akinlabi 1996; Zoll 2003, a.o). Using the melody function give us the tone-TBU association lines for free. We return to this point in the discussion.

Furthermore, the intimate relationship between certain tones and selected TBUs (e.g morphemes with underlying tones vs. toneless morphemes in most Bantu languages), reinforces the idea that while tone units may behave independently of segments, they are connected to and maybe inseparable from the latter at some deeply abstract level.[4] As such, an approach that promotes the deep connections of tones to their underlying morphemes while remaining free of additional assumptions, like the

[4] Analyses of tonogenesis as deriving from consonants provide additional support for this view.

association algorithm, is to be preferred over one that does not. This is exactly what the melody function provides us with by deriving the melody tier from the timing tier.

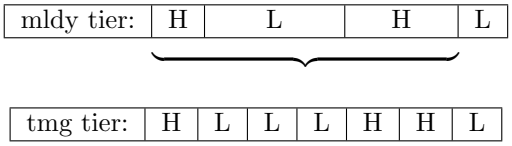
Let's illustrate how ML grammars, with their melody function component, work using the Unbounded Tone Plateauing (UTP) process in Luganda.

(5) Luganda (Hyman and Katamba, 2010; Jardine, 2020a)

a.	/kitabo/	[kitabo]	'book'	LLL
b.	/mutéma/	[mutéma]	'chopper'	LHL
c.	/kisikí/	[kisikí]	'log'	LLH
d.	/mutéma+bisikí/	[mutémá+bísíkí]	'log choper'	LHHHHH
e.	Unattested	*[mutéma+bisikí]	_	*LHLLLH

UTP is a process, illustrated in (5), where only a single unbroken stretch of H tones is attested (Hyman and Katamba, 2010; Jardine, 2016b). That is, HL, HHHLLL, LLLLLH, etc, are attested but not HLH, HLLH, HL^nH, etc. The superscripted n can be any natural number and because of this, UTP is *not* local, at least not over the timing tier. Thus, UTP is not characterizable with an SL grammar. However, since ML grammars can enforce phonotactic constraints over the melody as well, UTP can be derived by banning any HLH sequence on the melody tier. And because any stretch of H or L tones on the timing tier is represented as a single H or L on the melody tier, Hs that are unboundedly far away from each other are guaranteed to be within a bound window from each other on the melody tier. In the case of UTP, that window is of size $j = 3$. The diagrams in (6) below show how ML grammars enforce local phonotactic constraints over the melody tier (using a window of size j=3).

(6) Schematization of ML grammars (UTP)



ML grammars can characterize multiple surface phonotactic generalizations including but not limited to long distance phonological interactions (as seen with UTP). However, just like SL grammars, they treat phonological patterns as phonotactic constraints and not as the transformations they are widely accepted to be. In the section below, we discuss phonological patterns that represent a challenge to ML grammars.

2.3 Non-ML Tone Patterns

ML grammars are not only limited to enforcing phonotactic contraints, they also leave out some basic phonological patterns because these patterns either refer to some position in the input or have an exception to a generalization. We call the former the *Input Problem*, which we exemplify with tone shift in Rimi and the latter, the *Exception Problem*, exemplified with the non-assertive tone patterns in Karanga Shona.

We first turn to the Input Problem. In Rimi (Bantu, Tanzania; Meyers 1997; Breteler 2018), H tone shifts one TBU to the right of its underlying position[5] as shown in (7). The string representation of the examples in (7) are given in (8). Tone Bearing Units (TBUs) in Rimi can only have H or L tones; the underlined syllables and tone symbols in the examples represent the input positions of the shifting H tone.

(7) Tone Shift in Rimi (Meyers, 1997; Jardine, 2016b)

a.	/rá-mu-ntu/	[ra̲-mú-ntu]	'of a person'
b.	/u-púm-a/	[u-pu̲m-á]	'to go away'

[5]This is one of several tone rules in Rimi, including but not limited to a H-tone deletion rule in /...H_1 H_2.../ environments. For a full discussion of these rules, see Meyers (1997).

(8) String Representation of Rimi
a. $HLL \rightarrow \underline{L}HL$
b. $LHL \rightarrow L\underline{L}H$

The H tone shift in Rimi is clearly local because it only relies on the information in a local window of length 2. Any timing tier input /HL/ will always surface as [LH]. An analysis based on well-formedness would motivate this pattern with a constraint against HL sequences, leading to the conclusion that the Rimi pattern is melody-local. However, this is not accurate because the *HL substring is actually attested in Rimi as shown in (8a).

Describing surface forms in Rimi misses the generalization about the need to know the underlying position of the tone that undergoes the shift. As such, *HL will be forbidden if and only if (iff) the underlying form under consideration is /#HL#/ but not if the underlying form is /HLL/, in which case the attested surface string [LHL] will contain the wrongly banned *HL substring. It falls out that the inability of ML grammars to derive the Rimi pattern is due to the fact that the pattern explicitly refers to the position of the H tone in the input, an information that ML grammars don't have access to because they are surface oriented. This observation unveils the limitations of expressing this pattern as a phonotactic constraint and provides and initial motivation for its treatment as a process of input-output mapping, i.e as a function.

(9) H-tone pattern (adapted from Odden 1984 and Jardine 2020b)

a.	handáka-pá	'I didn't give'	H
b.	handáka-tóra	'I didn't take'	HL
c.	handáka-tóresá	'I didn't make take'	HLH
d.	handáka-tóréserá	'I didn't make take for'	HHLH
e.	handáka-tóréseraná	'I didn't make take for each other'	HHHLH
f.	handáka-tóréséresaná	'I didn't make take a lot for each other'	HHHLLH
g.	handáka-tóréséresesaná	(same as f.)	HHHLLLH

(10) L-tone pattern (idem)

h.	handá-pa	'I gave'	L
i.	handáka-biká	'I didn't cook'	LH
j.	handáka-bikísa	'I didn't make cook'	LHL
k.	handáka-bikísíra	'I didn't make cook for'	LHHL
l.	handáka-bikísísira	'I didn't make cook for each other'	LHHLL
m.	handáka-bikísísirana	'I didn't make cook a lot for each other'	LHHLLL
m.	handáka-bikísírisisana	(same as l.)	LHHLLLL

A slightly different input problem, termed the *Exception Problem* arises in Karanga Shona, another Bantu language (Odden, 1984; Myers, 1987; Hewitt and Prince, 1989; Jardine, 2020a). Put briefly, Karanga Shona has non-assertive verbs, which can either be H- or L- toned as shown in examples (9) and (10). H-toned assertive verbs always start with either a single H or a series of two or three H tones, and end in a single H with possibly an unbounded number of Ls between the initial sequence of Hs and the final H. Like the UTP pattern, the Karanga Shona pattern is not local over the timing tier, but is over the melody. To capture this pattern with ML grammars, one will need to ban *HL from the melody because non-assertive verbs have to end in an H.

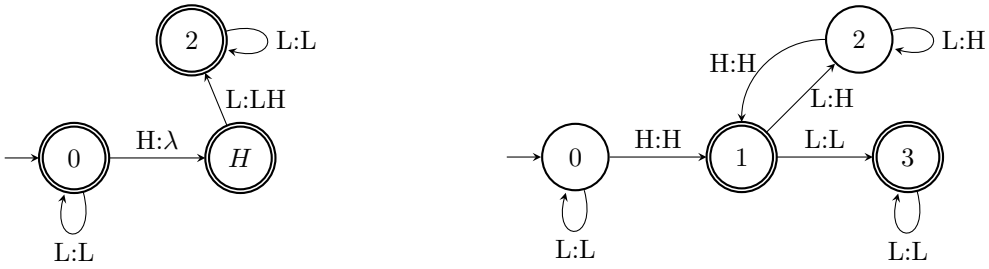
However, the problem with banning *HL from the melody is that dissyllabic stems can be #HL#, thus have a melody of the form HL. This means that banning *HL will systematically exclude the attested dissyllabic stems with an HL melody, as noted in Jardine (2020b). This pattern then presents a dilemma for the ML grammars, which can either ban HL melodies all together, whereby banning the attested #HL# in words with exactly two syllables or not banning HL, whereby allowing a forbidden form like #HLL to surface. Hence, the Karanga Shona non-assertive verb patterns are not ML grammars describable, at least not entirely (Jardine, 2020b).

All in all, the discussion in this section further warrants the need for a class of functions that can leverage the insights of ML grammars while covering more empirical grounds. In the section below, we discuss the notion of subsequentiality, a computational measure of complexity and show how certain tone patterns do not have that property, at least not over strings.

2.4 Non-subsequentiality of Complex Tone Patterns

The notion of subsequentiality (Schützenberger, 1977; Frougny and Sakarovitch, 1993) is a measure of computational complexity for whom deterministic computation is a defining property. A computation is said to be deterministic when at any point in reading the string, there is only one decision the machine can make with respect to the output; it is non-deterministic otherwise. To illustrate the difference between determinism and non-determinism, consider the Finite State Transducers (FSTs) in Figure 1 below, where the arrows represent the transitions. Each transition is labelled with tone symbols, where the one before the colon is the input and the one after the colon is the output of the transition. λ is an empty string.

That being said, Figure 1a[6] , which represents the Rimi example in (7), is deterministic because no transition out of a given state has the same input symbol as another transition out of the same state. In contrast, any L after the first H in Figure 1b (representing the UTP example in (5)), can either be output as L and take the machine from state 1 to state 3 or as H, which takes the machine to state 2. Since the FST does not have any way of knowing if there is a second H coming up, it makes both output options available. If the machine takes the transition to state 2, it must see another H for the computation to be successful because state 2 is not an accepting state (i.e it does not have the double circles), which is where a computation must end for it to be successful. On the contrary, if the machine takes the transition to state 3, it must not see another H in that state, otherwise the computation will crash. Intuitively, the FST in Figure 1b says the following: when Ls following an H are output as H, there must be another H further in the string but when that L is output as L, no other H must follow in the string. If any of the two conditions does not hold, the computation crashes. Crucially, the availability of two output options out of state 1 (for an input L) is exactly what it means for a machine to be non-deterministic and as a consequence any corresponding computation is said to be non-subsequential.



(a) A Deterministic FST.

(b) A Non-Deterministic FST.

Figure 1: A Deterministic FST for Rimi (left) and a Non-Deterministic one for Luganda (right).

Subsequentiality can intuitively be thought of in terms of bounded *look-ahead*, a less restrictive notion than phonological *myopia* (Wilson, 2003, 2006). In computational phonology, Subsequentiality denotes a restrictive thus non-complex class of phonological maps, whose triggers are either to the left (left-subsequential) or the right (right-subsequential) of the target(s) in a given domain but never on both sides. Subsequential functions thus read symbols from one end, going in one direction toward the opposite end. The diagrams in (11) below show the two forms of subsequentiality and the boxed *a* is the trigger. The diagram in (12) on the other hand, showcases Non-subsequentiality, a result of the combination (composition) of left- *and* right-subsequentiality[7].

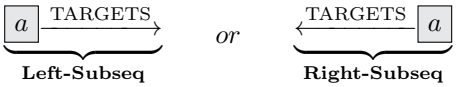
Unlike with subsequential processes, in which strings are computed in one direction, non-subsequential processes require the string to be computed in both directions simultaneously. That is because non-subsequential processes require information that can be unboundedly far away from each other. In

[6] As mentioned above, the tone shift in Rimi is only one of several tone processes in that language. Focusing on tone shift, the assumed Rimi function is one that is partial, i.e only defined for strings with maximally 1 H tone (e.g: LLLLL, LH, LLH, LLLHLLL). As a consequence, the FST in 1a can only compute partial functions defined on this subset of all possible strings in Rimi.

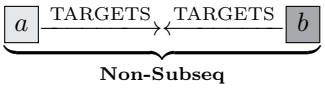
[7] This is because subsequential functions are not closed under composition (Elgot and Mezei, 1965), meaning composing a left- and right-subsequential function may yield a non-subsequential function.

a trigger-target context, non-subsequentiality thus requires an unbounded look-ahead (in both directions) to see whether or not the second trigger/blocker is present before the process applies. This is the case, among others, of the UTP pattern presented in details in §4.3.1.

(11) **Subsequentiality**



(12) **Non-subsequentiality**



Subsequentiality is a way to talk about the computational properties of processes as functions such that subsequential functions require less computational power than non-subsequential ones. Due to their transformational nature, functions can be subsequential if they are deterministic in one direction. The task undertaken in this paper is thus to characterize processes that would otherwise require two subsequential functions going in opposite directions (due to the process' referencing of information that are unboundedly far away from each other on both sides of a span of targets) with a single (unidirectional) subsequential function.

For this purpose, we propose to extend the subsequential property of the diagrams in (11) to account for processes with the non-subsequential property of the diagram in (12). Such an extension of subsequentiality to the non-subsequential UC processes can be achieved by introducing the notion of melody in the representation, an idea that leverages on the autosegmental theoretic insights that non-local tone processes on the timing tier are local over the melody tier (Graf and Heinz, 2015; Jardine, 2020a). This is precisely what the Input Melody Local (IML) functions aim to achieve: to use an enriched representation – one that includes the melody – for the purposes of reducing the computational complexity of the functions needed to derive complex processes.

3 Input Melody Locality

To achieve the need to extend subsequentiality to non-subsequential processes, we introduce the (IML) functions, which use insights from Jardine's ML grammars, but differ from the latter in that IML are functions, meaning they have an output to them. As such, just like the ML grammars, two ingredients are necessary for the IML functions to work: the melody function (same as the ML grammars') and an input-output function.

3.1 IML Functions

IML functions mimic input-oriented phonological processes in the sense that they only act over inputs to produce surface forms. They combine symbols from both the input timing tier and from the derived melody tier, which justifies their *Input-Melody*ness. The locality aspect of these functions comes from the fact that they can only see a finite number of symbols at any given time on both the melody and timing tiers.

IML functions have two main components: the *melody function* and the *input-output* function. As described in (3), the melody function takes as input, strings of tonally specified TBUs[8] from the timing tier (represented with tone symbols) and outputs a melody string, which is on a new tier called the melody tier. The input-output function takes as input, a combination of symbols from both the timing and the melody tiers and outputs a single surface tone symbol, then takes the next symbol on each of the two tiers and outputs yet another single surface tone symbol until the entirety of the substrings on each tier are fed to the function. A concrete example will be given later.

[8]Note that pre-association is assumed, which is independently needed in the analysis of long distance processes.

The Input-Output function may consume a symbol, say from the timing tier while consuming no new symbol (i.e staying put) on the melody tier or vice-versa. This property translates to *asynchronicity* in automata theory terms (which we turn to below) and allows for look ahead on either tiers. The output of each combination of input symbols from each tier can be the empty string λ or at least one tone symbols (e.g: H, LL, HL, etc). To see how the Input-Output function works, let's return to the Luganda UTP pattern whose AR is shown in Figure 3, where UTP applies in Figure (a) but not in (b).

(a) UTP (b) No UTP

Figure 3: AR of the Tone Plateauing applying (left) and not applying (right).

For a reminder, the UTP pattern is one in which only a single unbroken stretch of H tones is attested on the surface. A derivation of the AR in Figure 3a is shown in the diagram in (13) in IML terms.

(13) Schematization of an IML Function for UTP

Input	`mel`(w)	L	H	L			H
	w	L	H	L	L	L	H
Output	output	L	H	**H**	**H**	**H**	H

What makes UTP non-local is that one can arbitrarily increase the number of Ls between the Hs as in the diagram in (14), where we added two more Ls to our input and we get a similar output to the one in the above diagram but with two extra H symbols in the output (also bolded).

(14) Schematization of an IML Function for UTP (extended)

Input	`mel`(w)	L	H	L					H
	w	L	H	L	L	L	L	L	H
Output	output	L	H	**H**	**H**	**H**	**H**	**H**	H

Note that no matter how big the number of Ls between the Hs gets, the number of characters in the melody stays constant. This constancy in the melody is what guarantees that UTP, and for that matter any long distance process, will always be local over the melody. To put it more clearly, IML functions take some input characters combination of the type $X_m|Y_t$, where X_m is from the melody tier and Y_t from the timing tier, and output the combination as Z based on the characters located at some bounded distance either from the melody or timing tier character in the input. In other words, the output of any given IML function is dependent upon characters that are local either to the input character from the melody tier or to the one from the timing tier. So, an IML function can, again, be thought of as an ML grammar with mappings to an output, just like an ISL function and an SL grammar in §2.1,.

Just like other classes of functions, IML functions can be partial or total. As partial functions, they are only defined for some (but not all) elements in their input domain; in other words, there are input strings for which the functions are undefined. An IML function is total when it is defined for all strings in that input domain. Unless otherwise stated, the IML functions introduced in the analyses to follow are by default total functions, i.e defined for all strings in the function's input domain Σ^*, which corresponds to all possible inputs from a given natural language.

The function in (15) characterizes the UTP process described above and reads as follows: inputs with more than one H tone span (i.e $L^mH^nL^oH^n$) are output with a single H tone span (i.e $L^mH^{(2n+o)}$); those with a single stretch of H or without any H in the input are faithfully output as w; that is, when w is a string of only Ls or Ls and a single H. Note that the function is only defined for inputs with at most two H spans in it and not for inputs with three or more H spans. In this sense, the UTP function in (15) may be seen as a partial function or as a total function that only takes substrings as inputs. Under the latter view, it can compute any Luganda input including but not limited to those with more than two H spans, so long as it computes chunks (i.e substrings) of maximally two H spans at a time.

(16a) examplifies a case where UTP actually applies while (16b) represents the elswhere condition. For simplicity, examples like the one in (16) will be used throughout the paper instead of and as a shorthand for the formulation in (15).

(15) UTP Function

$$f_{utp}(\langle \texttt{mel}(w), w\rangle) \overset{\text{def}}{=} \text{L}^m\text{H}^{(2n+o)} \quad \text{if } w = \text{L}^m\text{H}^n\text{L}^o\text{H}^n,\ \texttt{mel}(w)\text{=(L)H(L)H};\ m \,\&\, \text{o}\geq 0,\text{n}= 1$$
$$\overset{\text{def}}{=} w \quad \text{elsewhere.}$$

(16) Examples of UTP Function

a. $f_{utp}(\langle LHLH, LHLLLH\rangle) = \text{LHHHHH}$
b. $f_{utp}(\langle LHL, LHLLLL\rangle) = \text{LHLLLL}$

Additionally, we want the IML functions to allow us to characterize tone patterns that refer to input positions – as is require in tone shifting processes. Such processes, whether bounded as in Rimi or unbounded as in Zigula, are straightforwardly captured by IML functions. Let's examplify this with the bounded tone shift in the Rimi data introduced in (8). The Rimi function correctly characterizes the tone shift (or tone reassociation process as Breteler (2018) calls it). The actual tone shift is illustrated in (17a) and since the Rimi tone shift predicts that a domain final H will not shift, we illustrated that scenario in (17b). The underlining indicates the input position of the shifted H tone.

(17) Tone Shift in Rimi

a. $f(\langle LHL, LLHLLL\rangle) = \text{LL}\underline{\text{L}}\text{HLL}$
b. $f(\langle LH, LLLLLH\rangle) = \text{LLLLLH}$

Now that the IML functions have been illustrated, it is worth mentioning that the two input tiers that compose them play two different roles in getting the right output. On the one hand, the melody tier input allows for a bounded look-ahead to see whether another trigger/blocker is coming up, a look-ahead that would otherwise be unbounded on the timing tier. On the other hand, the timing tier input ensures that the linear order of the positions occupied by the tone symbols is preserved in the output. Finally, a process may be left-IML (L-IML) or right-IML (R-IML). A process is L-IML if it is computed left-to-right and R-IML if it is computed right-to-left. Another way of conceptualizing the difference between the two is to think of an L-IML function as a left-subsequential function over a representation that includes the melody and a R-IML as its mirror image. Right- and Left-IML are standard variants of IML machines, just like Right- and Left-subsequential are standard variants of subsequential machines (Mohri, 1997).

That being said, we turn, in the following section, to the question of how IML functions can be computed using a specific type of Finite State Transducer.

3.2 Computing IML Functions with Finite State Transducers

In automata theory, a machine or automaton is said to compute a function f when that machine takes arguments of f and returns values of f as outputs (Copeland, 1996). In the present case, the functions to be computed are the IML functions introduced in the previous section and the machine to compute them are the Finite-State Transducers (FSTs). FSTs are often used to compute

phonological transformations because they offer a straightforward correspondence with subregular classes of functions, classes that have been shown to be important for phonology and whose expressivity can be derived from certain properties of the FSTs. For our purpose here, we'll focus on Multi-tape Finite State Transducers (MT-FSTs), a type of FSTs in which the machine has two *read* heads, each of which is on a different input tier, called *tape*.

IML functions can be computed with left-to-right or right-to-left (i.e one-way) MT-FSTs and not with the less expressive one-way single-tape FSTs[9] because as shown above, IML functions take a combination of strings from two different tiers, the timing tier and the melody tier.[10] Each of these tiers constitutes a tape from which the MT-FST reads its inputs. The melody function introduced in (3) is computed by a single-tape FST which takes any string as input and outputs the melody of that string; so, IML-computing MT-FSTs can be thought of as reading from both the melody function's input tape on the one hand and output tape on the other. IML functions are computable with exactly 2-Tape Deterministic FSTs.

Deterministic Multi-Tape Finite-State Machines (DM-FSTs) are MT-FSTs with the property of determinism. As seen above, a finite-state machine is said to be deterministic when there is maximally one transition out of any single state of the machine for every readable input in that state. Deterministic FSTs are less expressive than their non-deterministic counterparts and they correspond to the subregular class of subsequential functions (Chandlee et al., 2012b).

Formally, a DM-FST is a tuple $T{=}\langle \Sigma, \Gamma, Q, q_0, q_F, \Delta, \omega \rangle$ where:

- Σ and Γ are the input and output alphabets, respectively; where $\Sigma{=}\{$H,L,+$\}$ and $\Gamma{=}\{$H,L$\}$, meaning the input string of the MT-FSTs can only have H and L as symbols and the output string H, L and + as symbols.
- Q is the finite set of states; $q_0 \in Q$ is the single initial state (i.e where the computation begins), and $q_F \subseteq Q$ is the set of final or *accepting* states (i.e where successful computations end);
- $\Delta \subseteq Q{\times}\Sigma \times \Sigma \times \Sigma^{*}{\times}$Q is the finite set of transitions. A transition $\delta =$ (p, X|Y, Z, q) $\in \Delta$ is represented as $p \xrightarrow{X|Y:Z} q$, where $X|Y \in \Sigma \times \Sigma$ (or Σ^2). Intuitively, this means that when the transducer is in state p and the next input symbol is X on the melody tape and Y on the timing tape, the machine goes to state q and outputs Z.
- $\omega : q_F \times (\Sigma^n \cup \{\rtimes, \ltimes\} \cup \{\lambda\}) \rightarrow \Gamma^{*}$ is the terminal *output* function; where Γ^{*} is the set of output strings. The input alphabet Σ is extended with $\rtimes$ and $\ltimes$, respectively the left and right phonological domain boundary symbols;[11] and the λ, which can be thought of as the movement instruction meaning 'stay put' or 'pause'. Σ is the alphabet and the superscripted n represents the number of tapes the machine reads from – which is 2 in our current cases.

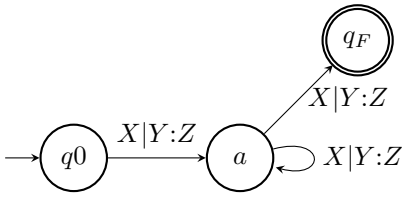


Figure 4: A Toy DM-FST.

[9] We follow Rawski and Dolatian (2020) in designating by single-tape FST, an FST with a single input tape and MT-FST, one with multiple input tapes.

[10] It is not clear how much more expressive MT-FSTs are, compared to the single-tape FSTs. Filiot and Reynier (2016) established that one-way single-tape FSTs correspond to 'rational functions' (known as 'regular functions' among American computer scientists (Dolatian and Rawski, 2020)). So, it is still an open question what the exact expressivity of multi-input functions like the IML ones and their corresponding one-way MT-FSTs is. However, whatever their expressivity turn out to be, the restrictions listed in (18) for the IML MT-FSTs can only reduce it.

[11] These symbols mark the beginning and end of the relevant phonological domain.

Schematically, the DM-FST described above is shown in Figure 4, where the circle labelled $q0$ is the initial state where the computation begins, a is some intermediate state (or states) and q_F is the final, accepting state, which is the state (or states) the machine should be in at the end of any successful computation; the computation is said to crash otherwise. The arrows between the states are the transitions and their label is a function that takes a combination of X and Y, respectively melody and timing tier variables which range over the input alphabet. For our purpose in this paper the alphabet in question is the set $\Sigma = \{$ H, L, + $\}$. The '+' symbol represents a morpheme boundary. The function's output is Z, a variable ranging over the output alphabet $\Gamma = \{$ H, L $\}$. While computing a given string, the machine takes the transition for which the associated function has an input that matches the symbol that the machine is currently reading in the input string. For example, if the machine is currently in state $q0$ and reads X on the melody and Z on the timing tier, it will take the transition labelled $X|Y : Z$ to state a. It is important to note that not all transitions take the machine out of its current state; sometimes the machine stays in the same state, we say it *loops*, which we represent with an arrow that both leaves and points to the same state as shown in state a.

That being said, a function is IML if it is computable by a DM-FST with the properties in (18). The properties are to be thought of as conditions that a DM-FST must meet to be said to compute an IML function. In other words, a function that is necessarily computed with a DM-FST that violates any of the conditions in (18) is not IML.

(18) Properties of an IML-computing DM-FST

a. For any transition with an input $(q, \sigma_1, ..., \sigma_n)$, where σ is a symbol in Σ, at least one $\sigma_i \neq \lambda$ (both inputs symbols cannot be empty strings);

b. For any two transitions $(q, \sigma_1, ..., \sigma_n)$ and $(q, \sigma'_1, ..., \sigma'_n)$, i.e two transitions out of the same state, it must be the case that for at least one tape i, $\sigma_i \neq \sigma'_i \neq \lambda$ (transitions cannot differ only by replacing λ)[12];

c. The read-heads on both tapes move in the same and one direction asynchronously (the read-heads on each tape may advance at different times)[13] ;

d. Each state of the machine corresponds to the j-1 and k-1 factor, where j and k are the fixed, largest input window size, respectively on the melody and the timing tapes, that needs to be scanned for the machine to decide what a given input symbol is to be output as. In other words, when the machine moves to a given state, it means that the machine has maximally scanned j-1 symbols on the melody and k-1 symbols on the timing tiers of the input.[14] (a state should neither represent more than j-1 and k-1 input symbols, respectively on the melody and timing tapes nor j-1 and k-1 output symbols); and finally,

e. Both the melody and timing tiers share the same input alphabet (the string on a given tape may not contain a symbol that is not on the other tape).[15]

The properties in (18) are similar to (though not the same as) the ones adopted in Rawski and Dolatian (2020) and Dolatian and Rawski (2020) to compute their Multi-Input Strictly Local (MISL) functions for Templatic Morphology and tone processes. However, the restrictions imposed on the MISL-computing DM-FSTs, though necessary, are not sufficient because the IML functions do not need the full power of the MISL-computing DM-FSTs, which need to be further constrained. The proposed additional constraint in this regard is in (18e) and requires that the melody and timing tapes have a shared alphabet. The fact that the melody tape is derived from the timing tape in IML functions guarantees that a DM-FST with this additional restriction will compute exactly IML functions. Such machine can not derive templatic morphology, which crucially requires the tapes to have different alphabets, namely an alphabet of consonants on the root tape, of vowels on the vocalism tape and of

[12]This avoids us the *non-determinism* introduced by two transitions of the form $\lambda \mid H$ and $H \mid H$, for example.

[13]If the machines were to be synchronous, they would suffer from what is known as state explosion because the larger the distance between the trigger and the target in the computation of a long distance process, the larger the number of states the machine will need to have. 1-tape FSTs have a similar problem with long distance processes (Hulden, 2009).

[14]It goes without saying that a state may represent a window size smaller than j-1 and/or k-1 simply because a process derived by scanning a given window can be derived by scanning a larger window, but not vice versa.

[15]The melody function guarantees that this property will always hold, but the reason it needs to be explicitly stated here is to prevent the insertion of new symbols during the computation.

template slots on the template tape. This suggests that IML functions constitute a more restrictive class than the MISL class; thus the right fit for phonological functions. To see how a DM-FST computes functions for an actual process, let's begin by showing a DM-FST for the *unbounded tone shift* to the penultimate vowel in Zigula.

As in many Bantu languages, Zigula (Bantu, Tanzania; Kenstowicz and Kisseberth 1990) roots can be toneless or H-toned. In the latter case or whenever there is a H tone in the word, the H tone shifts to the penultimate vowel, regardless of how far away this vowel is from the underlying position of the H tone as shown in (19) (Kenstowicz and Kisseberth, 1990).

(19) Unbounded Tone Shift in Zigula (adapted from Kenstowicz and Kisseberth 1990)

	Input	**Gloss**	**Output**
a.	/ku-gulus-a/	'to chase'	[ku-gulus-a]
b.	/á-gulus-a/	'(s)he is chasing'	[a-gulús-a]
c.	/ku-lómbez-a/	'to ask'	[ku-lombéz-a]
d.	/ku-lómbez-ez-an-a/	'to ask for each other'	[ku-lombez-ez-án-a]

This shift will be referred to as an unbounded shift (Jardine, 2016a). There are further complications when there is a second H tone, namely when an H-toned root is prefixed with the third person pronoun *a-* 's/he' or *wa-* 'they' (which underlyingly bear a H tone), thus we'll only focus on the unbounded tone shift (Kenstowicz and Kisseberth, 1990).[16] In the examples in (19) (and the ones thereafter), the acute accents represent H tone and the absence thereof represents tonally underspecified TBUs in the input and L tone in the output, both of which are represented with the symbol L throughout our analysis. This representational choice is not to obscure the distinction between tonally underspecified TBUs and underlying L tones, which is a well-motivated distinction in Bantu analyses. In fact, we could add $\emptyset$ to our input alphabet to represent tonally underspecified TBUs and only use L for underlyingly L-tones TBUs; the IML analysis will still work just fine. So, the choice to use L for both underlying TBU statuses is merely for simplicity purposes. (20) shows a string representation of the examples in (19).

(20) String Representation of the Unbounded Tone Shift in Zigula

	Input	**Melody**	**Output**
a.	LLLL	L	LLLL
b.	HLLL	HL	LLHL
c.	LHLL	LHL	LLHL
d.	HLLLLL	HL	LLLLHL

Because we are only focusing on the unbounded shift process, the Zigula melody function f_{zmel} in (21) is a partial function in the sense that it is only defined for inputs with at most a single H tone.[17] This means that the domain of the melody function is a proper subset L_{zmel} of all possible inputs Σ^* ($L_{zmel} \subset \Sigma^*$), such that L_{zmel}={L, LL, ..., L^n, H, HL, LHL, HLL, LHLL, LLHLL, ..., L^nHL^n}, from the alphabet Σ={H, L}. The Zigula Input-Output (IO) function, f_{zIO} is defined over a domain made of pairs of elements, i.e pairs of the melody function f_{zmel}'s input and output: f_{zIO}={(L,L),(L,LL), (L,LLL),...,(L,L^n),(LH,LH),(LH,LLH),...,(LH,L^nH),(LHL,LHL),(LHL,LLHLL),(LHL,LLLHLLL),..., (LHL,L^nHL^n) }; where $n \geq 0$.

(21) Zigula Melody Function Examples

a. f_{zmel}(LHLLLL) = LHL
b. f_{zmel}(LLLLLL) = L

[16] When there are two H tones in the word, three additional processes are observed: a one-step rightward shift, then spreading of the prefix's H tone, and a L tone insertion between the spreading first H and the shifted second H, enforcing the OCP. According to Kenstowicz and Kisseberth (1990)'s analysis, both the prefix and the root H tones first shift one TBU to the right before the latter shifts to the penult and the former spreads unboundedly up to (but not including) the TBU that precedes the shifted root's H tone; Kenstowicz and Kisseberth (1990) further argued that the antepenult TBU - which did not get spreaded over - surfaces with a 'buffer' L, preventing an OCP violation. In dissylabic roots where the two H tones surface on adjacent TBUs, the second one is downstepped. The details of these other processes have no bearing on our current analysis since we're only focusing on the unbounded tone shift.

[17] This restriction on the domain is melody local and subregular (Jardine, 2020b).

As mentioned above, it is the input and the output of the melody function f_{zmel} exemplified in (21) that are simultaneously used as inputs for the Zigula IO function in (22), where the string before the coma is the melody, i.e the output of the Melody Function and the one after is its input, i.e the initial string.

(22) Zigula IO Function Examples
   a. $f_{zIO}(\langle LHL, LHLLLL\rangle)$ = LLLLHL
   b. $f_{zIO}(\langle L, LLLLLL\rangle)$ = LLLLLL

For simplicity, examples like the one in (22) above will be used throughout the paper instead of and as a shorthand for the more generic formulation in (23), where $\texttt{mel}(w)$ is the melody string and w is the initial string.

(23) Zigula IO Function

$$f_{zIO}(\langle \texttt{mel}(w), w\rangle) \stackrel{\text{def}}{=} \mathrm{L}^{(m+o)-1}\mathrm{H}^n\mathrm{L}^n \quad \text{if } w = \mathrm{L}^m\mathrm{H}^n\mathrm{L}^o \text{ and } \texttt{mel}(w)\text{=(L)HL, where m} \geq 0, n = 1, o \geq 1;$$
$$\stackrel{\text{def}}{=} w \quad \text{elsewhere.}$$

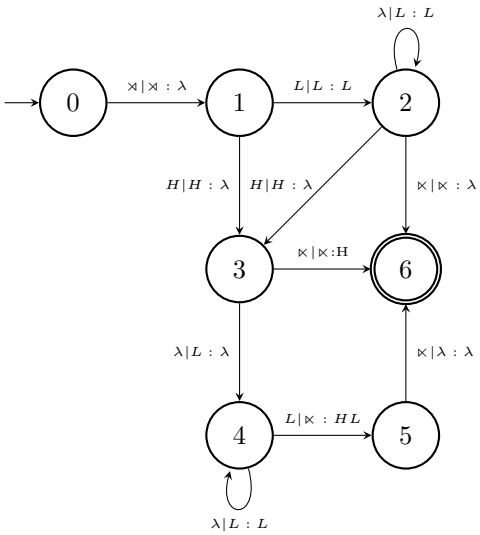


Figure 5: A 2-tape DM-FST for Unbounded tone shift to the penult in Zigula.

The FST in Figure 5 computes the Zigula function by shifting the H tone all the way to the penult. The *input* strings, which are to the left of the colon are symbols from the melody (on the left of the vertical line) and from the timing tier (on the right of the vertical line). The *output* string is to the right of the colon. State 0 is the initial state; state 1 is the left-delimiters' state and state 5 the right-delimiters' state. States 2 and 3 are L tone and H tone states, respectively. States 4 is a wait state on the melody tier but not on the timing tier. Finally, state 6, unlike the other states, is an accepting state, meaning a successful computation of a string ought to end in that state.

A sample derivation of how the Zigula FST computes the example in (22a) is broken down as in the Tables below. In this derivation, the machine starts out in state 0 where it reads the left boundary symbols on both tapes (i.e ⋊|⋊) and outputs an empty string (i.e λ) taking the transition to state 1.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	⋊LHL⋉	⋊LHLLLL⋉	⋊|⋊:λ	*q1*	

In state 1, it reads L both on the melody and timing tapes and outputs an L, while transitioning to state 2, where it reads $H|H$ and outputs λ, taking the transition to state 4.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
2.	*q1*	⋊L̲HL⋉	⋊L̲HLLLL⋉	L|L:L	*q2*	L
3.	*q2*	⋊LH̲L⋉	⋊LH̲LLLL⋉	H|H:λ	*q3*	L

At this stage, the machine stops moving on the melody tape; that is, it stops consuming symbols of the melody tape but continues to consume those on the timing tape, outputting an L each time. As far as the transition is concerned, the machine temporarily stops taking transitions out of state 4 and keeps looping in state 4.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
4.	*q3*	⋊LH̲L⋉	⋊LHL̲LLL⋉	λ|L:λ	*q4*	L
5.	*q4*	⋊LH̲L⋉	⋊LHLL̲LL⋉	λ|L:L	*q4*	LL
6.	*q4*	⋊LH̲L⋉	⋊LHLLL̲L⋉	λ|L:L	*q4*	LLL
7.	*q4*	⋊LH̲L⋉	⋊LHLLLL̲⋉	λ|L:L	*q4*	LLLL

The machine loops until it reads an L on the melody and the right boundary symbol on the timing tape. When this happens, it outputs HL and transitions to state 5, where it reads the boundary symbol on the melody tape and because it has already read a boundary symbol on the timing tape, it can only stay put on that tape. This is why the input to the last transition is ⋊|λ, outputing λ and taking the machine to the final state 6.

Step	Current state	Melody tape	Timing tape	Transition	Dest. State	Output
8.	*q4*	⋊LHL̲⋉	⋊LHLLLL̲⋉	L|⋉:HL	*q5*	LLLLHL
9.	*q5*	⋊LHL⋉̲	⋊LHLLLL⋉̲	⋉|λ:λ	*q6*	LLLLHL
10.	*q6*	⋊LHL⋉	⋊LHLLLL⋉			LLLLHL

When all the outputs of the machine are collected while preserving the order in which those outputs occurred, we get the surface $\lambda L\lambda\lambda LLLHL$ and dropping the λs (because they are empty strings) we get the expected surface $LLLLHL$. Note that it is crucial that the output strings be kept according to the time of their occurrence because this is what guarantees the right linear order of the output.

The machine is IML because it has all the properties of an IML DM-FST listed in (18). There are no transitions in the machine in Figure 5 where both the melody and the timing tier symbols of the input are empty, i.e λ, which complies with the property in (18a). Next, note that in all the states that have more than one transition out of them (e.g: States 1, 2 and 4), there is no instance where two transitions out of the same state have identical inputs or inputs that are only different in one having a λ where the other has another symbol. The absence of such instance is in line with the property in (18b). The property in (18c) can be seen in the fact that some transitions of the machine in Figure 5 have λ either as their melody symbol or their timing symbol. Since λ in any input tape means that the read-head on the tape in question has paused while the read-head on the other tape is moving, we can see in the transition out of state 3 that the melody input has λ while the timing tier input has L. It is the other way around in the transition out of state 5. Note that the property in (18c) is still satisfied even in cases where all read-heads move synchronously.

In Zigula, j=2 and k=3 because the main tape that allows the machine to move to a given state is the timing tier where the machine must maximally scan a window of size 3 (i.e k-1=2 symbols before the current input symbol). In Figure 5, the machine goes to a given state after only scanning the current input symbol in all of the states except for state 4, which requires reading a window size of 2 on the timing tape. As such, to be in state 4 means the machine has seen HL (size k-1=2) on the timing tier. Note that the transitions to state 3 and state 4 have no output (i.e have λ as outputs), this is because the machine is looking ahead on the timing tier to decide when to output the shifted H or if there are other Ls, to output them before finally outputting the shifted H. Another way of putting the window size is by saying that it is the largest sequence of input symbols the machine has to scan before finally outputting some symbol. In Figure 5, no state requires scanning more than j-1=1 and k-1=2 on the two tapes, so the machine in that figure also complies with the property in (18d). Finally, the machine also has the property in (18e) because there is no symbol on, say, the

melody tape that is absent from the timing tape. The only way this property can be violated is if the melody tape symbols are not derived from the timing tape. Because all the properties in (18) hold in the machine in Figure 5 as summarized in Table (16), it falls out that the Zigula DM-FST in Figure 5 is IML.

(24) IML Properties Evaluation Table

Properties	Machine Status
18a	✓
18b	✓
18c	✓
18d	✓
18e	✓

Table 1: The table shows that the DM-FST in Figure 5 is IML because all the properties required of an IML DM-FST hold; thus Zigula's Unbounded Tone Shift is IML.

IML is for ML what Input Strictly Local (ISL) functions (Chandlee, 2014) are for SL grammars in that IML functions enforce input locality and map inputs to outputs. As we show below, what sets IML functions apart is their ability to compute long distance processes, which neither ISL nor (1-tape)-subsequential functions can compute[18]. Output locality is enforce by another class of functions known as the Output Strictly Local (OSL) (Chandlee et al., 2015b). Finally, another class worth mentioning is the Autosegmental Input Strictly Local (A-ISL) class of function (Chandlee and Jardine, 2019a), which enforce locality over a tier, though with some nuances. See (Chandlee and Jardine, 2019a) for more details. Table 2 below summarizes the similarities and differences between families of phonological processes ISL and OSL functions characterize and those predicted to be characterizable with IML functions.

Class	Local Processes	Long-distance Processes	Unbounded Circumambient Processes
ISL	✓	✗	✗
OSL	✓	✗	✗
A-ISL	✓(+/-)	✓	✗
SUBSEQ.	✓	✓	✗
IML	✓	✓	✓

Table 2: Comparative table of the empirical coverage of ISL/OSL functions and the intended predicted coverage of IML functions. (+/-) indicates that there are some local processes that are not A-ISL.

In §4 below, we provide IML accounts of a representative number of tone patterns.

4 Empirical Survey of IML Functions

This section presents a series of analyses of common tonal processes organized in four groups: the local and subsequential processes, the non-local processes, the unbounded circumambient processes, and finally processes that are only IML with additional assumptions. Together, these processes are representative of the wide range of processes found in tone phonology, with regard to their computational expressivity. Thus, to the research question 'what kind of processes are phonological processes?', we hypothesize that phonological functions are ones that are IML. The survey below supports this hypothesis in that a majority of the processes are IML and even those that are not straightforwardly IML become IML with some well-motivated additional assumptions. This result suggests that IML is a good fit to the computational complexity typology of tone processes and more broadly of phonological processes, because tone processes are generally more complex than non-tonal ones.

[18]It is important to mention that ISL functions are a proper subset of subsequential functions(Chandlee and Heinz, 2018).

4.1 Local Processes

Local processes are those in which output strings are determined based solely on contiguous substrings of bounded length either in the input (ISL) or in the output (OSL) (Chandlee, 2014; Chandlee et al., 2014, 2015b). Although the present study is primarily concerned with long distance tone processes, the IML apparatus and its computing DM-FSTs are well suited to characterize locally bounded processes as well. We illustrate this by presenting a series of representative local processes. More specifically, this section presents analyses of the bounded tone shift in Rimi, the bounded tone spread in Northern Bemba and the unbounded tone spread in Ndebele.

4.1.1 Bounded Tone Shift in Rimi

As mentioned above, in Rimi, H tone shifts one-step to the right (Meyers, 1997; Jardine, 2016b). Example (7) is repeated in (25) below, illustrating the process. The Rimi function examples introduced in (17) are also repeated in (26) for convenience.

(25) Tone Shift in Rimi (Meyers, 1997; Jardine, 2016b)
  a. /rá-mu-ntu/ [ra̲-mú-ntu] 'of a person'
  b. /u-púm-a/ [u-pu̲m-á] 'to go away'

(26) Tone Shift in Rimi
  a. $f(\langle LHL, LLHLLL\rangle)$ = LLLHLL
  b. $f(\langle LH, LLLLLH\rangle)$ = LLLLLH

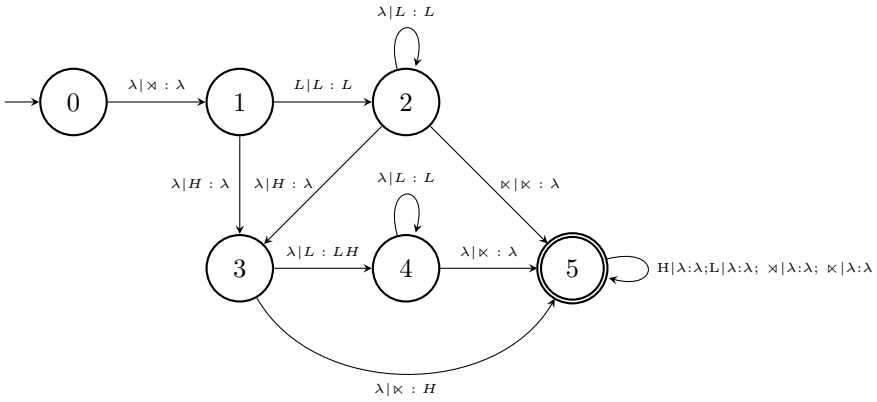


Figure 6: A 2-tape DM-FST for the one-step (bounded) tone shift in Rimi.

The Rimi function is computed by the DM-FST in Figure 6 and a sample derivation is given in Table 3. The input HLL is surrounded by domain delimiters as ⋊HLL⋉. In a first step, ⋊HLL⋉ will be fed to the melody FST, which will return ⋊HL⋉. The derivation in table 3 shows how the 2-tape DM-FST in Figure 6 takes ⋊HLL⋉ and its melody ⋊HL⋉, then outputs ⋊LHL⋉.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	⋊HL⋉	⋊̲HLL⋉	$\lambda\|\rtimes : \lambda$	*q1*	
2.	*q1*	⋊HL⋉	⋊H̲LL⋉	$\lambda\|H : \lambda$	*q3*	
3.	*q3*	⋊HL⋉	⋊HL̲L⋉	$\lambda\|L : LH$	*q4*	LH
4.	*q4*	⋊HL⋉	⋊HLL̲⋉	$\lambda\|L : L$	*q4*	LHL
5.	*q4*	⋊HL⋉	⋊HLL⋉̲	$\lambda\|\ltimes : \lambda$	*q5*	LHL
7.	*q5*	⋊̲HL⋉	⋊HLL⋉̲	$\rtimes\|\lambda : \lambda$	*q5*	LHL
8.	*q5*	⋊H̲L⋉	⋊HLL⋉̲	$H\|\lambda : \lambda$	*q5*	LHL
9.	*q5*	⋊HL̲⋉	⋊HLL⋉̲	$L\|\lambda : \lambda$	*q5*	LHL
10.	*q5*	⋊HL⋉̲	⋊HLL⋉̲	$\ltimes\|\lambda : \lambda$	*q5*	LHL

Table 3: A derivation of the Bounded Tone Shift in Rimi inputs $w = \rtimes$HLL$\ltimes$ and $\texttt{mel}(w) = \rtimes$HL$\ltimes$

The machine starts out by reading the left boundary symbols on the timing tape, and without reading anything on the melody tape, it outputs λ and transitions to state 1. It then reads an H on the timing tape, outputting yet another λ and transitioning to state 3, where it reads an L on the timing tape while still not reading anything on the melody tape. At this point, the machine has seen an H followed by an L on the timing tape, so it outputs LH, which is the crucial part of the derivation, i.e the part where the shift happens. Then, it reads an L on the timing tape and outputs an L. The machine takes another transition to state 4, where it reads the right delimiter on the timing tier, outputs a λ and transitions to state 5. Note that at this point the machine has consumed all the symbols on the timing tape while it has not yet consumed any symbol on the melody tape. As consequence, once in the accepting state 5, it only moves on the melody tape, consuming the left boundary symbol $\rtimes$, then H, then L and finally the right boundary symbol, while looping in that state and outputting λs. After that, the machine halts.

The DM-FST for Rimi is deliberately built to start out focusing on the timing tape because the process in question is ISL, i.e a strictly local process, which does not need reference to the melody. In other words, the machine only vacuously makes reference to the melody and does so only in the last few steps of the derivation. Note that the Rimi DM-FST is IML because it satisfies all the properties in (18), including but not limited to the fact that the machine does not refer to the output symbols and that each state only keeps track of the j-1 and k-1 in the input, where j and k are windows of size 2 and 3, respectively. All these properties make the Rimi function IML.

4.1.2 Bounded Tone Spread in Northern Bemba

Northern Bemba has two central tone processes: a binary H tonespread (also known as tone doubling) and an unbounded H tone spread[19]. We will only focus on the former here; i.e on the process of bounded tone spread, whereby a H tone spreads to a bounded or fixed number of TBUs. More specifically, non-phrase-final H tones spread to the immediately following TBU and do not spread any further as shown in 27a-b (Bickmore and Kula, 2014). This binary spread is OCP-constrained, in the sense that it is blocked when there is another H tone two TBUs away as shown in 27c-d; in which case spreading would violate OCP.

Because toneless TBUs in Northern Bemba surface with a L tone, we used the L symbol to represent toneless TBUs in the DM-FSTs. The examples are shown in (27) and N. Bemba function examples are shown in (28). Since we are focusing only on the binary H tone spread process, the boundary symbols in the DM-FST in Figure 7 delimit the word domain and the Northern Bemba function is a partial function because it is only defined for input strings representing words, not phrases.

(27) Bounded Tone Spread in Northern Bemba (adapted form Bickmore and Kula 2014)

a.	/bá-ka-fik-a/ (kumumana)	→	[bá-ká-fik-à]	'they will arrive tomorrow'
b.	/bá-ka-bil-a/ (mailo)	→	[bá-ká-bìl-à]	'they will sew tomorrow'
c.	/bá-ka-pít-a/ (mumusebo)	→	[bá-kà-pít-á]	'they will pass in the road'
d.	/bá-ka-cáp-a/ (mailo)	→	[bá-kà-cáp-á]	'they will wash tomorrow'

(28) The Northern Bemba Function Examples

a.	$f(\langle HL, HLLL\rangle)$	= HHLL
b.	$f(\langle L, LLLL\rangle)$	= LLLL
c.	$f(\langle HLH, HLH\rangle)$	= HLH

An IML function DM-FST that computes the bounded tone spread in Northern Bemba is shown in Figure 7.

A derivation of the example in (27a) (i.e for the input $\langle HL, HLLL\rangle$) is given in Table 4. In that derivation, the machine reads the boundary symbols on both tapes and outputs an empty string,

[19]This process happens when a H tone comes last on a phrase-final word.

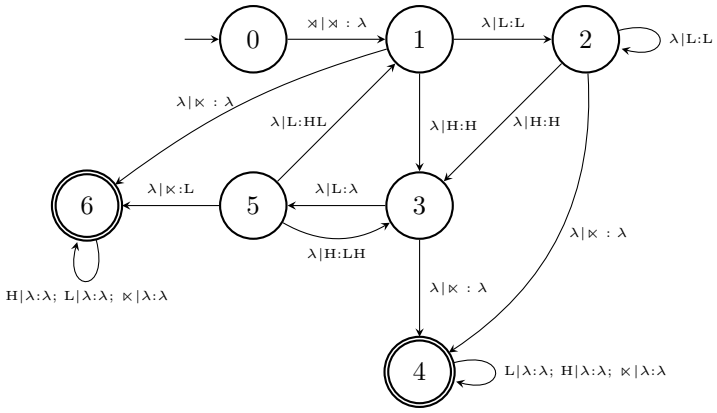


Figure 7: A 2-tape DM-FST for the one-step (bounded) tone spread in Northern Bemba.

transitioning to state 1. It then reads H on the timing tape while staying put on the melody tape, and outputs an H (i.e λ|H:H). That transition takes the machine to state 3, where the read head on the timing tape reads an L while that on the melody tape continues to stay put. The machine outputs a λ and transitions to state 5. It then reads an L on the timing tape, transitions to state 1, and outputs HL. In state 1, it reads another L on the timing tape while still staying put on the melody tape and outputting an L. That transition takes the machine to state 2, where it reads the boundary symbol on the timing tape and outputs λ, transitioning to the accepting state 4. Finally, the machine reads an H then an L and finally the right boundary symbol on the melody tape, outputting λs in each case and looping in state 4. Note that at this point, both read-heads on both tapes fall off, marking the end of the computation.

Step	Current state	Melody tape	Timing tape	Transition	Dest. tape	Output
1.	*q0*	$\underline{\rtimes}$HL⋉	$\underline{\rtimes}$HLLL⋉	⋊|⋊:λ	*q1*	
2.	*q1*	$\underline{\rtimes}$HL⋉	⋊$\underline{\text{H}}$LLL⋉	λ|H:H	*q3*	H
3.	*q3*	$\underline{\rtimes}$HL⋉	⋊H$\underline{\text{L}}$LL⋉	λ|L:λ	*q5*	H
4.	*q5*	$\underline{\rtimes}$HL⋉	⋊HL$\underline{\text{L}}$L⋉	λ|L:HL	*q1*	HHL
5.	*q1*	$\underline{\rtimes}$HL⋉	⋊HLL$\underline{\text{L}}$⋉	λ|L:L	*q2*	HHLL
6.	*q2*	$\underline{\rtimes}$HL⋉	⋊HLLL$\underline{\ltimes}$	λ|⋉:λ	*q4*	HHLL
7.	*q4*	⋊$\underline{\text{H}}$L⋉	⋊HLLL$\underline{\ltimes}$	H|λ:λ	*q4*	HHLL
8.	*q4*	⋊H$\underline{\text{L}}$⋉	⋊HLLL$\underline{\ltimes}$	L|λ:λ	*q4*	HHLL
9.	*q4*	⋊HL$\underline{\ltimes}$	⋊HLLL$\underline{\ltimes}$	⋉|λ:λ	*q4*	HHLL
10.	*q4*	⋊HL⋉	⋊HLLL⋉			HHLL

Table 4: A derivation of the Bounded Tone Spread in Northern Bemba Derivation with w = ⋊HLLL⋉ and $\texttt{mel}(w)$ = ⋊HL⋉

Intuitively, the machine outputs every low tone that immediately follows a high tone as a high and because it's a 1-step spread, every low tone afterwards is faithfully output as low. Note that the DM-FST only references the input symbols of the tapes in any given state and the machine being in a state is not due to the output of the last transition but rather to its input. This makes the Northern Bemba function an IML one.

The Northern Bemba DM-FST also captures the OCP blocking effect examplified in 28c. Table 5 below shows the derivation for it. The machine starts out by reading the left boundary symbols on both tapes in state 0, outputs λ and transitions to state 1. In this state, it reads H on the timing tape while staying put on the melody tape; outputs H and transitions to state 3. The machine then

reads L on the timing tape only, outputs λ and transitions to state 5, where it reads another H on the timing tape. Upon reading the H, the machine transitions back to state 3, outputting LH. Next, the machine reads the right boundary symbol, outputs λ and transitions to the accepting state 4, where it reads, only from the melody tape, the symbols H, L and $\ltimes$ in that order, outputting λs. The final output of that computation is the OCP-respecting string HLH.

Step	Current state	Melody tape	Timing tape	Transition	Dest. tape	Output
1.	*q0*	$\rtimes$HLH$\ltimes$	$\underline{\rtimes}$HLH$\ltimes$	$\rtimes\|\rtimes{:}\lambda$	*q1*	
2.	*q1*	$\underline{\rtimes}$HLH$\ltimes$	$\rtimes\underline{\text{H}}LH\ltimes$	λ|H:H	*q3*	H
3.	*q3*	$\underline{\rtimes}$HLH$\ltimes$	$\rtimes\text{H}\underline{\text{L}}H\ltimes$	λ|L:λ	*q5*	H
4.	*q5*	$\underline{\rtimes}$HLH$\ltimes$	$\rtimes\text{HL}\underline{\text{H}}\ltimes$	λ|H:LH	*q3*	HLH
5.	*q3*	$\underline{\rtimes}$HLH$\ltimes$	$\rtimes$HLH$\underline{\ltimes}$	$\lambda\|\ltimes{:}\lambda$	*q4*	HLH
6.	*q4*	$\rtimes\underline{\text{H}}LH\ltimes$	$\rtimes$HLH$\underline{\ltimes}$	H$\|\lambda{:}\lambda$	*q4*	HLH
7.	*q4*	$\rtimes\text{HL}\underline{\text{H}}\ltimes$	$\rtimes$HLH$\underline{\ltimes}$	L$\|\lambda{:}\lambda$	*q4*	HLH
8.	*q4*	$\rtimes\text{HL}\underline{\text{H}}\ltimes$	$\rtimes$HLH$\underline{\ltimes}$	H$\|\lambda{:}\lambda$	*q4*	HLH
9.	*q4*	$\rtimes\text{HLH}\underline{\ltimes}$	$\rtimes$HLH$\underline{\ltimes}$	$\ltimes\|\lambda{:}\lambda$	*q4*	HLH
10.	*q4*	$\rtimes$HLH$\ltimes$	$\rtimes$HLH$\ltimes$			HLH

Table 5: A derivation of the Bounded Tone Spread in Northern Bemba Derivation with $w = \rtimes\text{HLLL}\ltimes$ and $\texttt{mel}(w) = \rtimes\text{HL}\ltimes$

4.1.3 Unbounded Tone Spread in Ndebele

In Ndebele (Bantu, Zimbabwe) H tone spreads unboundedly to any number of TBUs until the antepenultimate syllable (Sibanda, 2004; Hyman, 2011) as shown in (29); (30) gives the pattern's function. Note that the unbounded spreading targets tonally unspecified TBUs, but for simplicity purposes, those tonally unspecified TBUs are represented with Ls. Furthermore, the domain of spreading is the phrase, meaning the boundary symbols mark the edges of the phrase. Just like in Zigula, the Ndebele function examplified in (30) is only defined for inputs with a single H tone, for simplicity sake.

(29) Ndebele Tone Spread (adapted from Hyman 2011)

a.	/ú-ku-hlek-a/	[ú-kú-hlek-a]	'to laugh'
b.	/ú-ku-hlek-is-a/	[ú-kú-hlé k-is-a]	'to amuse (make laugh)'
c.	/ú-ku-hlek-is-an-a/	[ú-kú-hlé k-ís-an-a]	'to amuse each other'

(30) The Ndebele Function Examples

a. $f(\langle HL, HLLLLL\rangle)$ = HHHHLL
b. $f(\langle L, LLLLLL\rangle)$ = LLLLLL

The Ndebele function in (30) is computed by the DM-FST in Figure 8. Table 6 below gives a step-by-step derivation for the example in (30b).

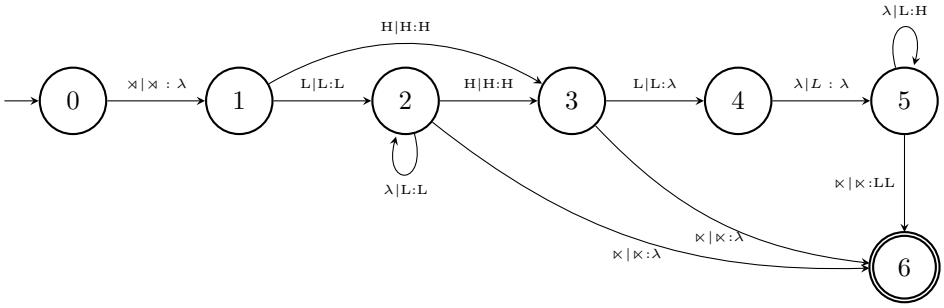


Figure 8: A 2-tape DM-FST for the unbounded tone spread to the antipenult in Ndebele.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	⋊HL⋉	⋊HLLLLL⋉	⋊|⋊:λ	*q1*	
2.	*q1*	⋊HL⋉	⋊HLLLLL⋉	H|H:H	*q3*	H
3.	*q3*	⋊HL⋉	⋊HLLLLL⋉	L|L:λ	*q4*	H
4.	*q4*	⋊HL⋉	⋊HLLLLL⋉	λ|L:λ	*q5*	H
5.	*q5*	⋊HL⋉	⋊HLLLLL⋉	λ|L:H	*q6*	HH
6.	*q6*	⋊HL⋉	⋊HLLLLL⋉	λ|L:H	*q6*	HHH
7.	*q6*	⋊HL⋉	⋊HLLLLL⋉	λ|L:H	*q6*	HHHH
8.	*q6*	⋊HL⋉	⋊HLLLLL⋉	⋊|⋉:LL	*q6*	HHHHLL
9.	*q7*	⋊HL⋉	⋊HLLLLL⋉		*q6*	HHHHLL

Table 6: The derivation shows the Unbounded tone spread in Ndebele with w = ⋊HLLLLL⋉ and $\texttt{mel}(w)$ = ⋊HL⋉

In the derivation shown in the table above, the DM-FST starts out in state 0 where is reads the left boundary symbols on both the melody and the timing tapes, outputting nothing (i.e λ) and taking a transition to state 1. In state 1, the machine reads an H on both tapes and outputs an H and transitions to state 3. At this point, the machine has seen an H, but before spreading that H, it needs to make sure the subsequent timing tape symbol is not the penult because the spreading only goes as far as the antepenult. For this reason, when the machine reads an L on both tapes in state 3, it transitions to state 4 and outputs λ as a way of waiting to see if the latest symbol it read on the timing tape is the penult. In state 4, the machine's read-head on the melody tape stays put while its timing tape read-head reads an L; the output is yet another λ, again to make sure that symbol is not the ultima.

The machine has now transitioned to state 5, meaning it has not only seen an H but also has waited enough (in two counts) to make sure the two symbols following the H on the timing tape are not the penult and the ultima, respectively. When the machine reads an L on the timing tape while staying put on the melody tape, it finally outputs an H and transitions to state 6, where it loops twice upon reading an L on the timing tape and outputting an H after each loop. Finally, the machine reads the right domain boundary symbols on both tapes, outputs two Ls and transitions to the accepting state 7. Note that the final two Ls compensate for the waiting (λ outputs) in state 4 and 5, which guarantee that the penult and final TBUs in the domain will never be spread over.

In intuitive terms, following a high tone the Ndebele DM-FST outputs every low tone as high except for the last two, which it keeps track of through its wait-states, i.e the states whose incoming transitions have the empty string as output.

4.2 Non-Local Processes

Non-local processes are presented in this subsection. By non-local, we mean tone processes involving triggers and targets that may be unboundedly far away from each other. Crucially, the non-local processes analyzed here are subsequential over string and comprise the unbounded H tone deletion in Arusa, the anticipatory Downstep in Tiriki and the non-assertive verb stems in Karanga Shona.

4.2.1 Unbounded H Tone Deletion in Arusa

In Arusa (Eastern Nilotic, Tanzania), a phrase-final H tone deletes when following another H tone that can occur any number of syllables before (Chandlee and Jardine, 2019a; Odden, 1994; Levergood, 1987). TBUs with deleted tones are supplied a default L tone on the surface. Examples (31a-b) show that the triggering H does not need to be adjacent to the phrase final target H. (31c) shows that it is always the phrase-final H tone that deletes as long as there's another H tone before it.

(31) Unbounded H Deletion in Arusa (adaped from Odden 1994)

a.	/enkér+sídáy/	[enkér sìdày]	'good ewe'
b.	/olórika+sídáy/	[olórìkà sìdày]	'good chair'
c.	/ádol+enkér+kití/	[ádòl ènkér kìtì]	'I see the small ewe'

The IML function examples for Arusa are given in (32). The example in (31a) presents an interesting case for the IML melody function in that the phrase domain has two adjacent underlying tones, the last of which will be deleted by the phrase final H tone deletion process. It's important to mention that in (31a,b), the second tone is doubly linked and when it deletes, all the TBUs it was linked to get the default L. Note that it does not matter whether the H tones are from two different underlying tones or not, the melody function considers them as a single span of Hs and will only retain a single H for the melody. In other words, the melody function takes, say, `mel`(LHHH) and outputs LH. This, however, does not prevent the Input-Output function of the IML functions from deriving the right surface form as shown in (32). In the string representations and in the DM-FSTs, the non-high tones are represented with Ls.

(32) The Arusa Function

a. $f(\langle LHLH, LHLLHH\rangle)$ = LHLLLL
b. $f(\langle LHLHLH, LHLLHLHH\rangle)$ = LHLLHLLL
c. $f(\langle LHL, LHLLLL\rangle)$ = LHLLLL

The Arusa process is IML and the IML DM-FST in (9) computes it as a right-IML function for convenience, but note that the process may also be computed with a left-IML. Table 7 summarizes how the DM-FST in Figure 9 satisfies the IML properties. The table in 8, on the other hand, shows a derivation with the inputs w = ⋊LHLLHH⋉ and `mel`(w) = ⋊LHLH⋉. Since the Arusa DM-FST used here is right-subsequential, the reading of the input strings from the input tapes and the writing of output strings on the output tape are inverted, i.e done from right to left.

(33) IML Properties Evaluation Table

Properties	Machine Status
18a	✓
18b	✓
18c	✓
18d	✓
18e	✓

Table 7: The table shows that the DM-FST in Figure 9 is IML because all the properties required of an IML DM-FST hold; thus Arusa's Unbounded H tone deletion is IML.

In state 0, the machine reads the right domain symbol on both tapes, outputs the empty string and takes a transition to state 1. It then reads an H on the melody tape, stays put on the timing tape, outputs an empty string and transitions to state 3. In this state, it moves once again on the melody tape only, where it reads an L, outputs an empty string and moves to state 4. So far the machines has probed the melody tape and read an H and an L from right to left, meaning if it reads another H on the melody, it can output the first H (from the right boundary) on the timing tier as H knowing there's another H coming up. That is exactly what happens in state 4 where it reads and H on both tapes, outputs an L and transitions to state 5. Upon reading an H on the timing tier while staying put on the melody tier in state 5, the machine outputs an L and loops.

Next, the machine's read-heads read an L on both tapes, output an L and transition to state 7. At this point, the machine has seen at least two Hs on the melody and as expected deleted (i.e turned them into Ls) all the timing tier Hs that are associated to the first melody H (from the right boundary still), so every timing tape tones read from this point on will surface faithfully. That's why when the machine read an L, then an H and another L, while staying put on the melody and looping in state 7, it output L, H and L, respectively and in that order. Finally, it read the left domain boundary symbols on both tapes, output an empty tape and transitioned to the accepting state 2. Because the IML DM-FSTs are one-way DM-FSTs, a right-IML like the one in Arusa does not only read the input symbols right-to-left but also writes the output symbols right-to-left (on the output tape).

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	q_0	⋊LHLH⋉	⋊LHLLHH⋉	⋉|⋉:λ	q_1	
2.	q_1	⋊LHLH⋉	⋊LHLLHH⋉	H|λ:λ	q_3	
3.	q_3	⋊LHLH⋉	⋊LHLLHH⋉	L|λ:λ	q_4	
4.	q_4	⋊LHLH⋉	⋊LHLLHH⋉	H|H:L	q_5	L
5.	q_5	⋊LHLH⋉	⋊LHLLHH⋉	λ|H:L	q_5	LL
6.	q_5	⋊LHLH⋉	⋊LHLLHH⋉	L|L:L	q_7	LLL
7.	q_7	⋊LHLH⋉	⋊LHLLHH⋉	λ|L:L	q_7	LLLL
8.	q_7	⋊LHLH⋉	⋊LHLLHH⋉	λ|H:H	q_7	HLLLL
9.	q_7	⋊LHLH⋉	⋊LHLLHH⋉	λ|L:L	q_7	LHLLLL
10.	q_7	⋊LHLH⋉	⋊LHLLHH⋉	⋊|⋊:λ	q_2	LHLLLL
11.	q_2	⋊HLHL⋉	⋊HHLLHL⋉			LHLLLL

Table 8: The DM-FST shows a derivations of the Unbounded L deletion in Arusa. The computed inputs are w = ⋊LHLLHH⋉ and $\texttt{mel}(w)$ = ⋊LHLH⋉.

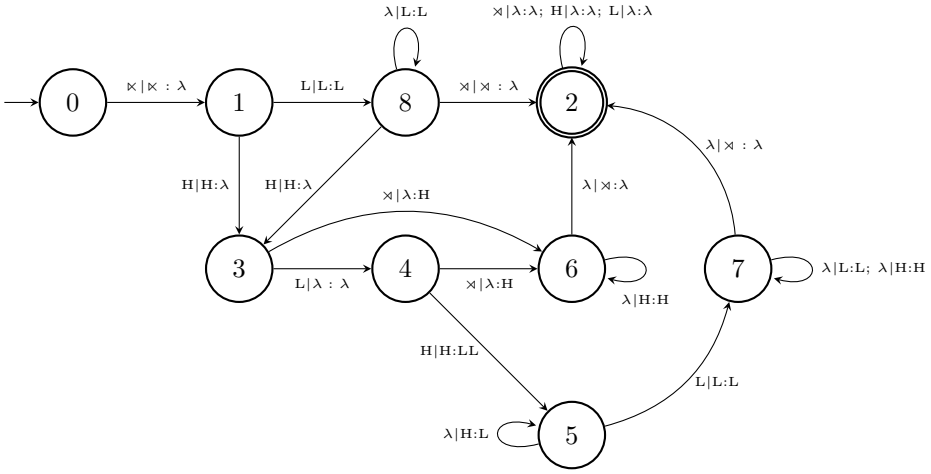


Figure 9: A 2-tape DM-FST for the unbounded phrase final H tone deletion in Arusa.

After a high tone, the Arusa DM-FST outputs the next high tone as low if it is the only other high tone. However, if there is more than one high tone after the first one, the machine outputs the last high tone in the domain as low. It is able to tell which tone is last by computing every string from the right edge of the domain. As in the previous cases, the Arusa function is also IML because the DM-FST computing it has the physical properties listed in (18); thus, being in a state is determined by the most recent input symbols from the two tapes with j-1=2 and k-1=1. The size j=3 comes from the fact that the largest possible window the DM-FST will ever have to scan to determine a given output is the melody substring **HLH**. Note that even if the melody has a longer string as in ⋊HLHL**HLH**L⋉, the DM-FST will always maximally need to consider a window of 3 symbols a time.

4.2.2 Anticipatory Downstep in Tiriki

In Tiriki (Bantu, Kenya; Kim and Paster 2007; Paster and Kim 2011; Hyman and Katamba 2010), there is a High Tone Anticipation (HTA) rule, whereby a H tone spreads unboundedly to all tonally unspecified TBUs on its left (Kim and Paster, 2007; Paster and Kim, 2011). However, when there are two underlying H tones separated by an unbounded number of toneless TBUs, the second H downsteps and with it, all the toneless TBUs it has spread to, which can be unboundedly far to the left up until the first H tone. The process occurs within a phrasal domain as shown in the example in (34). The underlined TBUs are underlyingly specified for tone.

(34) Anticipatory Downstep in Tiriki (Adapted from Hyman 2011)

a.	/xu-molom-el-a mulína/	[xú-mólóm-él-á mú-línà]	'to speak for a friend'	HHHHHHHL
b.	/xu-rhúmul-il-a mulína/	[xú-rhú$^{\downarrow}$múl-íl-á múlínà]	'to hit for a friend'	HH$^{\downarrow}$HHHHHL

In (34a), the H tone of *mulína* is anticipated on all the preceding TBUs within the phrase without being downstepped. However, in (34b) this H tone downsteps when there is another H tone before it within the domain. Because the second tone which downsteps also spreads unboundedly leftward, we say the downstep is anticipatory. The unbounded leftward spread component of the Tiriki process is OSL, which is local over the output (Chandlee, 2014; Chandlee et al., 2015a) and subsequential over strings. It is also IML-characterizable as shown below. Note however, that not all OSL functions are IML as we will see with the Alternating Meussen's Rule in §4.4. In the examples below, (35a) represents the anticipatory downstep and (35b) the regular unbounded leftward H tone spread. The Tiriki function is a partial function because it is only defined for input strings in which spans of Hs contain exactly one H. That is, the input strings may not contain contiguous Hs.

(35) The Tiriki Function Examples

a. $f(\langle LHLHL, LHLLLLHL\rangle) = \text{HH}^{\downarrow}\text{HHHHHL}$
b. $f(\langle LHL, LLLLLLHL\rangle) = \text{HHHHHHHL}$
c. $f(\langle L, LLLLLLLL\rangle) = \text{LLLLLLLL}$

The 2-tape DM-FST in Figure 10 computes the anticipatory downstep process in Tiriki. In the DM-FST, the symbol L is used *in lieu of* $\emptyset$, representing tonally unspecified TBU symbols. The particular DM-FST in Figure 10 computes the process left-to-right, although the process can also be computed with a right-subsequential DM-FST.

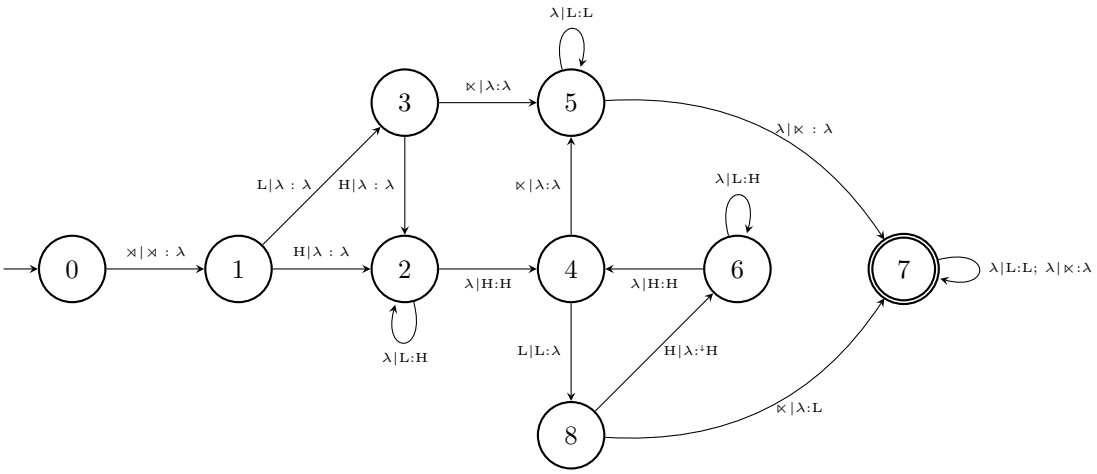


Figure 10: A 2-tape DM-FST for the Anticipatory Downstep in Tiriki.

The table below shows a step-by-step derivation of the Tiriki example in (34b) and reads as follows: the machine starts out by reading the left boundary symbols on both the timing and the melody tapes, outputting the empty string (i.e λ) and taking a transition to state 1. While in state 1, the melody read-head reads an L while the timing read-head stays put. The output is also the empty string and the machine transitions to state 3. The melody read-head then reads an H, while the timing read-head still stays put; the machine outputs the empty string and takes another transition to state 2. State 2 means the machine has seen an H on the melody, so any timing tape symbol L that the machine consumes can then be outputted as H. The melody read-head stays put while the timing read-head consumes L; the machine outputs H and loops once, meaning it stays in state 2.

Next, the machine reads H on the timing tape with its melody read-head staying put (i.e reads λ), outputting H and transitioning to state 4, where it reads L on both tapes and takes yet another transition to state 8, while outputting the empty string. In state 8, the machine probes the melody tape

by moving its melody read-head, which reads H (while the timing read-head stays put). The machine outputs $^{\downarrow}$H because this is the second H the machine has read on the melody tape, and transitions to state 6. Here the read-head on the melody tape stays put, while the read-head on the timing tape reads Ls, and the machine to output an H for each of the three Ls read on the timing tape, while it loops in state 6 until the timing read-head finally reads an H. This takes the machine back to state 4 with an H as the output. In the current state 4, the machine reads an L on both tapes, outputs an empty string, and transitions to state 8, where it finally reads the right boundary symbols on both tapes. Upon reading these boundary symbols, it outputs an L and transitions to the accepting state 7.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	q_0	$\underline{\rtimes}$LHLHL$\ltimes$	$\underline{\rtimes}$LHLLLLHL$\ltimes$	$\rtimes\|\rtimes{:}\lambda$	q_1	
2.	q_1	$\rtimes\underline{\text{L}}$HLHL$\ltimes$	$\underline{\rtimes}$LHLLLLHL$\ltimes$	L$\|\lambda{:}\lambda$	q_3	
3.	q_3	$\rtimes$L$\underline{\text{H}}$LHL$\ltimes$	$\underline{\rtimes}$LHLLLLHL$\ltimes$	H$\|\lambda{:}\lambda$	*2*	
4.	q_2	$\rtimes$L$\underline{\text{H}}$LHL$\ltimes$	$\rtimes\underline{\text{L}}$HLLLLHL$\ltimes$	λ|L:H	q_2	H
5.	q_2	$\rtimes$L$\underline{\text{H}}$LHL$\ltimes$	$\rtimes\underline{\text{L}}\underline{\text{H}}$LLLLHL$\ltimes$	λ|H:H	q_4	HH
6.	q_4	$\rtimes$LH$\underline{\text{L}}$HL$\ltimes$	$\rtimes$LH$\underline{\text{L}}$LLLLHL$\ltimes$	L|L:λ	q_8	HH
7.	q_8	$\rtimes$LH$\underline{\text{L}}$HL$\ltimes$	$\rtimes$LH$\underline{\text{L}}$LLLLHL$\ltimes$	H$\|\lambda{:}^{\downarrow}$H	q_6	HH$^{\downarrow}$H
8.	q_6	$\rtimes$LH$\underline{\text{L}}$HL$\ltimes$	$\rtimes$LHL$\underline{\text{L}}$LLLHL$\ltimes$	λ|L:H	q_6	HH$^{\downarrow}$HH
9.	q_6	$\rtimes$LH$\underline{\text{L}}$HL$\ltimes$	$\rtimes$LHLL$\underline{\text{L}}$LLHL$\ltimes$	λ|L:H	q_6	HH$^{\downarrow}$HHH
10.	q_6	$\rtimes$LH$\underline{\text{L}}$HL$\ltimes$	$\rtimes$LHLLL$\underline{\text{L}}$LHL$\ltimes$	λ|L:H	q_6	HH$^{\downarrow}$HHHH
11.	q_6	$\rtimes$LH$\underline{\text{L}}$HL$\ltimes$	$\rtimes$LHLLLL$\underline{\text{H}}$L$\ltimes$	λ|H:H	q_4	HH$^{\downarrow}$HHHHH
12.	q_4	$\rtimes$LHLH$\underline{\text{L}}\ltimes$	$\rtimes$LHLLLLH$\underline{\text{L}}\ltimes$	L|L:λ	q_8	HH$^{\downarrow}$HHHHH
13.	q_8	$\rtimes$LHLHL$\underline{\ltimes}$	$\rtimes$LHLLLLHL$\underline{\ltimes}$	$\ltimes\|\ltimes$:L	q_7	HH$^{\downarrow}$HHHHHL
14.	q_7	$\rtimes$LHLHL$\ltimes$	$\rtimes$LHLLLLHL$\ltimes$			HH$^{\downarrow}$HHHHHL

Table 9: The DM-FST shows a derivations of the Anticipatory Downstep in Tiriki. The computed inputs are ($w = \rtimes$LHLLLLHL$\ltimes$ and $\texttt{mel}(w) = \rtimes$LHLHL$\ltimes$).

In intuitive terms, the machine looks on the melody tier to see if there is a high tone, in which case all low tones encountered on the timing tier prior to seeing the high tone are output as high, capturing the unbounded leftward spreading of the high tone. The machine then probes the melody tier again by reading the next two symbols in there. If say the second of the two is a high tone, then the next low tone the machine reads on the timing tier is realized as a downstepped high tone in anticipation for the high tone coming down the road. Other low tones after that one are output as regular high until the machine finally reads the high tone on the timing tier, which is faithfully output as high. The function computed by the Tiriki DM-FST is IML because the DM-FST has all the required properties of an IML-computing DM-FST, including but not limited to the fact that each state represents the j-1 and k-1 most recent input symbols. One can see that by noticing that being in a state in Figure 10 is determined by the input of the incoming transition(s), not their output.

4.2.3 Anticipatory Upstep in Amo

Amo (Benue-Congo, Nigeria; [Anderson, 1980; Di Luzio, 1972]) has an anticipatory upstep process whereby a H tone upsteps in anticipation to a phonemic downstep that can be unboundedly far ahead within the phrasal domain (Hyman, 1979, 2011) as shown in (36). Since the source of this data did not provide underlying forms, we assume that in (36b), the input did not have the upstep.

(36) Anticipatory Upstep in Amo (Hyman, 1979, 2011)

a.	kìté úkté úkɔ́ɔ́mí fínáwà	'the place of the bed of the animal'	LHHHHHHHL
b.	kì$^{\uparrow}$té úkɔ́ɔ́mí fíká$^{\downarrow}$lé	'the place of the bed of the monkey'	L$^{\uparrow}$HHHHHH$^{\downarrow}$HL

Note that the upstep affects all the H tones that precede the downstepped tone and not just the first tone in the series. We thus assume that all the H-bearing TBUs that precede the downstepped

one are associated to the same first H tone[20]. This process is right-IML and crucially not left-IML because any machine that computes it from left-to-right may have to wait indefinitely, after seeing the first H, to output a symbol because the triggering ˡH may be unboundedly far to the right. The only alternative to a computation from the right is one that is non-deterministic from the left.

The function that characterizes the anticipatory upstep is as follows. Because the anticipatory upstep is triggered by the downstepped tone ˡH, we include it in our input alphabet (i.e we used the alphabet { L, H, **ˡH** } instead of { L, H }). As expected, the IML function output for the input with the triggering ˡH in (37a) has the upstep, which the output in (37b) lacks because the triggering ˡH is absent in its input.

(37) The Amo Function

a. $f(\langle LH^{\downarrow}H, LHHH^{\downarrow}H\rangle)$ = L$^{\uparrow}$HHH$^{\downarrow}$H

b. $f(\langle LH, LHHHH\rangle)$ = LHHHH

The DM-FST in Figure 11 computes the Anticipatory Upstep in Amo. Crucially, this DM-FST is right-subsequential because the process under consideration is not computable with a left-subsequential DM-FST, under the current assumptions. The reason being that any left-to-right computation will require an unbounded lookahead even on the melody, to ensure that there is a ˡH coming up. Since (a) the series of H tones preceding a ˡH is affected by the upstep process (as described in Hyman 1979, 2011), and (b) assuming this series of Hs can be separated by Ls, it follows that there is no limit to how many HL sequences could precede the triggering ˡH. In this case the melody tape, mirroring the timing tape, may have an unbounded number of alternating HL sequences, making the process non-subsequential going from the left.

Starting from the right, however, avoids that problem because as soon as the triggering ˡH is encountered, every H after that will be output as ˆH. Note that the analysis of this pattern as right-subsequential hinges upon assumption (b), because if upstep only affected the series of Hs immediatly before the ˡH, i.e Hs that are not broken up by Ls, then the process would be derivable with a left- as well as a right-subsequential DM-FST. Due to the limited discussion of the Amo pattern in the literature, we can not definitely conclude that Amo is not left-subsequential. In Figure 11, we assume (b) which is why the DM-FST is right-subsequential. Also, that DM-FST characterizes a partial Amo function in the sense that the function only performs upstep and does not first spread an H leftward before upstepping it; meaning the output of the spreading function is what constitutes the input to the upstep function characterized in Figure 11. A derivation is shown in Table 10.

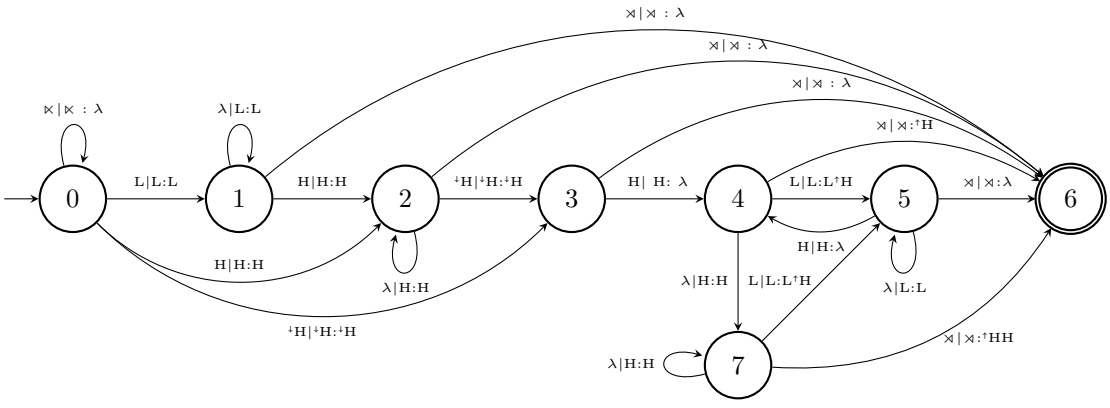


Figure 11: A 2-tape DM-FST for the Anticipatory Upstep in Amo. The FST reads the symbols from the right.

[20]We assume this because of the collective anticipatory upstep that all preceding H tones undergo, according to Hyman (1979, 2011).

(38) Anticipatory Upstep in Amo (w = ⋊LHHH$^{\downarrow}$H⋉ and $\texttt{mel}(w)$ = ⋊LH$^{\downarrow}$H⋉)

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉	⋉|×:λ	*q0*	
2.	*q0*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉	$^{\downarrow}$H|$^{\downarrow}$H:$^{\downarrow}$H	*q3*	
3.	*q3*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉	H|H:λ	*q4*	$^{\downarrow}$H
4.	*q4*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉	λ|H:H	*q7*	H$^{\downarrow}$H
5.	*q7*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉	λ|H:H	*q7*	HH$^{\downarrow}$H
6.	*q7*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉	L|L:L$^{\uparrow}$H	*q5*	L$^{\uparrow}$HHH$^{\downarrow}$H
7.	*q5*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉	⋊|⋊:λ	*q6*	L$^{\uparrow}$HHH$^{\downarrow}$H
8.	*q6*	⋊LH$^{\downarrow}$H⋉	⋊LHHH$^{\downarrow}$H⋉			L$^{\uparrow}$HHH$^{\downarrow}$H

Table 10: The DM-FST shows a Derivations of the Anticipatory upstep in Amo. The computed inputs are (w = ⋊LHHH$^{\downarrow}$H⋉ and $\texttt{mel}(w)$ = ⋊LH$^{\downarrow}$H⋉).

The derivation reads as follows: while in the initial state, the machine reads the right boundary symbols on both tapes and take a transition to state 2, outputing nothing. In state 2, it reads $^{\downarrow}$H on both tapes, outputs $^{\downarrow}$H and transitions to state 2. At this point, the machine has seen the trigger $^{\downarrow}$H. Next, it reads H on both tapes, outputs nothing and moves to state 3, where it loops twice while consuming Hs on the timing tape and staying put on the melody. Each loop outputs an H. The melody tape's read-head finally moves, reading L while the timing tape's read-head stays put; the output is $^{\uparrow}$H and the machine takes a transition to state 4. In the current state 4, the machine consumes L on the timing tier and nothing on the melody tier, outputting L. Finally, the machine reads the left boundary symbols and transitions to the accepting state 5, outputting nothing. Because the strings on both tapes were read from right-to-left, the output string has to be flipped back in order to get the observed output.

Put simply, the machine computes the string from the end and after reading a downstepped high tone, it looks on the melody to see if there's a high tone somewhere around the beginning of the string. If there's one, every low tone encountered on the timing tier is then output as high until the machine reads the spreading high tone, which is output as a upstepped high tone. If there's no high tones after the downstepped high at the end, every low tone is faithfully output as low. As in the previous cases, although each transition has a specific output, being in a state is not determined by that output, but rather by the most recent input symbols on both tapes. It falls out then that the Amo function is IML.

4.3 Unbounded Circumambient Processes

Jardine (2016b) defines an unbounded circumambient (UC) processes as one where triggers and blockers can be any number of TBUs away on both sides from the targets. These processes are famously non-subsequential over strings as discussed in §2.4, but are IML-definable. In addition to the UTP process, we also present an IML analysis of the anticipatory upstep in Amo, which is also non-subsequential over strings, although it is not exactly a UC process per Jardine (2016b)'s definition. Finally, the IML analysis of the ternary spread in Copperbelt Bemba is presented here because it is also non-subsequential over string.

4.3.1 UTP in Luganda

As explained above, UTP is a process triggered by at least Two Hs that can be arbitrarily far away from each other, where the L-toned TBUs in-between get realized as H tones (Hyman and Katamba, 2010). UTP is not just a long distance process, it is a non-myopic long distance process. 5, repeated in (39) below illustrates wit examples.

(39) Luganda (Hyman and Katamba, 2010; Jardine, 2020b)

a.	/kitabo/	[kitabo]	'book'	LLL
b.	/mutéma/	[mutéma]	'chopper'	LHL
c.	/kisikí/	[kisikí]	'log'	LLH
d.	/mutéma+bisikí/	[mutémá+bísíkí]	'log chopper'	LHHHHH
e.	Unattested	*[mutéma+bisikí]	_	*LHLLLH

(40) AR for /mutéma+bisikí/ 'log choper'

(input) $\overrightarrow{\text{(output)}}$

The UTP function in (15) is computed by the DM-FST in *Figure 12* below.

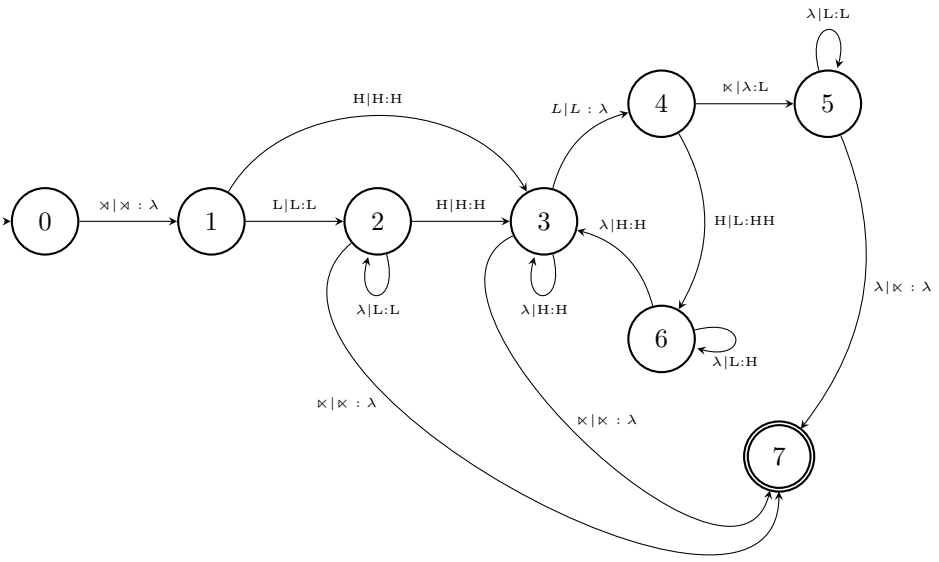


Figure 12: A 2-tape DM-FST for UTP.

The derivation for the UTP example in (40) is in (11). The drivation considered as input, the combination of the underlying strings $w = \rtimes$LHLLLH$\ltimes$ and their melody $\texttt{mel}(w) = \rtimes$LHLH$\ltimes$. Note that the input strings have two H tones, the conditioning environment for tone plateauing.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	⋊LHLH⋉	⋊LHLLLH⋉	⋊|⋊:λ	*q1*	
2.	*q1*	⋊LHLH⋉	⋊LHLLLH⋊	L|L:L	*q2*	L
3.	*q2*	⋊LHLH⋉	⋊LHLLLH⋊	H|H:H	*q3*	LH
4.	*q3*	⋊LHLH⋉	⋊LHLLLH⋊	H|λ:λ	*q4*	LH
5.	*q4*	⋊LHLH⋉	⋊LHLLLH⋊	H|L:H	*q6*	LHH
6.	*q6*	⋊LHLH⋉	⋊LHLLLH⋊	λ|L:H	*q6*	LHHH
7.	*q6*	⋊LHLH⋉	⋊LHLLLH⋊	λ|L:H	*q6*	LHHHH
8.	*q6*	⋊LHLH⋉	⋊LHLLLH⋊	λ|H:H	*q3*	LHHHHH
9.	*q3*	⋊LHLH⋉	⋊LHLLLH⋉	⋉|⋉:λ	*q7*	LHHHHH
10.	*q7*	⋊LHLH⋉	⋊LHLLLH⋊			LHHHHH

Table 11: The table shows a sample derivations with UTP. The computed input is $w =$ ⋊LHLLLH⋉ and $\texttt{mel}(w) =$ ⋊LHLH⋉.

The machine starts out reading the boundary symbols on both tapes in the initial state 0, outputing an empty string and transitioning to state 1. It then reads Ls on both tapes and outputs an L. This takes the machine to state 2 where it reads an H on both tapes and transitions to state 3. At this point, the machine looks ahead on the melody to see if an H in coming up: for this purpose, it consumes the L symbol on the melody tape but nothing (λ) on the timing tape, outputing an empty string and transitioning to state 4. While in state 4, it reads an H on the melody and an L on the timing. At this point the machine has seem a second H on the melody, so it knows that every Ls on the timing tier will need to be output as H and that is exactly what the machine outputs for each $H|L$ and $\lambda|L$ inputs, respectively transitioning to states 5 and then 6. In state 6, it loops once because it read another $\lambda|L$ from the two tapes. When the machine finally reads an H on the timing tier, it know that H represents the end of that tone plateau, so it outputs an H and transitions to state 3. Finally, the right boundary symbol is read, taking the machine to the accepting state 7 and outputting an empty string. When collected, all the single outputs form the string LHHHHH, which is the right output from the input LHLLLH.

As mentioned above and shown in example (39b), H tone plateaus only if there's another H tone somewhere in the word. Crucially, H tone does not spread when there's only one tone. The derivation in (12) shows that the FST captures the lack of spreading too.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	$\underline{\rtimes}$LHL$\ltimes$	$\underline{\rtimes}$LHL$\ltimes$	$\rtimes\|\rtimes{:}\lambda$	*q1*	
2.	*q1*	$\rtimes\underline{\text{L}}HL\ltimes$	$\rtimes\underline{\text{L}}HL\rtimes$	L|L:L	*q2*	L
3.	*q2*	$\rtimes$L$\underline{\text{H}}$L$\ltimes$	$\rtimes$L$\underline{\text{H}}$L$\rtimes$	H|H:H	*q3*	LH
4.	*q3*	$\rtimes$LH$\underline{\text{L}}\ltimes$	$\rtimes$LH$\underline{\text{L}}\rtimes$	L|L:λ	*q4*	LH
5.	*q4*	$\rtimes$LHL$\underline{\ltimes}$	$\rtimes$LH$\underline{\text{L}}\rtimes$	$\ltimes\|\lambda$:L	*q5*	LHL
6.	*q5*	$\rtimes$LHL$\underline{\ltimes}$	$\rtimes$LHL$\underline{\rtimes}$	$\lambda\|\rtimes{:}\lambda$	*q7*	LHL
7.	*q7*	$\rtimes$LHL$\ltimes$	$\rtimes$LHL$\rtimes$			LHL

Table 12: The table shows a sample derivation without UTP. The computed input is $w = \rtimes$LHL$\ltimes$ and $\texttt{mel}(w) = \rtimes$LHL$\ltimes$.

When there is only one H in the input, the machine still starts the same as when there are two H tones. It first reads the left boundary symbols on both tapes and outputs an empty string and taking a transition to state 1. It then reads Ls on both tapes and outputs L, taking another transition to state 2. It then reads an H on both tapes, outputs H and transitions to state 3. At this point, just like in (11) it takes a peek into what is coming up on the melody, so it consumes a symbol of the melody but stays put on the timing tier, while taking a transition to state 4. The machine then reads the right edge symbol on the melody and L on the timing tape, realizing there is no other H tone down the road. This prompts the machine to output $\ltimes|L$ as L and transition to state 5. Finally, it reads $\lambda|\ltimes$ on the two tapes and outputs the empty string. As expected, the output of LHL is LHL.

Intuitively, when the DM-FST for UTP simultaneously reads a high tone on both tiers, it faithfully outputs a single high tone. However, before any further readings of the symbols on the timing tier, it first checks the melody tier to see if there's another high tone coming down the road in the string by scanning a window of size j=3. If there's one, it outputs any low tones encountered on the timing tier as high; if not, all low tones are faithfully output as low. The UTP function is IML because each state in the DM-FST in Figure 12 represents the most recent j-1 and k-1 input symbols from the two tapes (where k=2).

4.3.2 Ternary Spreading in Copperbelt Bemba

In Copperbelt Bemba, H tone spreads unboundedly to the right (41a-d) unless there is another underlying H that can be arbitrarily far to the right, in which case the first H only spreads up to three TBUs at most (41e-g), obeying the OCP (Bickmore and Kula, 2013). Although the ternary spreading is local, note that it results from a long distance blocking effect of the second H, otherwise the first H would have spread all the way to the end of the domain.

(41) Ternary and Unbounded Spreading in Copperbelt Bemba (adapted from Bickmore and Kula 2013)[21]

a.	/u-ku-tul-a/	[u-ku-tul-a]	'to pierce'
b.	/bá-ka-fik-a/	[bá-ká-fíká]	'they will arrive'
c.	/bá-ka-mu-londolol-a/	[bá-ká-mú-lóndólól-á]	'they will introduce him/her'
d.	/tu-ka-pápaatik-a/	[tu-ka-pápáátík-á]	'we flatten'
e.	/bá-ka-pat-a kó/	[bá-ká-pát-a kó]	'they will hate'
f.	/bá-ka-londolol-a kó/	[bá-ká-lóndolol-a kó]	'they will introduce them'
g.	/tu-ka-béleeŋg-el-an-a kó/	[tu-ka-bélééŋg-él-an-a kó]	'they will introduce them'

IML functions examples of the process are given in (42). Note that in 42a-d where the input has at least two H tones, the function outputs have the ternary spread, unlike the output of (42)-e, which spreads unboundedly because its input has a single H. A derivation of the example 42-b is given in Table 13 below.

(42) The Copperbelt Bemba Function

a. $f(\langle HLH, HLH\rangle)$ = HLH
b. $f(\langle HLH, HLLH\rangle)$ = HHLH
c. $f(\langle HLH, HLLLH\rangle)$ = HHHLH
d. $f(\langle HLH, HLLLLLH\rangle)$ = HHHLLLH
e. $f(\langle HL, HLLLLL\rangle)$ = HHHHHH

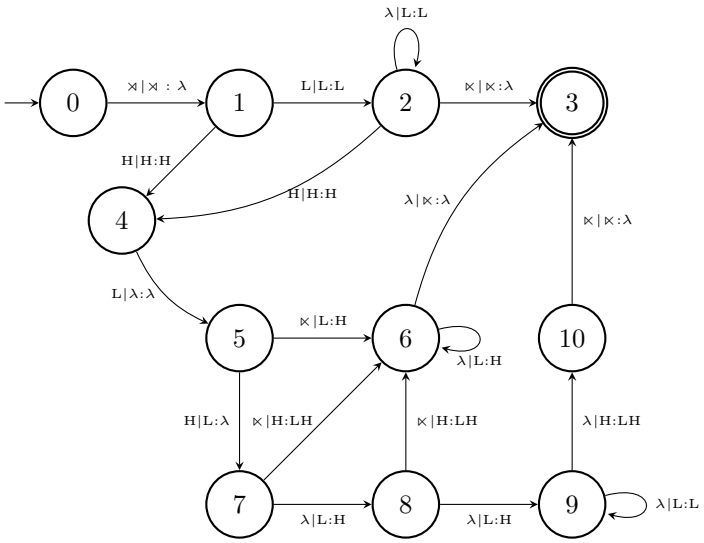


Figure 13: A 2-tape DM-FST for the Ternary Spread in Copperbelt Bemba.

The machine starts out by reading the boundary symbols on each of the two tapes, outputs λ and moves to the state 1. It then reads an H on the melody and timing tapes, outputs an H and moves to state 4. Since the machine has just seen an H on both tapes, it looks ahead on the melody tape to see if there's a another H coming up. Doing so, it reads an L on the melody tape while staying put on the timing tape, outputs a λ and advances to state 5, where it reads H on the melody and L on the timing tapes, outputting nothing. The empty output of this transition is what enforces the OCP, that requires that there be at least a low-toned TBU between the ternary spreading first H and the remote blocking second H. Now that the machine has moved to state 7, it reads an L on its timing tape while keeping its melody read-head in place, outputting an H and transitioning to state 8. Once more, the machine reads another L on the timing tape and outputs an H, taking yet another transition to state 9. The last two H outputs constitute the ternary spread because the next symbol the machine reads

[21] The surface low tones are omitted.

on the timing tape is an H, the blocker that was foreseen earlier on the melody tape. When this H is read on the timing tape, the machine outputs an LH while moving to state 10, before finally reading the end symbols on both tapes and moving to the accepting state 3.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	$\underline{\rtimes}$HLH$\ltimes$	$\underline{\rtimes}$HLLLH$\ltimes$	$\rtimes\|\rtimes{:}\lambda$	*q1*	
2.	*q1*	$\rtimes\underline{\text{H}}LH\ltimes$	$\rtimes\underline{\text{H}}$LLLH$\ltimes$	H|H:H	*q4*	H
3.	*q4*	$\rtimes$H$\underline{\text{L}}$H$\ltimes$	$\rtimes\underline{\text{H}}$LLLH$\ltimes$	L|λ:λ	*q5*	H
4.	*q5*	$\rtimes$HL$\underline{\text{H}}\ltimes$	$\rtimes$H$\underline{\text{L}}$LLH$\ltimes$	H|λ:λ	*q7*	H
5.	*q7*	$\rtimes$HL$\underline{\text{H}}\ltimes$	$\rtimes$HL$\underline{\text{L}}$LH$\ltimes$	λ|L:H	*q8*	HH
6.	*q8*	$\rtimes$HL$\underline{\text{H}}\ltimes$	$\rtimes$HLL$\underline{\text{L}}$H$\ltimes$	λ|L:H	*q9*	HHH
7.	*q9*	$\rtimes$HL$\underline{\text{H}}\ltimes$	$\rtimes$HLLL$\underline{\text{H}}\ltimes$	λ|H:LH	*q10*	HHHLH
8.	*q10*	$\rtimes$HLH$\underline{\ltimes}$	$\rtimes$HLLLH$\underline{\ltimes}$	$\ltimes\|\ltimes{:}\lambda$	*q3*	HHHLH
9.	*q3*	$\rtimes$HLH$\ltimes$	$\rtimes$HLLLH$\ltimes$			HHHLH

Table 13: The DM-FST shows a Derivations of the Ternary Spread in Copperbelt Bemba. The computed inputs are ($w = \rtimes$HLLLH$\ltimes$ and $\texttt{mel}(w) = \rtimes$HLH$\ltimes$).

Intuitively, when there are two high tones in the string, the machine spreads the first one twice at most while making sure there's at least a low tone between the two of them but if there's only one high tone in the string, the machine spreads it unboundedly to the end. The Copperbelt Bemba function is IML because the DM-FST in Figure 13 has all the properties in (18); and the transition to a given state is solely dependent upon the most recent input symbols from both tapes. What this also means is that even if the output of a given transition is changed, the transition will still go through; note however that the same can not be said of changing the input of any of the transitions. When we do this, the derivation either crashes or the machine takes a completely different transition.

4.4 Conditionally IML and Non-IML Functions

In this section, we discuss two types of processes that are not straightforwardly IML. The first type is represented by Karanga Shona non-assertive verb stem, whose tones are from a fixed set of melody just like in pure melody languages such as Mende, Hausa, etc. The challenge posed by this type of process is one of representation; that is, the range of available melody being fixed, tone and TBUs cannot be posited to have the deep connection assumed so far in the IML analyses. Thus the question: what do we posit as the input to the melody and the Input-Output functions? To circumvent this problem, we follow Hewitt and Prince (1989)'s 'edge-in' analysis of the process by assuming that the first and last tone of a given melody are respectively derived from the first and last TBUs; and that the second tone of the melody, if there are more than two is derived from the second TBU. This configuration guarantees that there will be an input suitable for the IML function to derive the right output.

The second one is represented by the Alternating Meussen's Rule (AMR) in Shona, which keeps track of even and odd numbered symbols in the string. As we show below, this function is not IML because the states of the DM-FST that compute it don't represent the most recent j-1 and k-1 input symbols but are rather depend on the output of the most recent transitions.

4.4.1 Non-Assertive Verb Stems in Karanga Shona

In Karanga Shona (Bantu, Zimbabwe; Odden 1984), non-assertive verb stems (root+extentions+final vowel) are realized with one of two possible tone patterns, the H-tone pattern and the L-tone pattern[22]. In morphosyntactic environments[23] with three or more syllables, the H-tone pattern starts with one, two or three H-tones and ends in a single H tone, with some number of L tones in the middle,

[22]This is because the patterns are determined based on whether the root has an underlying H or L/toneless.

[23]The relevant morphosyntactic environements for these patterns are: Conditional, Relative clause, Negative or reflexive (Odden, 1984)

while the L-tone pattern starts with LH or LHH and ends with some number of Ls. The former pattern is exemplified in (43) and the latter in (44). Note that the root *-p-* 'give', when extended to form the monosyllabic stem *-pa* can surface either as H-toned or L-toned, depending on the larger morphosyntactic environment the stem appears in.

To account for the two patterns and their morphosyntactically conditioned alternation in monosyllabic stems in a unified way, Odden (1984) assumed that there is a single sequence of tone, HHLB, that is mapped onto all verb stems, whether they are H-toned or L-toned. The B in HHLB is a tone variable that surfaces as H in H-toned stems and L in L-toned stems. In this analysis, we follow Odden in assuming that there is a single tone melody that is assigned to both patterns; we diverge from Odden in two regards: (1) we separate the underlying, lexical tone of the stem from the assigned melody, which in this case is HLB and (2) we assume after Hewitt and Prince (1989) that the melody tier symbols are related to the timing tier ones through the 'edge-in' association (Yip, 1988), where the first H is associated to the second TBU of the stem[24] and the last tone is associated to the last TBU of the stem. The examples are adapted from Odden (1984) (also see Jardine 2020b). In these examples, the verb stem is the form to the right of the morpheme boundary.

(43) H-tone pattern

a.	handáka-pá	'I didn't give'	H
b.	handáka-tóra	'I didn't take'	HL
c.	handáka-tóresá	'I didn't make take'	HLH
d.	handáka-tóreserá	'I didn't make take for'	HHLH
e.	handáka-tóreseraná	'I didn't make take for each other'	HHHLH
f.	handáka-tóreseresaná	'I didn't make take a lot for each other'	HHHLLH
g.	handáka-tóreseresesaná	(same as f.)	HHHLLLH

(44) L-tone pattern

h.	handá-pa	'I gave'	L
i.	handáka-biká	'I didn't cook'	LH
j.	handáka-bikísa	'I didn't make cook'	LHL
k.	handáka-bikísira	'I didn't make cook for'	LHHL
l.	handáka-bikísisira	'I didn't make cook for each other'	LHHLL
m.	handáka-bikísisirana	'I didn't make cook a lot for each other'	LHHLLL
m.	handáka-bikísirisisana	(same as l.)	LHHLLLL

Jardine (2020a) summarizes the generalization in the two patterns in the following terms: "if a verb stem ends in a LH sequence, it cannot start with a LHH sequence (...). Likewise, if a verb stem begins with a HHH sequence, it cannot end with a L" (Jardine, 2020b, p.20). Note that this generalization is long distance in the sense that the sequence of initial tones constraints what the final tone sequence (which can be any number of TBUs away) can be and vice versa. IML function examples are given in 45a-b. The DM-FST in Figure 14 computes the function and the Table 14 gives a sample derivation. As such, the assumed input string for (45a) is HLH, for (45b) is HLLH, for (45c) is HLLLLLH, for (45d) is LHL and for (45e) is LHLLL.

(45) The Karanga Shona Function

a.	$f(\langle HLH, HLH\rangle)$	= HLH
b.	$f(\langle HLH, HLLH\rangle)$	= HHLH
c.	$f(\langle HLH, HLLLLLH\rangle)$	= HHHLLLH
d.	$f(\langle LHL, LHL\rangle)$	= LHL
e.	$f(\langle LHL, LHLLL\rangle)$	= LHHLL

The derivation in Table 14 is one of the H-tone pattern, where $w = \rtimes$HLLLLH$\ltimes$ and $\texttt{mel}(w) = \rtimes$HLH$\ltimes$. It reads as follows: The machine starts out reading the left boundary symbols on both tapes in state 0 and transitions to state 1 while outputting nothing (i.e λ). In state 1, the machine

[24] The second TBU of the stem is also the first non-previously associated TBU because the underlying, lexical tone of the stem is associated to the first TBU.

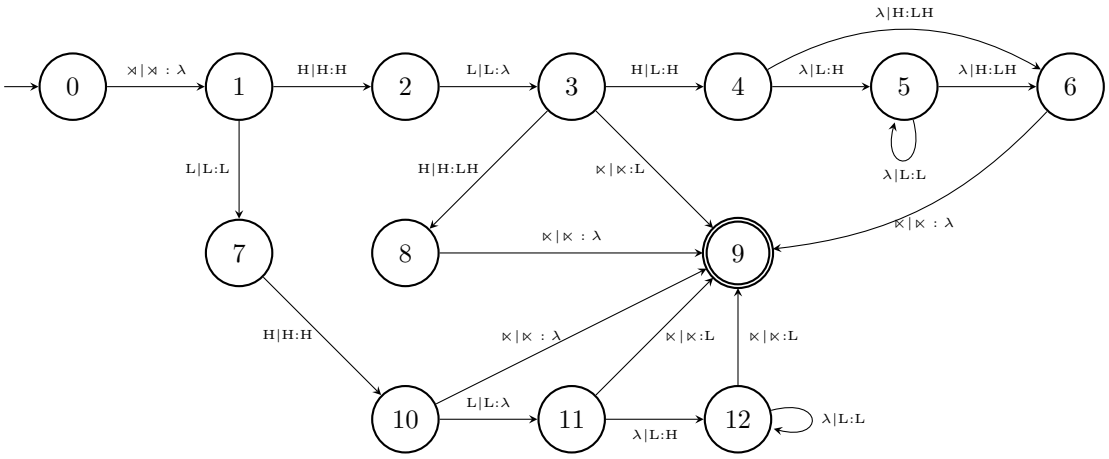


Figure 14: A 2-tape DM-FST for the non-assertive verb stem in Karanga Shona.

reads an H on each tape, outputs an H and takes a transition toward state 2, where it reads an L on the two tapes and outputs a λ while taking a transition to state 3. The λ output can be seen as a wait strategy that allows the machine to see whether there are symbols left on the timing tier, which will determine whether the next output will be an H or an L. Next, the FST reads an H on the melody tier and an L on the timing tier, outputs an H and takes a transition to state 4; then its melody read-head stays put while its timing one reads an L. The output of the machine is another H along with a transition to state 5. At this point the machine has output three consecutive Hs – the maximum number of Hs allowed in the H-tone pattern – and for this reason, when another L is read on the timing tape, the machine outputs an L and loops once. Next, it reads an H on the timing tier while the melody read-head is still staying put, outputs an LH and transitions to state 6. Finally, the machine reads the right boundary symbols on both tapes and outputs the empty string λ. Note that the derivation would have crashed if there was another L after the last H on the timing tier.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	$\underline{\rtimes}$HLH⋉	$\underline{\rtimes}$HLLLLH⋉	⋊|⋊:λ		
2.	*q1*	⋊$\underline{\text{H}}$LH⋉	⋊$\underline{\text{H}}$LLLLH⋉	H|H:H	*q2*	H
3.	*q2*	⋊H$\underline{\text{L}}$H⋉	⋊H$\underline{\text{L}}$LLLH⋉	L|L:λ	*q3*	H
4.	*q3*	⋊HL$\underline{\text{H}}$⋉	⋊HL$\underline{\text{L}}$LLH⋉	H|L:H	*q4*	HH
4.	*q4*	⋊HL$\underline{\text{H}}$⋉	⋊HLL$\underline{\text{L}}$LH⋉	λ|L:H	*q5*	HHH
5.	*q5*	⋊HL$\underline{\text{H}}$⋉	⋊HLLL$\underline{\text{L}}$H⋉	λ|L:L	*q5*	HHHL
6.	*q5*	⋊HL$\underline{\text{H}}$⋉	⋊HLLLL$\underline{\text{H}}$⋉	λ|H:LH	*q6*	HHHLLH
7.	*q6*	⋊HLH$\underline{\rtimes}$	⋊HLLLLH$\underline{\rtimes}$	⋉|⋉:λ	*q9*	HHHLLH
8.	*q9*	⋊HLH⋉	⋊HLLLLH⋉			HHHLLH

Table 14: The DM-FST shows a derivations of non-assertive H-toned verb stem in Karanga Shona. The computed inputs are (w = ⋊HLLLLH⋉ and $\texttt{mel}(w)$ = ⋊HLH⋉).

By clarifying what input is assumed, the Karanga Shona function computed in the DM-FST in Figure 14 is IML because each state represents the most recent input symbols on both tapes.

4.4.2 Alternating Meussen's Rule in Shona

There is one process that is not IML-characterizable, but could be Output Melody Local and that is the Alternating Meussen's Rule (AMR) in Shona. The rule consists of a recursive application of

Meussen's rule and results in an alternation between Hs and Ls on the surface as shown in example (46), where odd numbered Hs are output as Hs and even numbered ones as Ls. We assume the OCP by considering that consecutive tautomorphemic H tones are associated to the same H. For example, in *hóvé* 'fish', there is a single underlying H tone that is doubly associated. This process can be thought of as one in which odd numbered H-toned morphemes surface with Hs and even numbered ones with Ls, in an all-H word that is.

(46) Alternating Meussen's Rule in Shona (Odden 1986; Chandlee and Jardine 2019a)

a.	/né-hóvé/	[né-hòvè]	'with-fish'
b.	/né-é-hóvé/	[né-è-hóvé]	'with-of-fish'
c.	/sé-né-é-hóvé/	[sé-nè-é hòvè]	'like-with-of-fish'

The first issue with this process is that an HHH input, where the three Hs are bore by two different morphemes will be ambiguous between an HLL realization and an HHL one without a morpheme boundary. As noted by Chandlee and Jardine (2019a), without "morpheme boundaries the map isn't even a function" (p.22) simply because a single input string maps to more than one output and therefore trivially not IML. To fix the first problem, morpheme boundaries are included in the representation. We go about this by increasing our input alphabet with the symbol '+'. However, even the addition of the morpheme boundary symbol does not change the fact that the process outputs even numbered Hs as Ls and odd numbered Hs as Hs, a computation that requires a powerful system[25], and that constitutes the second issue with this process.

An even and odd counting process is an issue because such process can not be computed by an IML DM-FST, which the machine in Figure 15 is not. The first way to see that that machine is not IML is intuitive and consists of noting that to output a given H, the machine must know whether it is odd or even, and to get that information it needs to know how far away is the current melody symbol from the first (odd) melody symbol. As such, the size of the window to be scanned can never be fixed and will always be relative to the distance of the current symbol from the first one. In other words, each state can not represent j-1 and k-1 for any j and k because j and k can never be fixed.

One may wonder why the machine in Figure 15 still has a finite number of states and did not have the number of its states exploded as one would expect under the configuration just described. The answer to that question leads us to the second way of seeing that this particular DM-FST is not IML: the machine is in a given state depending on (at most) the 3 most recent *output* symbols, not the 3 most recent *input* symbols. If, say, the output of the transition out of state 1 was to be changed to L, this would force the output of the transition out of state 3 to be H in order to keep up with the alternating spirit of the process. This dependency on the output makes the AMR function in Shona non-IML.

(47) The AMR in Shona Function

a.	$f(\langle H+H+H+H, H+H+H+HH\rangle)$	= HLHLL
b.	$f(\langle H+H+H+H+H, H+HH+H+H+H\rangle)$	= HLLHLH

A sample derivation of the input strings $w = \rtimes$H+H+H+H$\ltimes$ and $\texttt{mel}(w) = \rtimes$H+H+H+HH$\ltimes$, from example 46c, is shown in the derivation Table 15. The table reads as follows: while in state 0, the machine reads the domain boundary symbols on both tapes and moves to state 1, outputting the empty string. Next, it reads Hs on both tapes, moves to state 2 and outputs an H. Both read-heads then read the morpheme boundary symbol +, with the machine outputting the empty string and taking a transition to state 3. This state indicates that the machine has seen an H tone followed by a morpheme boundary symbol, such that when the machine reads another H on both tapes, it outputs an L and takes another transition to state 4. In the current state, the machine reads a morpheme boundary symbol on each tape and outputs an empty string, transitioning back to state 1.

State 1 means that the machine has either just read domain boundaries on both tapes or an H (that was outputted as L) followed by a morpheme boundary on both tapes. In the case at hand it is the latter, which is why when it reads H on both tapes, it outputs an H and takes a transition to

[25] Chandlee and Jardine (2019a) noted that patterns that keep track of odd and even numbered symbols are not First-Order definable, citing McNaughton and Papert (1971).

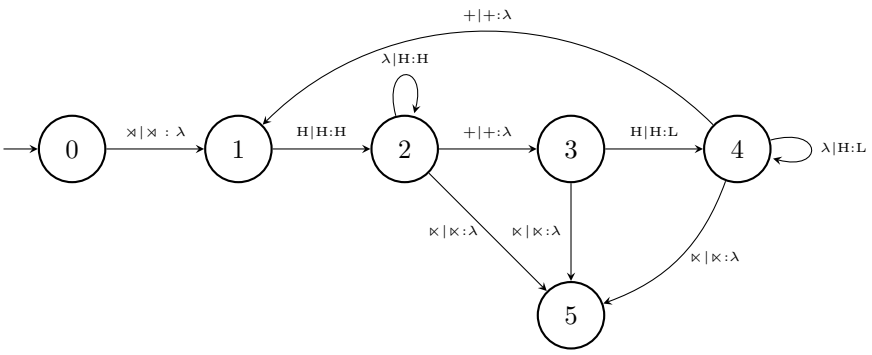


Figure 15: A 2-tape DM-FST for the Alternating Meussen's Rule in Shona. The machine does not compute an IML function.

state 2. Here it reads the morpheme boundary symbols on both tapes and outputs an empty string then moves to state 3. The read-heads of the machine now read H on both tapes with L as output. The machine which has now transitioned to state 4 stays put on the melody tape while reading an H on the timing tape with another L as output, then loops in that state. Note that until now the read-heads of the machine have moved synchronously, due to the fact that all morphemes encountered until now are monosyllabic. Finally, the machine reads the domain boundary symbols on both tapes and outputs the empty string before taking a last transition to the accepting state 5.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	⋉|⋉:λ		
2.	*q1*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	H|H:H	*q2*	H
3.	*q2*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	+|+:λ	*q3*	H
4.	*q3*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	H|H:L	*q4*	HL
5.	*q4*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	+|+:λ	*q1*	HL
6.	*q1*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	H|H:H	*q2*	HLH
7.	*q2*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	+|+:λ	*q3*	HLH
8.	*q3*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	H|H:L	*q4*	HLHL
9.	*q4*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	⋉|H:L	*q4*	HLHLL
10.	*q4*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉	⋉|⋉:λ	*5*	HLHLL
11.	*q5*	⋊H+H+H+H⋉	⋊H+H+H+HH⋉			HLHLL

Table 15: The DM-FST shows a Derivations of the Alternating Meussen's Rule in Shona. The computed inputs are w = ⋊H+H+H+H⋉ and $\texttt{mel}(w)$ = ⋊H+H+H+HH⋉.

The derivation in Table 15 is successful because the states of the DM-FST in Figure 15 represent, as discussed above, the most recent *output* symbols. We conjecture that the Shona AMR function is Output Melody Local (OML), meaning it's a function that operates over a local window on the output melody, not on the input melody. As a consequence, any DM-FST that computes it must also refer to the output as does the one in Figure 15. A thorough examination of the OML class of functions will be left for future work.

(48) IML Properties Evaluation Table

Properties	Machine Status
18a	✓
18b	✓
18c	✓
18d	✗
18e	✓

Table 16: The table shows that the DM-FST in Figure 15 is not IML because it violates one of the properties required of an IML DM-FST, namely the property in 18d; thus the Shona AMR is also not IML.

4.5 Empirical Summary

Table 17 below presents a summary of the IML tone patterns investigated above and a comparison with the other known classes of functions. The table shows that IML functions do not only capture the tone processes derivable by the previously known classes of function, but also unbounded circumambient processes in tone. It is important to note that the list of tone processes in the table is not exhaustive but is representative of the processes commonly found in tone. In anticipation to the discussion in §5, the table also includes two non-phonologically attested patterns that are crucially non-IML.

Tone Patterns and their Subregular Classes						
Patterns	Languages	ISL	OSL	A-ISL	Subseq	IML
Bounded shift	Rimi	✓	✗	✓	✓	✓
Bounded Spread	Bemba	✓	✓	✓	✓	✓
Bounded Meussen's Rule	Luganda	✓	✓	✗	✓	✓
Unbounded Shift	Zigula	✗	✗	✓	✓	✓
Unbounded Spread	Ndebele	✗	✓	✗	✓	✓
Unbounded Deletion	Arusa	✗	✗	✓	✓	✓
Anticipatory downstep	Tiriki	✗	✓	✓	✓	✓
Anticipatory Upstep	Amo	✗	✗	✓(?)	✗	✓
UTP	Luganda	✗	✗	✗	✗	✓
SG-like	C. Bemba	✗	✗	✗	✗	✓
Edge-In Assoc.	K. Shona	-	-	-	✓	✓*
AMR	Shona	✗	✓	✗	✓	✗
*Majority Rule	-	-	-	-	-	✗
*Midpoint Pathology	-	-	-	-	-	✗

Table 17: Comparative summary table of IML functions and the subregular classes of functions along with the different tone processes they characterize. The asterisk in the IML column means that the pattern in question is IML only with additional assumptions.

5 Discussion and Future Research

In this section, we discuss how IML functions and the Autosegmental Theory compare on one hand and how IML functions distinguish between attested long distance processes and the logically possible yet unattested processes.

IML functions are inspired by the Autosegmental Theory (Williams, 1976; Goldsmith, 1976) and share many properties with the latter. Firstly, IML functions employ the idea of tiers, the timing and the melody tiers, and just like in the Autosegmental Theory, these tiers interact through association lines. Association lines establish a relation between elements of the melody and timing tiers. However, IML functions differ from the Autosegmental Theory in that the melody and timing tiers don't start out independently since the former is derived from the latter through the melody function. As a consequence, the association lines are not some external mechanism that we need to superimpose on the tiers to make their interactions possible, as is the case in the Autosegmental Theory. By its very

nature, the melody function keeps track of which elements of the timing tier is associated to which element of the melody tier in the input.

Secondly, the Well-formedness conditions proposed in Autosegmental Theory (Goldsmith, 1976, p.48) are preserved in IML functions: the *no-gapping* constraint obtains from the melody function and the *no-crossing constraint* is emergent, as a bi-product of the subsequentiality of the second component of the IML functions, i.e the Input-Output function.[26] Because subsequentiality requires that symbols be read from one end going in one direction toward the opposite end, only two movement instructions are available to the read-head on each of the tapes of the corresponding DM-FST: *no movement/stay put* and *move to the subsequent symbol*. With these two movement instructions, a read-head can not go back in the opposite direction to read the symbol that precedes an already read symbol. As a result, there can never be cases (say, in a left subsequential function) where a symbol following the current symbol, say on the timing tier, is read at the same time as the symbol that precedes the last read symbol on the melody tier. If we assume that two symbols read at the same time on different tiers are associated on the surface, then the configuration just described precisely violates the no-crossing constraint.

In Autosegmental Theory, that scenario is as in (49), where the L tone preceding the H tone is associated with the TBU that follows the TBU that H is associated to.

(49) Violation of No-crossing Constraint in AR
LHL

Table 18 shows the IML derivation of the *no-crossing* constraint-violating structure in (49). At step 4, the derivation crashes because the read-head on the melody tier moves back to an already read symbol, violating the property of subsequentiality of the function, which requires that the read-heads only move in one direction. The crashing of the derivation indicates that the pattern it represents is not IML. It then falls out that any IML-compatible structure will never violate the no-crossing constraint, at the very least.

Step	Current state	Melody tape	Timing tape	Transition	Dest. state	Output
1.	*q0*	⋊LHL⋉	⋊LHL⋉	⋊|⋊:λ	*q1*	
2.	*q1*	⋊LHL⋉	⋊LHL⋉	L|L:L	*q2*	L
3.	*q2*	⋊LHL⋉	⋊LHL⋉	H|H:H	*q2*	LH
***4.**	*q2*	⋊LHL⋉	⋊LHL⋉	–	–	–

Table 18: No-crossing Constraint-Violating Derivation with IML functions.

Furthermore, an interesting empirical result from the application of IML functions helps us make a testable hypothesis about the controversy surrounding the notion of directionality of association in Autosegmental Theory. IML functions are mainly developed for long-distance and unbounded circumambient processes, which generally require pre-association. They are not so much compatible with languages like Mende (Leben, 1978) or Kukuya (Hyman, 1987), which have a fixed inventory of melody and thus polarising the debate around the question of directionality in association. Still, IML gives us a nice empirical result regarding directionality in long distance processes. That is, for each process characterizable with a right-IML (R-IML) function, there is a left-IML (L-IML) function that also characterizes the exact same process. This behavior of IML functions supports the empirical hypothesis that most (though not all) phonological processes, if at all IML-characterizable, fall at the intersection of R-IML and L-IML.

A prediction of this hypothesis is that a long distance tone process will be indifferent to directionality, meaning it will be both L-IML and R-IML. Should this prediction be borne out, it will constitute an additional support for the idea that directionality of association needs not be encoded in the grammar (Zoll, 2003), at least not for long distance processes. This is not to say that directionality, as a formal property of subsequential functions is to be overlooked, but rather that phonological

[26] Eisner (1997) got around the need to stipulate these constraints by simply eliminating explicit association lines.

processes involving tone interaction at a long distance should not care whether they are computed in one direction or in the other. Amo seems to constitute a challenge to this hypothesis in being a long distance process that is only characterizable with a right-IML function. Note however, that Amo has an element of locality to it because the anticipatory upstep is built on an unbounded rightward spread, which is OSL thus local on the output. Maybe the observed limitation in the directionality of its computation is due to the interaction between the local and long distance parts of the process. That's all we will say about that process at this point. If anything, it is Karanga Shona that is a challenge (at least partially) to this hypothesis since it requires making a directionality assumption of 'Edge-in' for the melody function to work. The Input-Output function then applies in either direction. Also note that Karanga Shona is not only a long distance process, it is also a 'pure' melody language like the above-mentioned Mende languages, which we argued, IML does not do well on.

Turning to the non-IML functions, we have seen that the AMR in Shona is not IML because the states of the machine that computes it represent the most recent output instead of only representing the most recent inputs on the two tapes. We conjecture that the Shona AMR function is OML, a class that will be for IML, what OSL is for ISL, i.e one that focuses on the output instead.

Given IML functions' ability to characterize complex phonological processes in tone, it is only logical to ask the question as to whether they also predict the existence of patterns we don't otherwise see in phonology. To answer that question, we will examine two patterns that are established in the literature to be ones that are logically possible yet unattested in phonology. The first one is the Majority Rule, a pattern where the feature with the largest number of iterations drives a given process, and the second one is the 'Midpoint Pathology' that arises from gradient evaluations of Generalized Alignment constraints (McCarthy and Prince, 1993b) in OT, where a single floating H tone can be pressured into docking on the center-most TBU in the word (Eisner 1997; also see Buckley 2009 for a similar pathological consequence in stress systems).

Formally, a majority rule is a function of the type:

(50) Majority Rule

$$f(\mathrm{a}^m\mathrm{b}^n) = \begin{cases} a^{m+n} & \text{if} \quad m > n \\ b^{m+n} & \text{if} \quad m < n \end{cases}$$

This function is not IML for the simple reason that any machine that computes it will have to keep track of the number of *a*s and *b*s, i.e the machine will need to realize when it has passed the middle $(m+n/2)$ of the input string. Such realization requires as much states as there are symbols in the input string (i.e $m+n$ states) because in automata theory, each state encodes a piece of the memory of the machine. Given that m and n are variables, the number of states in the machine can not be finite and will change as a function of the sum of m and n, thus the function is not even FST-computable. Note that the melody tier will be powerless here because it does not keep track of the number of symbols in the spans of symbols (from the timing tier) it resulted from. Likewise, the Midpoint Pathology is also not predicted for the exact same reason, i.e the need for the machine to realize when it reaches the middle of the string.

The exclusion of the Majority Rule and the Midpoint Pathology from the predicted typology of IML functions is an indication that they cut out a class that is just right for phonology. The expressivity of the IML class of functions resulting in their ability to derive a variety of phonological processes in tone speaks for its descriptive adequacy (Chomsky, 1965). Yet, this class does not seem overly powerful since it fails to generate unattested patterns, which is a desirable result. And because this class includes the most computationally complex processes in phonology, one can say with confidence that the ML class of functions delimits the upper bound of what is a possible phonological process. Put differently, we can hypothesis that any phonological process ought to be subsequential, either over a given string (and?/)or over the corresponding melody of similar alphabet.

The 'subsequentiality hypothesis' for phonology is only possible with an enriched representation – like using tiers, which highlights the fact that representation matters in phonology. That being said, the subsequentiality hypothesis can not be limited to tone, raising the question of whether the class of IML functions includes non-tonal phonological processes. IML functions are trivially adaptable to segmental processes, one only needs to replace tone symbols in the current alphabet with the symbols

relevant to the segmental process in question, say a feature. However, a challenge with using features comes from cases of vowel harmony where the harmonizing feature only spreads onto vowels with specific (other) features. While giving a definite analysis for vowel harmony is beyond the scope of this paper, a tentative solution would be to allow interacting features to have a shared melody tier. This will also help keep the number of input tiers for the IML functions to two.

Finally, there are a number of questions that will be left for future research. On a theoretical level, there is the question of the abstract characterization of IML functions. In this paper, we only laid out the building blocks of the new class using an automata characterization. However, even the Multi-tape machines used for our purpose here have not yet been properly characterized. That is, we have yet to prove that Multi-tape machines correspond to a specific class in (or subclass of) the Chomsky Hierarchy. Empirically, IML functions don't directly extend to languages with a fixed set of melody like Mende, Kukuya, Ahusa, etc. This is simply because in those languages, the melody and the string of TBUs start out independently from each other, contrary to the core assumption in IML functions that the former is derived from the latter. Rawski and Dolatian (2020)' class of MISL, because it assumes independance between tiers, is able to capture the 'melody languages', but again this is also a class so expressive it characterizes templatic morphology, thus not restrictive enough for phonology. Taking into account the Shona AMR that has been shown to be non-IML and conjectured to be OML, the findings of this paper strongly suggest that the melody local functions (i.e IML and OML) delimit the outer bounds of complexity in phonology.

6 Conclusion

In this paper, we proposed and defined a new class of functions, the IML functions, which rely on an enriched representation made of a timing tier and the derived melody tier. The new class proved descriptively adequate in characterizing a number of tone patterns that are representative of a range of processes with varying levels of complexity, including but not limited to the complex Unbounded Circumambient processes. The class of IML functions also proved restrictive in that it does not predict the existence of non-attested patterns like the Majority Rule and the Midpoint Pathology. Furthermore, the new class allows for the strong and testable phonological hypothesis that a phonological process must be subsequential over a string of TBUs and/or its derived melody. Future work will focus on the abstract characterization of IML functions, on how to extend them to 'pure' melody languages and on exploring the OML class of functions.

References

Akinlabi, A. (1996). Featural affixation. *Journal of Linguistics*, 32(2):239–289.

Bickmore, L. and Kula, N. C. (2014). Mutually-feeding iterative rules in Copperbelt Bemba Phrasal Tonology. Paper presented at the Eighth North American Phonology Conference.

Bickmore, L. S. and Kula, N. C. (2013). Ternary spreading and the ocp in copperbelt bemba. *Studies in African Linguistics*, 42(2):101–132.

Breteler, J. (2018). A foot-based typology of tonal reassociation. *LOT Dissertation Series*.

Buckley, E. (2009). Locality in metrical typology. *Phonology*, 26:389–435.

Chandlee, J. (2014). *Strictly Local Phonological Processes*. PhD thesis, University of Delaware.

Chandlee, J., Athanasopoulou, A., and Heinz, J. (2012a). Evidence for classifying metathesis patterns as subsequential. In Choi, J., Hogue, E. A., Punske, J., Tat, D., Schertz, J., and Trueman, A., editors, *Proceedings of the 29th West Coast Conference on Formal Linguistics*. Somerville, MA: Cascadilla Press.

Chandlee, J., Athanasopoulou, A., and Heinz, J. (2012b). Evidence for classifying metathesis patterns as subsequential. In *The Proceedings of the 29th West Coast Conference on Formal Linguistics*, pages 303–309.

Chandlee, J., Eyraud, R., and Heinz, J. (2014). Learning Strictly Local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

Chandlee, J., Eyraud, R., and Heinz, J. (2015a). Output strictly local functions. In Kornai, A. and Kuhlmann, M., editors, *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 14)*, pages 52–63, Chicago, IL.

Chandlee, J. and Heinz, J. (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*, pages 42–51, Montreal, Canada. Association for Computational Linguistics.

Chandlee, J. and Heinz, J. (2018). Strictly locality and phonological maps. *LI*, 49:23–60.

Chandlee, J., Heinz, J., and Jardine, A. (2018). Input Strictly Local opaque maps. *Phonology*, 35:1–35.

Chandlee, J. and Jardine, A. (2014). Learning phonological mappings by learning Strictly Local functions. In Kingston, J., Moore-Cantwell, C., Pater, J., and Staubs, R., editors, *Proceedings of the 2013 Meeting on Phonology (UMass Amherst)*, Proceedings of the Annual Meetings on Phonology. LSA.

Chandlee, J. and Jardine, A. (2019a). Autosegmental input-strictly local functions. *Transactions of the Association for Computational Linguistics*, 7:157–168.

Chandlee, J. and Jardine, A. (2019b). Quantifier-free least fixed point functions for phonology. In *Proceedings of the 16th Meeting on the Mathematics of Language*, pages 50–62.

Chandlee, J., Jardine, A., and Heinz, J. (2015b). Learning repairs for marked structures. In *Proceedings of the 2015 Annual Meeting on Phonology*. LSA.

Chomsky, N. (1965). Aspects of the theory of syntax (vol. 11). *MIT Press. doi*, 10:90008–5.

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

Copeland, B. J. (1996). What is computation? *Synthese*, 108(3):335–359.

Dolatian, H. and Rawski, J. (2020). Multi-input strictly local functions for templatic morphology. *Proceedings of the Society for Computation in Linguistics (SCiL)*, pages 282–296.

Eisner, J. (1997). Efficient generation in primitive Optimality Theory. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 313–320, Madrid.

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

Filiot, E. and Reynier, P. (2016). Transducers, logic, and algebra for functions of finite words. *ACM SIGLOG News*, 3(3):4–19.

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

Gibbon, D. (1987). Finite state processing of tone systems. In *Proceedings of the third conference on European chapter of the Association for Computational Linguistics*, pages 291–297. Association for Computational Linguistics.

Goldsmith, J. (1976). *Autosegmental Phonology*. PhD thesis, Massachussets Institute of Technology.

Graf, T. and Heinz, J. (2015). Commonality in disparity: The computational view of syntax and phonology. *Slides of a talk given at GLOW*, 18.

Heinz, J. (2018). The computational nature of phonological generalizations. *Phonological Typology, Phonetics and Phonology*, pages 126–195.

Heinz, J. and Lai, R. (2013). Vowel harmony and subsequentiality. In Kornai, A. and Kuhlmann, M., editors, *Proceedings of the 13th Meeting on Mathematics of Language*, Sofia, Bulgaria.

Heinz, J., Rawal, C., and Tanner, H. G. (2011). Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 58–64. Association for Computational Linguistics.

Hewitt, M. and Prince, A. (1989). Ocp, locality, and linking: The n. karanga verb. In *Proceedings of the Eighth WCCFL*, pages 176–191.

Hulden, M. (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.

Hyman, L. M. (1979). A reanalysis of tonal downstep. *Journal of African Languages and Linguistics*, 1(1):9–30.

Hyman, L. M. (1987). Prosodic domains in kukuya. *Natural Language & Linguistic Theory*, 5(3):311–333.

Hyman, L. M. (2011). Tone: Is it different. *The handbook of phonological theory*, 2.

Hyman, L. M. and Katamba, F. X. (2010). Tone, syntax, and prosodic domains in luganda. *UC Berkeley PhonLab Annual Report*, 6(6).

Jardine, A. (2016a). Autosegmental representations in zigula and shambaa.

Jardine, A. (2016b). Computationally, tone is different. *Phonology*, 33(2):247–283.

Jardine, A. (2016c). *Locality and non-linear representations in tonal phonology*. PhD thesis, University of Delaware.

Jardine, A. (2020a). Melody learning and long-distance phonotactics in tone. *Natural Language & Linguistic Theory*, pages 1–51.

Jardine, A. (2020b). Melody learning and long-distance phonotactics in tone. *Natural Language and Linguistic Theory*, pages 1–51.

Kenstowicz, M. (1994). *Phonology in Generative Grammar*. Blackwell Publishing.

Kenstowicz, M. and Kisseberth, C. (1990). Chizigula tonology: the word and beyond. *The phonology-syntax connection*, pages 163–194.

Kim, Y. and Paster, M. (2007). Downstep in tiriki. *UC-Berkeley & Pomona College, Ms.*

Kornai, A. (1995). *Formal Phonology*. Garland Publication.

Koser, N., Oakden, C., and Jardine, A. (2019). Tone association and output locality in non-linear structures. In *Proceedings of the Annual Meetings on Phonology*, volume 7.

Leben, W. (1973). Suprasegmental phonology. doctoral dissertation, massachusetts institute of technology.

Leben, W. R. (1978). The representation of tone. In *Tone*, pages 177–219. Elsevier.

Levergood, B. J. (1987). *Topics in Arusa Phonology and Morphology*. PhD thesis.

Luo, H. (2017). Long-distance consonant agreement and subsequentiality. *Glossa: a journal of general linguistics*, 2(1).

McCarthy, J. and Prince, A. (1993a). Generalized alignment. In Booij, G. and van Marle, J., editors, *Yearbook of Morphology*, pages 79–153. Dordrecht: Kluwer.

McCarthy, J. and Prince, A. (1993b). Generalized alignment. Ms., University of Massachusetts, Amherst, and Rutgers University.

McCarthy, J. J. (1986). OCP effects: gemination and antigemination. *LI*, 17:207–263.

McCollum, A., Bakovic, E., Mai, A., and Meinhardt, E. (2017). Conditional blocking in tutrugbu requires non-determinism: implications for the subregular hypothesis. *Presentation at NELS*, 48.

McNaughton, R. and Papert, S. (1971). *Counter-Free Automata*. MIT Press.

Meyers, S. (1997). Ocp effects in optimality theory. *Natural Language & Linguistic Theory*, 15(4):847–892.

Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Myers, S. P. (1987). Tone and the structure of words in shona.

Odden, D. (1984). Stem tone assignment in shona. *Autosegmental studies in Bantu tone*, pages 255–280.

Odden, D. (1986). On the role of the Obligatory Contour Principle in phonological theory. *Language*, 62(2):353–383.

Odden, D. (1988). Anti antigemination and the ocp. *Linguistic inquiry*, 19(3):451–475.

Odden, D. (1994). Adjacency parameters in phonology. *Language*, pages 289–330.

Paster, M. and Kim, Y. (2011). Downstep in tiriki. *Linguistic Discovery*, 9(1).

Prince, A. and Smolensky, P. (1993). Optimality Theory: Constraint interaction in generative grammar. *Rutgers University Center for Cognitive Science Technical Report*, 2.

Prince, A. and Smolensky, P. (2004). *Optimality Theory: Constraint Interaction in Generative Grammar*. Blackwell Publishing.

Pulleyblank, D. (1986). *Tone in lexical phonology*, volume 4. Springer Science & Business Media.

Rawski, J. and Dolatian, H. (2020). Multi-input strict local functions for tonal phonology. *Proceedings of the Society for Computation in Linguistics*, 3(1):245–260.

Schützenberger, M. P. (1977). Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4:47–57.

Sibanda, G. (2004). Verbal phonology and morphology of ndebele. berkeley, ca: University of california dissertation.

Williams, E. S. (1976). Underlying tone in margi and igbo. *Linguistic Inquiry*, pages 463–484.

Wilson, C. (2003). Analyzing unbounded spreading with constraints: marks, targets, and derivations. Unpublished ms.

Wilson, C. (2006). Unbounded spreading is myopic. Paper presented at the workshop Current Perspectives on Phonology.

Yip, M. (1988). Template morphology and the direction of association. *NLLT*, 6:551–577.

Zoll, C. (2003). Optimal tone mapping. *Linguistic Inquiry*, 34(2):225–268.