Recognizing Reduplicated Forms: Finite-State Buffered Machines

Yang Wang
Department of Linguistics
University of California, Los Angeles
Los Angeles, CA, USA
yangwangx@g.ucla.edu

Abstract

Total reduplication is common in natural language phonology and morphology. However, formally as copying on reduplicants of unbounded size, unrestricted total reduplication requires computational power beyond context-free, while other phonological and morphological patterns are regular, or even sub-regular. Thus, existing language classes characterizing reduplicated strings inevitably include typologically unattested context-free patterns, such as reversals. This paper extends regular languages to incorporate reduplication by introducing a new computational device: finite state buffered machine (FSBMs). We give its mathematical definitions and discuss some closure properties of the corresponding set of languages. As a result, the class of regular languages and languages derived from them through a copying mechanism is characterized. Suggested by previous literature (Gazdar and Pullum, 1985), this class of languages should approach the characterization of natural language word sets.

1 The Puzzle of (Total) Reduplication

Formal language theory (FLT) provides computational mechanisms characterizing different classes of abstract languages based on their inherent structures. Following FLT in the study of human languages, in principle, researchers would expect a hierarchy of grammar formalisms that matches empirical findings: more complex languages in such a hierarchy are supposed to be 1) less common in natural language typology; and 2) harder for learners to learn.

The classical Chomsky Hierarchy (CH) puts formal languages into four levels with increasing complexity: regular, context-free, context-sensitive, recursively enumerable (Chomsky, 1956; Jäger and Rogers, 2012). Does the CH notion of formal complexity have the desired empirical correlates? Several findings suggest that those four levels do not align with natural languages precisely, some leading to major refinements on the CH. First, the unbounded crossing dependencies in Swiss-German case marking (Shieber, 1985) facilitated attempts to characterize mildly context-sensitive languages (MCS), which extend context-free languages (CFLs) but still preserve some useful properties of CFLs (e.g., Joshi, 1985; Seki et al., 1991; Stabler, 1997). Secondly, it is generally accepted that phonology is regular (e.g. Johnson, 1972; Kaplan and Kay, 1994). However, being regular is argued to be an unrestrictive property for phonological well-formed strings: for example, a language whose words are sensitive to an even or odd number of certain sounds is unattested (Heinz, 2018). With strong typological evidence, the sub-regular hierarchy was further developed, which continues to be an active area of research (e.g., McNaughton and Papert, 1971; Simon, 1975; Heinz, 2007; Heinz et al., 2011; Chandlee, 2014; Graf, 2017).

In this paper, we analyze another mismatch between existing well-known language classes and empirical findings: reduplication, which involves copying operations on certain base forms (Inkelas and Zoll, 2005). The reduplicated phonological strings are either of total identity (*total reduplication*) or of partial identity (*partial reduplication*) to the base forms. Table 1 provides examples showing the difference between total reduplication and partial reduplication: in Dyirbal, the pluralization of nominals is realized by fully copying the singular stems, while in Agta examples, plural forms only copy the first CVC sequence of the corresponding singular forms (Healey, 1960; Marantz, 1982).

Reduplication is common cross-linguistically. According to Rubino (2013) and Dolatian and Heinz (2020), 313 out of 368 natural languages exhibit productive reduplication, in which 35 languages only have total reduplication, but not partial

Proceedings of the Seventeenth SIGMORPHON Workshop on Computational Research

in Phonetics, Phonology, and Morphology,pages 177–187

Total reduplication: Dyirbal plurals (Dixon, 1972, 242)			
Singular	*Gloss*	*Plural*	*Gloss*
midi	'little, small'	midi-midi	'lots of little ones'
gulgiɽi	'prettily painted men'	gulgiɽi-gulgiɽi	'lots of prettily painted men'

Partial reduplication: Agta plurals (Healey, 1960,7)			
Singular	*Gloss*	*Plural*	*Gloss*
labáng	'patch'	lab-labáng	'patches'
takki	'leg'	tak-takki	'legs'

Table 1: Total reduplication: Dyirbal plurals (top); partial reduplication: Agta plurals (bottom).

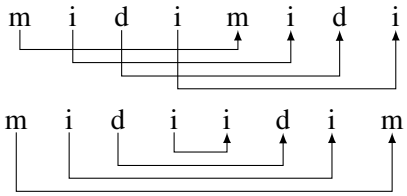

Figure 1: Crossing dependencies in Dyirbal total reduplication *'midi-midi'* (top) versus nesting dependencies in unattested string reversal *'midi-idim'* (bottom)

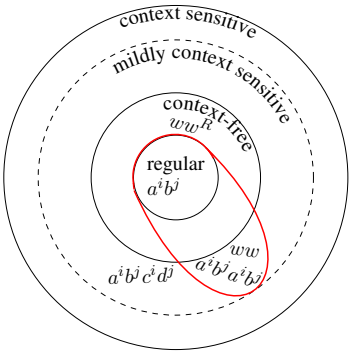

Figure 2: The class of regular with copying languages in CH

reduplication. As a comparison, it is widely recognized that context-free string reversals are rare in phonology and morphology (Marantz, 1982) and appear to be confined to language games (Bagemihl, 1989).

Unrestricted total reduplication, or unbounded copying, can be abstracted as $L_{ww} = \{ww \mid w \in \Sigma^*\}$, a well-known non-context free language (Culy, 1985; Hopcroft and Ullman, 1979).[1] Its non-context-freeness comes from the incurred crossing dependencies among symbols, similar to Swiss-German case marking constructions. However, the typologically-rare string reversals ww^R demonstrate nesting dependencies, which are context-free (see Fig. 1 as an illustration).

Given most phonological and morphological patterns are regular, how can one fit in reduplicated strings without including reversals? Gazdar and Pullum (1985, 278) made the remark that

> We do not know whether there exists an independent characterization of the class of languages that includes the regular sets and languages derivable from them through reduplication, or what the time complexity of that class might be, but it currently looks as if this class might be relevant to the characterization of NL word-sets.

Motivated by Gazdar and Pullum (1985), this article aims to give a formal characterization of regular with copying languages. Specifically, it examines what minimal changes can be brought to regular languages to include stringsets with two adjacent copies, while excluding some typologically unattested context-free patterns, such as reversals, shown in Fig. 2. One possible way to probe such a language class is by adding copying to the set of operations whose closure defines regular languages. Instead, the approach we take in this paper is to add reduplication to finite state automata (FSAs), which compute regular languages.

[1] Total reduplication does not immediately guarantee *unboundedness*. When the set of bases is finite, i.e, $\{ww \mid w \in L\}$ when L is finite, total reduplication *can* be squeezed in languages described by 1 way finite state machines (Chandlee, 2017), though doing so eventually leads to state explosion (Roark and Sproat, 2007; Dolatian and Heinz, 2020). Computationally, only total reduplication with infinite number of potential reduplicants is true *unbounded copying*. With careful treatment, unbounded copying, externalizing a primitive copying operation, can be justified as a model of reduplication in natural languages. More in-depth discussion of 1): *bounded* versus *unbounded* and 2): copying as a primitive operation can be found in Clark and Yoshinaka (2014); Chandlee (2017); Dolatian and Heinz (2020).

Various attempts followed this vein:[2] one example is finite state registered machine in Cohen-Sygal and Wintner (2006) (FSRAs) with finitely many registers as its memory, limited in the way that it only models *bounded copying*. The state-of-art finite state machinery that computes *unbounded copying* elegantly and adequately is 2-way finite state transducers (2-way FSTs), capturing reduplication as a string-to-string mapping ($w \rightarrow ww$) (Dolatian and Heinz, 2018a,b, 2019, 2020). To avoid the mirror image function ($w \rightarrow ww^R$), Dolatian and Heinz (2020) further developed subclasses of 2-way FSTs which cannot output anything during right-to-left passes over the input (cf. rotating transducers: Baschenis et al., 2017).

It should be noted that the issue addressed by 2-way FSTs is a different one: reduplication is modeled as a function ($w \rightarrow ww$), while this paper focuses on a set of languages containing identical substrings (ww). The stringset question is non-trivial and well-motivated for reasons of both formal aspects and its theoretical relevance. Firstly, since the studied 2-way FSTs are not readily invertible, how to get the inverse relation $ww \rightarrow w$ remains an open question, as acknowledged in Dolatian and Heinz (2020). Although this paper does not directly address this morphological analysis problem, recognizing which strings are reduplicated and belong to L_{ww} or any other copying languages may be an important first step.[3]

As for the theoretical aspects, there are some attested forms of meaning-free reduplication in natural languages.Zuraw (2002) proposes *aggressive reduplication* in phonology: speakers are sensitive to phonological similarity between substrings within words and reduplication-like structures are attributed to those words. It is still arguable whether those meaning-free reduplicative patterns of unbounded strings are generated via a morphological function or not. Overall, it is desirable to have models that help to detect the substring identity within surface strings when those sub-strings are in the regular set.

This paper introduces a new computational device: finite state buffered machine (FSBMs). They are two-taped finite state automata, sensitive to copying activities within strings, hence able to detect identity between sub-strings. This paper is organized as follows: Section 2 provides a definition of FSBMs with examples. Then, to better understand the copying mechanism, complete-path FSBMs, which recognize exactly the same set of languages as general FSBMs, are highlighted. Section 3 examines the computational and mathematical properties of the set of languages recognized complete-path FSBMs. Section 4 concludes with discussion and directions for future research.

2 Finite State Buffered Machine

2.1 Definitions

FSBMs are two-taped automata with finite-state core control. One tape stores the input, as in normal FSAs; the other serves as an unbounded memory buffer, storing reduplicants temporarily for future identity checking. Intuitively, FSBMs is an extension to FSRAs but equipped with unbounded memory. In theory, FSBMs with a *bounded* buffer would be as expressive as an FSRA and therefore can be converted to an FSA.

The buffer interacts with the input in restricted ways: 1) the buffer is queue-like; 2) the buffer needs to work on the same alphabet as the input, unlike the stack in a pushdown automata (PDA), for example; 3) once one symbol is removed from the buffer, everything else must also be wiped off before the buffer is available for other symbol addition. These restrictions together ensure the machine does not generate string reversals or other non-reduplicative non-regular patterns.

There are three possible modes for an FSBM M when processing an input: 1) in normal (N) mode, M reads symbols and transits between states, functioning as a normal FSA; 2) in buffering (B) mode, besides consuming symbols from the input and taking transitions among states, it adds a copy of just-read symbols to the queue-like buffer, until it exits buffering (B) mode; 3) after exiting buffering (B) mode, M enters emptying (E) mode, in which M matches the stored symbols in the buffer against input symbols. When all buffered symbols have been matched, M switches back to normal (N) mode for another round of computation. Under the current augmentation, FSBMs can only capture local reduplication with two adjacent, completely identical copies. It cannot handle non-local reduplication, nor multiple reduplication.

[2] Some other examples, pursuing more linguistically sound and computationally efficient finite state techniques, are Walther (2000), Beesley and Karttunen (2000) and Hulden (2009). However, they fail to model unbounded copying. Roark and Sproat (2007), Cohen-Sygal and Wintner (2006) and Dolatian and Heinz (2020) provide more comprehensive reviews.

[3] Thanks to the reviewer for bringing this point up.

Definition 1. A **Finite-State Buffered Machine (FSBM)** is a 7-tuple $\langle \Sigma, Q, I, F, G, H, \delta \rangle$ where

- Q: a finite set of states
- I $\subseteq Q$: initial states
- F $\subseteq Q$: final states
- $G \subseteq Q$: states where the machine must enter buffering (B) mode
- $H \subseteq Q$: states visited while the machine is emptying the buffer
- $G \cap H = \emptyset$
- δ: $Q \times (\Sigma \cup \{\epsilon\}) \times Q$: the state transitions according to a specific symbol

Specifying G and H states allows an FSBM to control what portions of a string are copied. To avoid complications, G and H are defined to be disjoint. In addition, states in H identify certain special transitions. Transitions between two H states check input-memory identity and consume symbols in both the input and the buffer. By contrast, transitions with at least one state not in H can be viewed as normal FSA transitions. In all, there are effectively two types of transitions in δ.

Definition 2. A configuration of an FSBM $D = (u, q, v, t) \in \Sigma^* \times Q \times \Sigma^* \times \{\text{N}, \text{B}, \text{E}\}$, where u is the input string; v is the string in the buffer; q is the current state and t is the current mode the machine is in.

Definition 3. Given an FSBM M and $x \in (\Sigma \cup \{\epsilon\})$, $u, w, v \in \Sigma^*$, we define that a configuration D_1 yields a configuration D_2 in M ($D_1 \vdash_M D_2$) as the smallest relation such that: [4]

- For every transition (q_1, x, q_2) with at least one state of $q_1, q_2 \notin H$
  $(xu, q_1, \epsilon, \text{N}) \vdash_M (u, q_2, \epsilon, \text{N})$ with $q_1 \notin G$
  $(xu, q_1, v, \text{B}) \vdash_M (u, q_2, vx, \text{B})$ with $q_2 \notin G$
- For every transition (q_1, x, q_2) and $q_1, q_2 \in H$
  $(xu, q_1, xv, \text{E}) \vdash_M (u, q_2, v, \text{E})$
- For every $q \in G$
  $(u, q, \epsilon, \text{N}) \vdash_M (u, q, \epsilon, \text{B})$
- For every $q \in H$
  $(u, q, v, \text{B}) \vdash_M (u, q, v, \text{E})$
  $(u, q, \epsilon, \text{E}) \vdash_M (u, q, \epsilon, \text{N})$

[4] Note that a machine cannot do both symbol consumption and mode changing at the same time.

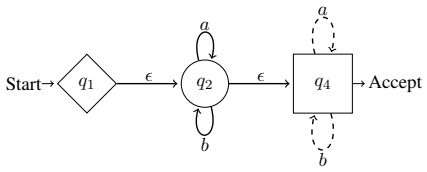

Figure 3: An FSBM M_1 with G = $\{q_1\}$ (diamond) and H = $\{q_3\}$ (square); dashed arcs are used only for the emptying process. L(M_1) = $\{ww \mid w \in \{a, b\}^*\}$

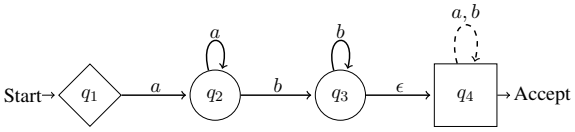

Figure 4: An FSBM M_2 with G = $\{q_1\}$ and H = $\{q_4\}$. L(M_2) = $\{a^i b^j a^i b^j \mid i, j \geq 1\}$

Definition 4. A run of FSBM M on w is a sequence of configurations $D_0, D_1, D_2 \ldots D_m$ such that 1): $\exists\, q_0 \in I$, $D_0 = (w, q_0, \epsilon, \text{N})$; 2): $\exists\, q_f \in F$, $D_m = (\epsilon, q_f, \epsilon, \text{N})$; 3): $\forall\, 0 \leq i < m$, $D_i \vdash_M D_{i+1}$. The language recognized by an FSBM M is denoted by $L(M)$. $w \in L(M)$ iff there's a run of M on w.

2.2 Examples

In all illustrations, G states are drawn with diamonds and H states are drawn with squares. The special transitions between H states are dashed.

Example 1. Total reduplication Figure 3 offers an FSBM M_1 for L_{ww}, with any arbitrary strings made out of an alphabet $\Sigma = \{a, b\}$ as candidates of bases.

L_{ww} is the simplest representation of unbounded copying, but this language is somewhat structurally dull. For the rest of the illustration, we focus on the FSBM M_2 in Figure 4. M_2 recognizes the non-context free $\{a^i b^j a^i b^j \mid i, j \geq 1\}$. This language can be viewed as total reduplication added to the regular language $\{a^i b^j \mid i, j \geq 1\}$ (recognized by the FSA M_0 in Figure 5).

State q_1 is an initial state and more importantly a G state, forcing M_2 to enter B mode before it takes any arcs and transits to other states. Then, M_2 in B mode always keeps a copy of consumed input

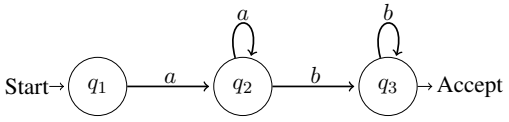

Figure 5: An FSA M_0 with $L(M_0)$= $\{a^i b^j \mid i, j \geq 1\}$

symbols until it proceeds to q_4, an H state. State q_4 requires M_2 to stop buffering and switch to E mode in order to check for string identity. Using the special transitions between H states (in this case, a and b loops on State q_4), M_2 checks whether the stored symbols in the buffer matches the remaining input. If so, after emitting out all symbols in the buffer, M_2 with a blank buffer can switch to N mode. It eventually ends at State q_4, a legal final state. Figure 6 gives a complete run of M_2 on the string "*abbabb*". Figure 7 shows M_2 rejects the non-total reduplicated string "*ababb*" since a final configuration cannot be reached.

Example 3. Partial reduplication Assume $\Sigma = \{b, t, k, ng, l, i, a\}$, the FSBM M_3 in Figure 8 serves as a model of two Agta CVC reduplicated plurals in Table 1.

Given the initial state q_1 is in G, M_3 has to enter B mode before it takes any transitions. In B mode, M_3 transits to a plain state q_2, consuming an input consonant and keeping it in the buffer. Similarly, M_3 transits to a plain state q_3 and then to q_4. When M_3 first reaches q_4, the buffer would contain a CVC sequence. q_4, an H state, urges M_3 to stop buffering and enter E mode. Using the special transitions between H states (in this case, loops on q_4), M_3 matches the CVC in the buffer with the remaining input. Then, M_3 with a blank buffer can switch to N mode at q_4. M_3 in N mode loses the access to loops on q_4, as they are available only in E mode. It transits to q_5 to process the rest of the input by the normal transitions between q_5. A successful run should end at q_5, the only final state. Figure 9 gives a complete run of M_3 on the string "*taktakki*".

2.3 Complete-path FSBMs

As shown in the definitions and the examples above, an FSBM is supposed to end in N mode to process an input. There are two possible scenarios for a run to meet this requirement: either never entering B mode or undergoing full cycles of N, B, E, N mode changes. The corresponding languages reflect either no copying (functioning as plain FSAs) or full copying, respectively.

In any specific run, it is the states that inform an FSBM M of its modality. The first time M reaches a G state, it has to enter B mode and keeps buffering when it transits between plain states. The first time when it reaches an H state, M is supposed to enter E mode and transit only between H states in E mode. Hence, to go through full cycles of mode changes, once M reaches a G state and switches to B mode, it has to encounter some H states later to be put in E mode. To allow us to only reason about only the "useful" arrangements of G and H states, we impose an ordering requirement on G and H states along a path in a machine and define a complete path.

Definition 5. A path s from an initial state to a final state in a machine is said to be complete if

1. for one H state in s, there is always a preceding G state;
2. once one G state is in s, s must contain must contain at least one H following that G state
3. in between G and the first H are only plain states.

Schematically, with P representing those non-G, non-H plain states and I, F representing initial, final states respectively, the regular expression denoting the state information in a path s should be of the form: $I(P^*GP^*HH^*P^* \mid P^*)^*F$.

Definition 6. A **complete-path** finite state buffered machine is an FSBM in which all possible paths are complete.

Example FSBMs we provide so far (Figure 3, Figure 4 and in Figure 8) are complete-path FSBMs. For the rest of this section, we describe several cases of an incomplete path in a machine M.

No H states When a G state does not have any reachable H state following it, there is no complete run, since M always stays in B mode.

No H states in between two G states When a G state q_0 has to transit to another G state q_0' before any H states, M cannot go to q_0', for M would enter B mode at q_0 while transiting to another G state in B mode is ill-defined.

H states first When M has to follow a path containing two consecutive H states before any G state, it would clash in the end, because the transitions among two H states can only be used in E mode. However, it is impossible to enter E mode without entering B mode enforced by some G states.

It should be emphasized that M in N mode can pass through one (and only one) H state to another plain state. For instance, the language of the FSBM

	Used Arc	*State Info*	*Configuration*	
1.	*N/A*	$q_1 \in I$	(*abbabb*, q_1, ϵ, N)	
2.	*N/A*	$q_1 \in G$	(*abbabb*, q_1, ϵ, B)	Buffering triggered by q_1 and empty buffer
3.	(q_1, a, q_2)	$q_2 \notin G$	(*bbabb*, q_2, *a*, B)	
4.	(q_2, b, q_3)		(*babb*, q_3, *ab*, B)	
5.	(q_3, b, q_3)		(*abb*, q_3, *abb*, B)	
6.	(q_3, ϵ, q_4)		(*abb*, q_4, *abb*, B)	Emptying triggered by q_4
7.	*N/A*		(*abb*, q_4, *abb*, E)	
8.	(q_4, a, q_4)		(*bb*, q_4, *bb*, E)	
9.	(q_4, b, q_4)		(*b*, q_4, *b*, E)	
10.	(q_4, b, q_4)	$q_4 \in H$	(ϵ, q_4, ϵ, E)	Normal triggered by q_4 and empty buffer
11.	*N/A*	$q_4 \in F$	(ϵ, q_4, ϵ, N)	

Figure 6: M_2 in Figure 4 accepts $abbabb$

	Used Arc	*State Info*	*Configuration*	
1.	*N/A*	$q_1 \in I$	(*ababb*, q_1, ϵ, N)	
2.	*N/A*	$q_1 \in G$	(*ababb*, q_1, ϵ, B)	Buffering triggered by q_1 and empty buffer
3.	(q_1, a, q_2)	$q_2 \notin G$	(*babb*, q_2, *a*, B)	
4.	(q_2, b, q_3)	$q_3 \in H$	(*abb*, q_3, *ab*, B)	
6.	(q_3, ϵ, q_4)		(*abb*, q_4, *ab*, B)	Emptying triggered by q_4
5.	*N/A*		(*abb*, q_4, *ab*, E)	
6.	(q_4, a, q_4)		(*bb*, q_4, *b*, E)	
7.	(q_4, b, q_4)	$q_4 \in H$	(*b*, q_4, ϵ, E)	Normal triggered by q_4 and empty buffer
8.	*N/A*		(*b*, q_4, ϵ, N)	
			Clash	

Figure 7: M_2 in Figure 4 rejects $ababb$

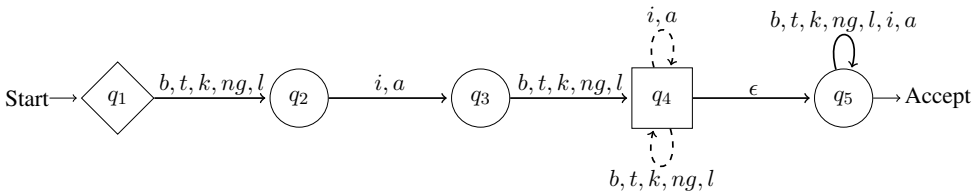

Figure 8: An FSBM M_3 for Agta CVC-reduplicated plurals: G = $\{q_1\}$ and H = $\{q_4\}$

	Used Arc	*State Info*	*Configuration*	
1.	*N/A*	$q_1 \in G$	(*taktakki*, q_1, ϵ, N)	Buffering triggered by q_1 and empty buffer
2.	*N/A*		(*taktakki*, q_1, ϵ, B)	
3.	(q_1, t, q_2)	$q_2 \notin G$	(*aktakki*, q_2, *t*, B)	
4.	(q_2, a, q_3)		(*ktakki*, q_3, *ta*, B)	
5.	(q_3, k, q_4)	$q_4 \in H$	(*takki*, q_4, *tak*, B)	Emptying triggered by q_4
6.	*N/A*		(*takki*, q_4, *tak*, E)	
7.	(q_4, t, q_4)		(*akki*, q_4, *ak*, E)	
8.	(q_4, a, q_4)		(*kki*, q_4, *k*, E)	
9.	(q_4, k, q_4)	$q_4 \in H$	(*ki*, q_4, ϵ, E)	Normal triggered by q_4 and empty buffer
10.	*N/A*		(ki, q_4, ϵ, N)	
11.	(q_4, ϵ, q_5)		(ki, q_5, ϵ, N)	
12.	(q_5, k, q_5)		(i, q_5, ϵ, N)	
13.	(q_5, i, q_5)	$q_5 \in F$	(ϵ, q_5, ϵ, N)	

Figure 9: M_3 in Figure 8 accepts $taktakki$

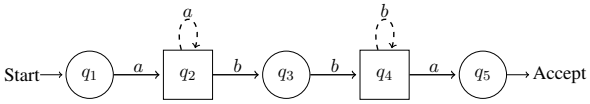

Figure 10: An incomplete FSBM M_4 with G = $\emptyset$ and H = $\{q_2, q_4\}$; $L(M_4) = \{abba\}$

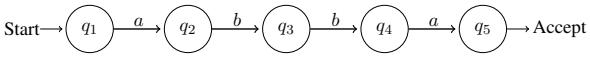

Figure 11: An FSA (or an FSBM with $G = \emptyset$ and $H = \emptyset$) whose language is equivalent as M_4 in Figure 10

M_4 in Figure 10 is equivalent to the language recognized by the FSA in Figure 11. M_4 remains to be an incomplete FSBM because it doesn't have any G state preceding the H states q_2 and q_4.

The languages recognized by complete-path FSBMs are precisely the languages recognized by general FSBMs. One key observation is the language recognized by the new machine is the union of the languages along all possible paths. Then, the validity of such a statement builds on different incomplete cases of G and H states along a path: they either recognize the empty-set language or show equivalence to finite state machines. Therefore, the language along an incomplete path of the machine is still in the regular set. Only a complete path containing at least one well-arranged $G \dots HH^*$ sequence uses the copying power and extends the regular languages. Therefore, in the next section, we focus on complete-path FSBMs.

3 Some closure properties of FSBMs

In this section, we show some closure properties of complete-path FSBM-recognizable languages and their linguistic relevance. Section 3.1 discusses its closure under intersection with regular languages; Section 3.2 shows it is closed under homomorphism; Section 3.3 briefly mentions union, concatenation, Kleene star. These operations are of special interests because they are regular operations defining regular expressions (Sipser, 2013, 64). That complete-path FSBMs are closed under regular operations leads to a conjecture that the set of languages recognized by the new automata is equivalent to the set of languages denoted by a version of regular expression with copying added.

Noticeably, given FSBMs are FSAs with a copying mechanism, the proof ideas in this section are similar to the corresponding proofs for FSAs, which can be found in Hopcroft and Ullman (1979) and Sipser (2013).

3.1 Intersection with FSAs

Theorem 1. *If L_1 is a complete-path FSBM-recognizable language and L_2 is a regular language, then $L_1 \cap L_2$ is a complete-path FSBM-recognizable language.*

In other words, if L_1 is a language recognized by a complete-path FSBM $M_1 = \langle Q_1, \Sigma, I_1, F_1, G_1, H_1, \delta_1 \rangle$, and L_2 is a language recognized by an FSA $M_2 = \langle Q_2, \Sigma, I_2, F_2, \delta_2 \rangle$, then $L_1 \cap L_2$ is a language recognizable by another complete-path FSBM. It is easy to construct an intersection machine M where $M = \langle Q, \Sigma, I, F, G, H, \delta \rangle$ with 1) $Q = Q_1 \times Q_2$; 2) $I = I_1 \times I_2$; 3) $F = F_1 \times F_2$; 4) $G = G_1 \times Q_2$; 5) $H = H_1 \times Q_2$; 6) $((q_1, q_1'), x, (q_2, q_2')) \in \delta$ iff $(q_1, x, q_2) \in \delta_1$ and $(q_1', x, q_2') \in \delta_2$. Paths in M would inherit the completeness from M_1 given the current construction. Then, $L(M) = L_1 \cap L_2$, as M simulates $L_1 \cap L_2$ by running M_1 and M_2 simultaneously. M accepts w if and only if both M_1 and M_2 accept w.

In nature, FSAs can be viewed as FSBMs without copying: they can be converted to an FSBM with an empty G set, an empty H set and trivially no special transitions between H states.

That FSBM-recognizable languages are closed under intersection with regular languages is of great relevance to phonological theory: assume a natural language X imposes backness vowel harmony, which can be modeled by an FSA M_{VH}. In addition, this language also requires phonological strings of certain forms to be reduplicated, which can be modeled by an FSBM M_{RED}. One hereby can construct another FSBM M_{RED+VH} to enforce both backness vowel harmony and the total identity of sub-strings in those forms. Not limited to harmony systems, phonotactics other than identity of sub-strings are regular (Heinz, 2018), indicating almost all phonological markedness constraints can be modeled by FSAs. When FSBMs intersect with FSAs computing those phonotactic restrictions, the resulting formalism is still an FSBM but not other grammar with higher computational power. Thus, FSBMs can model natural language phonotactics once including recognizing surface sub-string identity.

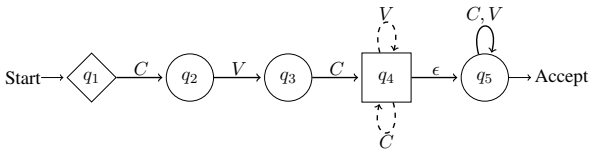

Figure 12: An FSBM M_5 on the alphabet $\{C, V\}$ such that $L(M_5) = h(L(M_3))$ with M_3 in Figure 8

3.2 Homomorphism and inverse alphabetic homomorphism

Definition 7. A (string) homomorphism is a function mapping one alphabet to strings of another alphabet, written $h : \Sigma \to \Delta^*$. We can extend h to operate on strings over Σ^* such that 1) $h(\epsilon_\Sigma) = \epsilon_\Delta$; 2) $\forall a \in \Sigma$, $h(a) \in \Delta^*$; 3) for $w = a_1 a_2 \dots a_n \in \Sigma^*$, $h(w) = h(a_1)h(a_2)\dots h(a_n)$ where each $a_i \in \Sigma$. An alphabetic homomorphism h_0 is a special homomorphism with h_0: $\Sigma \to \Delta$.

Definition 8. Given a homomorphism h: $\Sigma \to \Delta^*$ and $L_1 \subseteq \Sigma^*$, $L_2 \subseteq \Delta^*$, define $h(L_1) = \{h(w) \,|\, w \in L_1\} \subseteq \Delta^*$ and $h^{-1}(L_2) = \{w \,|\, h(w) = v \in L_2\} \subseteq \Sigma^*$.

Theorem 2. *The set of complete-path FSBM-recognizable languages is closed under homomorphisms.*

Theorem 2. can be proved by constructing a new machine M_h based on M. The informal intuition goes as follows: relabel the odd arcs to mapped strings and add states to split the arcs so that there is only one symbol or ϵ on each arc in M_h. When there are multiple symbols on normal arcs, the newly added states can only be plain non-G, non-H states. For multiple symbols on the special arcs between two H states, the newly added states must be H states. Again, under this construction, complete paths in M lead to newly constructed complete paths in M_h.

The fact that complete-path FSBMs guarantee the closure under homomoprhism allows theorists to perform analyses at certain levels of abstraction of certain symbol representations. Consider two alphabets $\Sigma = \{b, t, k, ng, l, i, a\}$ and $\Delta = \{C, V\}$ with a homomorphism h mapping every consonant (b, t, k, ng, l) to C and mapping every vowel (i, a) to V. As illustrated by M_3 on alphabet Σ (Figure 8) and M_5 on alphabet Δ (Figure 12), FSBM-definable patterns on Σ would be another FSBM-definable patterns on Δ.

We conjecture that the set of languages recognized by complete-path FSBMs is not closed under inverse alphabetic homomorphisms and thus inverse homomorphism. Consider a complete-path FSBM-recognizable language $L = \{a^i b^j a^i b^j \,|\, i, j \geq 1\}$ (cf. Figure 4). Consider an alphabetic homomorphism $h : \{0, 1, 2\} \to \{a, b\}^*$ such that $h(0) = a$, $h(1) = a$ and $h(2) = b$. Then, $h^{-1}(L) = \{(0|1)^i 2^j (0|1)^i 2^j \,|\, i, j \geq 1\}$ seems to be challenging for FSBMs. Finite state machines cannot handle the incurred crossing dependencies while the augmented copying mechanism only contributes to recognizing *identical* copies, but not general cases of symbol correspondence. [5]

3.3 Other closure properties

Union Assume there are complete-path FSBMs M_1 and M_2 such that $L(M_1) = L_1$ and $L(M_2) = L_2$, then $L_1 \cup L_2$ is a complete-path FSBM-recognizable language. One can construct a new machine M that accepts an input w if either M_1 or M_2 accepts w. The construction of M keeps M_1 and M_2 unchanged, but adds a new plain state q_0. Now, q_0 becomes the only initial state, branching into those previous initial states in M_1 and M_2 with ϵ-arcs. In this way, the new machine would guess on either M_1 or M_2 accepts the input. If one accepts w, M will accept w, too.

Concatenation Assume there are complete-path FSBMs M_1 and M_2 such that $L(M_1) = L_1$ and $L(M_2) = L_2$, then there is a complete-path FSBM M that can recognize $L_1 \circ L_2$ by normal concatenation of two automata. The new machine adds a new plain state q_0 and makes q_0 the only initial state, branching into those previous initial states in M_1 with ϵ-arcs. All final states in M_2 are the only final states in M. Besides, the new machine adds ϵ-arcs from any old final states in M_1 to any possible initial states in M_2. A path in the resulting machine is guaranteed to be complete because it is essentially the concatenation of two complete paths.

Kleene Star Assume there is a complete-path FSBM M_1 such that $L(M_1) = L_1$, L_1^* is a complete-path FSBM-recognizable language. A new automaton M is similar to M_1 with a new initial state q_0. q_0 is also a final state, branching into

[5] The statement on the inverse homomorphism closure is left as a conjecture. We admit that a more formal and rigorous mathematical proof proving $h^{-1}(L)$ is not complete-path FSBM-recognizable should be conducted. To achieve this goal, a more formal tool, such as a developed pumping lemma for the corresponding set of languages, is important.

old initial states in M_1. In this way, M accepts the empty string ϵ. q_0 is never a G state nor an H state. Moreover, to make sure M can jump back to an initial state after it hits a final state, ϵ-arcs from any final state to any old initial states are added.

4 Discussion and conclusion

In summary, this paper provides a new computational device to compute unrestricted total reduplication on any regular languages, including the simplest copying language L_{ww} where w can be any arbitrary string of an alphabet. As a result, it introduces a new class of languages incomparable to CFLs. This class of languages allows *unbounded* copying without generating non-reduplicative non-regular patterns: we hypothesize context-free string reversals are excluded since the buffer is queue-like. Meanwhile, the MCS Swiss-German cross-serial dependencies, abstracted as $\{a^i b^j c^i d^j \mid i, j \geq 1\}$, is also excluded, since the buffer works on the same alphabet as the input tape and only matches *identical* sub-strings.

Following the sub-classes of 2-way FSTs in Dolatian and Heinz (2018a,b, 2019, 2020), which successfully capture unbounded copying as *function*s while exclude the mirror image mapping, complete-path FSBMs successfully capture the total-reduplicated stringsets while exclude string reversals. Comparison between the characterized languages in this paper and the image of functions in Dolatian and Heinz (2020) should be further carried out to build the connection. Moreover, one natural next step is to extend FSBMs as acceptors to finite state buffered transducers (FSBT). Our intuition is FSBTs would be helpful in handling the morphological analysis question ($ww \rightarrow w$), a not-yet solved problem in the 2-way FSTs that Dolatian and Heinz (2020) study. After reading the first w in input and buffering this chunk of string in the memory, the transducer can output ϵ for each matched symbol when transiting among H states.

Another potential area of research is applying this new machinery to Primitive Optimality Theory (Eisner, 1997; Albro, 1998). Albro (2000, 2005) used weighted finite state machine to model constraints while represented the set of candidates by Multiple Context Free Grammars to enforce base-reduplicant correspondence (McCarthy and Prince, 1995). Parallel to Albro's way, given complete-path FSBMs are intersectable with FSAs, it is possible to computationally implement the reduplicative identity requirement by complete-path FSBMs without using the full power of mildly context sensitive formalisms. To achieve this goal, future work should consider developing an efficient algorithm that intersects complete-path FSBMs with *weighted* FSAs.

The present paper is the first step to recognize reduplicated forms in adequate yet more restrictive models and techniques compared to MCS formalisms. There are some limitations of the current approach on the whole typology of reduplication. Complete-path FSBMs can only capture local reduplication with *two* adjacent identical copies. As for non-local reduplication, the modification should be straightforward: the machines need to allow the filled buffer in N mode (or in another newly-defined memory holding mode) and match strings only when needed. As for multiple reduplication, complete-path FSBMs can easily be modified to include multiple copies of the same base form ($\{w^n \mid w \in \Sigma^*, n \in \mathbb{N}\}$) but cannot be easily modified to recognize the non-semilinear language containing copies of the copy ($\{w^{2^n} \mid w \in \Sigma^*, n \in \mathbb{N}\}$). It remains to be an open question on the computational nature of multiple reduplication. Last but not the least, as a reviewer points out, recognizing non-identical copies can be achieved by either storing or emptying not exactly the same input symbols, but mapped symbols according to some function f. Under this modification, the new automata would recognize $\{a^n b^n \mid n \in \mathbb{N}\}$ with $f(a) = b$ but still exclude string reversals. In all, detailed investigations on how to modify complete-path FSBMs should be the next step to complete the typology.

Acknowledgments

The author would like to thank Tim Hunter, Bruce Hayes, Dylan Bumford, Kie Zuraw, and the members of the UCLA Phonology Seminar for their feedback and suggestions. Special thanks to the anonymous reviewers for their constructive comments and discussions. All errors remain my own.

References

Daniel M Albro. 1998. Evaluation, implementation, and extension of primitive optimality theory. Master's thesis, UCLA.

Daniel M. Albro. 2000. Taking primitive Optimality Theory beyond the finite state. In *Proceedings of the Fifth Workshop of the ACL Special Interest Group*

in Computational Phonology, pages 57–67, Centre Universitaire, Luxembourg. International Committee on Computational Linguistics.

Daniel M Albro. 2005. *Studies in computational optimality theory, with special reference to the phonological system of Malagasy*. Ph.D. thesis, University of California, Los Angeles, Los Angeles.

Bruce Bagemihl. 1989. The crossing constraint and 'backwards languages'. *Natural language & linguistic Theory*, 7(4):481–549.

Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. 2017. Untwisting two-way transducers in elementary time. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12.

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

Jane Chandlee. 2014. *Strictly local phonological processes*. Ph.D. thesis, University of Delaware.

Jane Chandlee. 2017. Computational locality in morphological maps. *Morphology*, 27:599–641.

Noam Chomsky. 1956. Three models for the description of language. *IRE Trans. Inf. Theory*, 2:113–124.

Alexander Clark and Ryo Yoshinaka. 2014. Distributional learning of parallel multiple context-free grammars. *Mach. Learn.*, 96(1–2):5–31.

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

Christopher Culy. 1985. The complexity of the vocabulary of bambara. *Linguistics and philosophy*, 8(3):345–351.

Robert M. W. Dixon. 1972. *The Dyirbal Language of North Queensland*, volume 9 of *Cambridge Studies in Linguistics*. Cambridge University Press, Cambridge.

Hossep Dolatian and Jeffrey Heinz. 2018a. Learning reduplication with 2-way finite-state transducers. In *Proceedings of the 14th International Conference on Grammatical Inference*, volume 93 of *Proceedings of Machine Learning Research*, pages 67–80. PMLR.

Hossep Dolatian and Jeffrey Heinz. 2018b. Modeling reduplication with 2-way finite-state transducers. In *Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pages 66–77, Brussels, Belgium. Association for Computational Linguistics.

Hossep Dolatian and Jeffrey Heinz. 2019. RedTyp: A database of reduplication with computational models. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2019*, pages 8–18.

Hossep Dolatian and Jeffrey Heinz. 2020. Computing and classifying reduplication with 2-way finite-state transducers. *Journal of Language Modelling*, 8(1):179–250.

Jason Eisner. 1997. Efficient generation in primitive Optimality Theory. In *35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics*, pages 313–320, Madrid, Spain. Association for Computational Linguistics.

Gerald Gazdar and Geoffrey K Pullum. 1985. Computationally relevant properties of natural languages and their grammars. *New generation computing*, 3(3):273–306.

Thomas Graf. 2017. The power of locality domains in phonology. *Phonology*, 34(2):385–405.

Phyllis M. Healey. 1960. *An Agta Grammar*. Bureau of Printing, Manila.

Jeffrey Heinz. 2007. *The Inductive Learning of Phonotactic Patterns*. Ph.D. thesis, University of California, Los Angeles.

Jeffrey Heinz. 2018. The computational nature of phonological generalizations. In Larry Hyman and Frans Plank, editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton.

Jeffrey Heinz, Chetan Rawal, and Herbert G Tanner. 2011. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human language technologies*, pages 58–64.

John E Hopcroft and Jeffrey D Ullman. 1979. Introduction to automata theory, languages, and computation. *Addison-Welsey, NY*.

Mans Hulden. 2009. *Finite-state Machine Construction Methods and Algorithms for Phonology and Morphology*. Ph.D. thesis, University of Arizona, Tucson, USA.

Sharon Inkelas and Cheryl Zoll. 2005. *Reduplication: Doubling in morphology*, volume 106. Cambridge University Press.

Gerhard Jäger and James Rogers. 2012. Formal language theory: refining the chomsky hierarchy. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1598):1956–1970.

C. Douglas Johnson. 1972. *Formal Aspects of Phonological Description*. Monographs on linguistic analysis. Mouton, The Hague.

Aravind K. Joshi. 1985. *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?*, Studies in Natural Language Processing, page 206–250. Cambridge University Press.

Ronald M. Kaplan and Martin Kay. 1994. Regular models of phonological rule systems. *Comput. Linguist.*, 20(3):331–378.

Alec Marantz. 1982. Re reduplication. *Linguistic inquiry*, 13(3):435–482.

John J. McCarthy and Alan S. Prince. 1995. Faithfulness and reduplicative identity. In Jill N. Beckman, Laura Walsh Dickey, and Suzanne Urbanczyk, editors, *Papers in Optimality Theory*. GLSA (Graduate Linguistic Student Association), Dept. of Linguistics, University of Massachusetts, Amherst, MA.

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

Brian Roark and Richard Sproat. 2007. *Computational approaches to morphology and syntax*, volume 4. Oxford University Press.

Carl Rubino. 2013. Reduplication. In Matthew S. Dryer and Martin Haspelmath, editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Stuart M Shieber. 1985. Evidence against the context-freeness of natural language. In *Philosophy, Language, and Artificial Intelligence*, pages 79–89. Springer.

Imre Simon. 1975. Piecewise testable events. In *Automata Theory and Formal Languages*, pages 214–222, Berlin, Heidelberg. Springer Berlin Heidelberg.

Michael Sipser. 2013. *Introduction to the Theory of Computation*, third edition. Course Technology, Boston, MA.

Edward Stabler. 1997. Derivational minimalism. In *Logical Aspects of Computational Linguistics*, pages 68–95, Berlin, Heidelberg. Springer Berlin Heidelberg.

Markus Walther. 2000. Finite-state reduplication in one-level prosodic morphology. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*.

Kie Zuraw. 2002. Aggressive reduplication. *Phonology*, 19(3):395–439.