On regular copying languages

Yang Wang and Tim Hunter
University of California, Los Angeles

ABSTRACT

This paper proposes a formal model of regular languages enriched with unbounded copying. We augment finite-state machinery with the ability to recognize copied strings by adding an unbounded memory buffer with a restricted form of first-in-first-out storage. The newly introduced computational device, finite-state buffered machines (FSBMs), characterizes the class of regular languages and languages derived from them through a primitive copying operation. We name this language class *regular copying languages* (RCLs). We prove a pumping lemma and examine the closure properties of this language class. As suggested by previous literature (Gazdar and Pullum 1985, p.278), regular copying languages should approach the correct characterization of natural language word sets.

Keywords: reduplication, copying, finite-state machinery, queue automata

INTRODUCTION 1

The aim of this paper is to introduce a formal model of possible natural language word forms which is restrictive enough to rule out many unattested patterns, but still expressive enough to allow for reduplication. Among the well-known existing classes of formal languages, there is a tension between these two goals. The overwhelming majority of attested phonological patterns fall within the finite-state class (Kaplan and Kay 1994), and perhaps within even more restrictive

subclasses (Heinz 2007). Reduplication is the striking exception to this generalization. But at present, if we look for alternatives to the finite-state characterization which are powerful enough to express reduplication, we only find classes of formal languages which additionally allow a wide variety of unattested patterns — for example, nesting/mirror-image patterns, or arbitrary cross-serial dependency patterns significantly more general than reduplication itself. This gives us no way to retain the finite-state characterization's (apparently correct) prediction that mirror-image patterns and so on will be unattested, while avoiding the (apparently incorrect) prediction that reduplication will be unattested.

Jäger and Rogers (2012) review other cases where natural language generalizations do not appear to correspond neatly to degrees of complexity as defined by the formalisms of the classical Chomsky Hierarchy, and the "refinements" of the hierarchy that these findings have prompted. In the case of natural language syntax, for example, it is widely accepted that context-free grammars are insufficiently expressive (Huybregts 1984; Shieber 1985; Culy 1985); but the next level up on the classical hierarchy, context-sensitive grammars, are far too expressive to be a plausible characterization of possible natural languages. This situation prompted the development of many *mildly context-sensitive* formalisms (Joshi 1985; Kallmeyer 2010), whose generative capacity sits in between the context-free and context-sensitive levels. Another "mismatch" has been observed in phonology, where even the lowest level of the classical hierarchy, the finite-state languages, has been argued to be insufficiently restrictive. To address this, a number of researchers have developed *sub-regular* formalisms (e.g., Heinz *et al.* 2011; Chandlee 2014; Heinz 2018).

In this paper, the situation we are addressing is slightly less straightforward than the two mismatches just mentioned. The development of sub-regular formalisms was a response to a perception that *all* the levels of the classical hierarchy were too powerful. The mildly context-sensitive formalisms address the fact that, with regard to syntax, each of the classical levels is either too weak (finite-state, context-free) or too powerful (context-sensitive, recursively enumerable). The situation we address in this paper, in contrast, is one where the classical context-free class is both too powerful in some ways (since it allows mirror-image patterns) and too restrictive in other

ways (since it disallows reduplication). We, therefore, seek a formalism that *cuts across* the levels of the classical hierarchy, rather than one which adds a level that sits within the existing hierarchical relationships.

We introduce *finite-state buffered machines (FSBMs)* as a step towards solving this problem. The idea is to preserve as much as possible of the restrictiveness of the finite-state class and add "just" what is necessary to generate copying patterns. FSBMs include unbounded memory in the form of a first-in-first-out buffer, but the use of this memory is restricted in two important ways. First, this memory buffer uses the alphabet of surface symbols, rather than a separate alphabet like the stack alphabet of a pushdown automaton (PDA). Second, the allowable ways of interacting with this memory buffer are closely tied to the surface string being generated: the only storage operation adds a copy of the current surface symbol to the memory buffer, and the only retrieval operation empties the entire memory buffer and adds its contents to the generated string. For example, in computing a string of the form $urrv$, an FSBM will proceed through three phases corresponding to the sub-strings u, r and v, much like a standard finite-state machine generating the string urv. But throughout the middle phase, a copy of each surface symbol of r will be stored in the FSBM's memory buffer, and at the transition from this middle phase to the third phase the buffer will be emptied and its contents appended to the computed string; thus ur has r appended to it, before the machine proceeds to compute the v portion in the third phase.

In Section 2 we discuss the computational challenge posed by reduplication in more detail, and outline the ways our approach differs from a number of other attempts to enrich otherwise restrictive formalisms with copying mechanisms. We present FSBMs in full in Section 3, give a pumping lemma in Section 4, and explore the mathematical properties of the generated class of languages in Section 5. Section 6 discusses some remaining issues, including various kinds of non-canonical reduplication, and a formal distinction between what we will call *symbol-oriented* generative mechanisms (such as string-copying) and the better-known mechanisms underlying the classical Chomsky Hierarchy. Section 7 concludes the paper.

2 BACKGROUND

Section 2.1 outlines the important empirical properties of reduplication that make it a poor fit to the classical Chomsky Hierarchy; in particular, we aim to show that an appropriate characterization of possible natural language word forms should include the pattern ww, for unboundedly many strings w, but not ww^R, where w^R is the reverse of w. Section 2.2 reviews various modifications to classical automata, like our proposal, that incorporate some form of unbounded queue-like memory. In Section 2.3 we discuss other modifications to finite-state automata that were motivated by reduplication, but do not accommodate the crucial property of unboundedness.

2.1 *The puzzle of reduplication*

2.1.1 Reduplication in natural languages

Reduplication, creating identity within word forms, is common cross-linguistically. Table 1 provides illustrative examples. Dyirbal exhibits *total reduplication*, with the plural form of a nominal comprised of two perfect copies of the full singular stem; whereas *partial reduplication* is exemplified in Agta, where plural forms only copy the first CVC sequence of the corresponding singular forms (Healey 1960; Marantz 1982).[1] In the sample reported by Rubino (2013) and further surveyed in Dolatian and Heinz (2020), 313 out of 368 natural languages exhibit productive reduplication, of which 35 languages have total reduplication but not partial reduplication. Moravcsik (1978, p. 328) hypothesized that all languages with attested partial reduplication would also use total reduplication.

By comparison, context-free palindrome patterns are rare in phonology and morphology (Marantz 1982) and appear to be confined to language games (Bagemihl 1989; Gil 1996), whose phonological status is unclear. Figure 1 illustrates the important difference between Dyirbal total reduplication (‘*midi-midi*’) and the logically-possible but unattested palindrome pattern (‘*midi-idim*’).

Total reduplication: Dyirbal plurals (Dixon 1972, p. 242; Inkelas 2008, p. 352)			
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, p.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)*

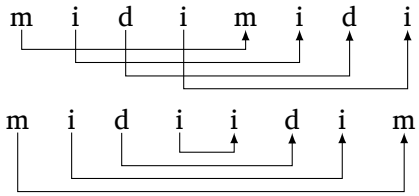


Figure 1: *Crossing dependencies in Dyirbal total reduplication 'midi-midi' (top) versus nesting dependencies in unattested string reversal 'midi-idim' (bottom)*

From the perspective of a computational analysis, it will be important to establish that (at least some) reduplication constructions are *unbounded*, in the sense that they are usefully modeled by string-sets of the form $\{ww \mid w \in S\}$ for some infinite set S. A partial reduplication construction, such as the Agta case above where an initial CVC sequence is copied, is obviously not unbounded in this sense, since — assuming a finite alphabet — there are only finitely-many CVC sequences (Chandlee and Heinz 2012).[2] But as observed by Clark and Yoshinaka (2014) and Chandlee (2017), even amongst total reduplication constructions we must take care to distinguish between unrestricted, productive total reduplication (which is unbounded in the

[1] For clarity, we adopt a simplistic analysis here. When the bases start with a vowel, Agta copies the first VC sequence, as in *uffu* 'thigh' and *uf-uffu* 'thighs'. Thus, a more complete generalization is that Agta copies a (C)VC sequence.

relevant sense) and total reduplication on a *finite* set of bases. For example, it is important to establish that *midi-midi* is not simply part of a collection $\{ww \mid w \in S\}$ where S is some finite memorized set (e.g. the set of all lexemes of a particular category); in such a case, the resulting set of reduplicated forms would itself be finite, and therefore within most familiar language classes. Table 2 illustrates the relationship between productivity, the partial/total distinction, and unboundedness.

	Restricted to lexemes (not productive)	Not restricted to lexemes (productive)
Partial Reduplication	**Bounded**	**Bounded**
Total Reduplication	**Bounded**	**Unbounded**

Table 2: *Reduplication and bounded/unbounded copying*

A famous case of reduplication that is unbounded in the relevant sense is the Bambara 'Noun *o* Noun' construction (Culy 1985). For example, the stem **wulu** *dog* can be copied to form **wulu o wulu** *whichever dog*. The important point about productivity comes from the interaction of this reduplication with the agentive *la* construction, illustrated in (1) (Culy 1985, pp. 346–347).

(1) a. wulu + nyini + la = wulunyinina
dog search for
"one who searches for dogs", i.e., "dog searcher"

b. wulu + filè + la = wulufilèla
dog watch
"one who watches dogs", i.e., "dog watcher"

This agentive construction itself is recursive, in the sense that it can build on its own outputs, as illustrated in (2); and the outputs of the agentive construction, including the recursively-formed ones, can be

[2] In principle, a reduplication operation which copied, for example, *half* of the relevant stem, would be a case of unbounded copying in this sense that would likely nonetheless be described as partial reduplication. But the attested cases of "partial reduplication" appear to all involve templates that do not depend on the length of the base (see the most frequent attested shapes in Moravcsik 1978; Rubino 2005; Dolatian and Heinz 2020), like the Agta examples above.

used in the 'Noun *o* Noun' reduplicative construction, as illustrated in (3).

(2) a. wulunyinina + nyini + la = wulunyininanyinina
dog searcher search for
"one who searches for dog searchers"

b. wulunyinina + filè + la = wulunyininafilèla
dog searcher watch
"one who watches dog searchers"

(3) a. wulunyinina **o** wulunyinina
dog searcher dog searcher
(1a) (1a)
"whichever dog searcher"

b. wulufilèla **o** wulufilèla
dog watcher dog watcher
(1b) (1b)
"whichever dog watcher"

c. wulunyininanyinina **o** wulunyininanyinina
(2a) (2a)
"whichever one who searches for dog searchers"

d. wulunyininafilèla **o** wulunyininafilèla
(2b) (2b)
"whichever one who watches dog searchers"

The set of all outputs of this reduplication process can therefore naturally be thought of as taking the form $\{ww \mid w \in S\}$, where S is the *infinite* set of nouns, including outputs of the agentive construction.

Further evidence that reduplication is productive in this sense comes from its applicability to borrowed words: Yuko (2001, p. 68) cites the totally-reduplicated plurals *teknik-teknik* 'techniques' and *teknologi-teknologi* 'technologies' attested in Malay, for example. Similarly, the code-switching data from Tagalog in (4) (Waksler 1999), shows the English word 'swimming' being (partially) reduplicated.

(4) Saan si Jason? Nag-SWI-SWIMMING siya.
where DET Jason PRESENT-REDUP-SWIMMING he
'Where is Jason? He's swimming.'

In addition, in a few experiments that, either directly or indirectly, study the learnability of surface identity-based patterns, copying appears to be salient and easy to learn. The famous study by Marcus *et al.* (1999) shows that infants can detect and habituate to different identity-based patterns: ABA vs. ABB and AAB vs. ABB, where A and B are CV syllables. Crucially, the particular syllables used at test time were distinct from any seen during training.

Evidence that reduplication/copying (ww) patterns have an importantly different status than reversal (ww^R) patterns — converging with the typological absence of reversal patterns noted above — comes from one recent artificial grammar learning study (Moreton *et al.* 2021). In this experiment, adult learners were trained to identify either a reduplication or a syllable reversal pattern. Participants were also asked to explicitly state the rule they had learned (if they could). Participants in the reduplication group showed final above-chance performance whether they could state the rule or not. However, in the syllable-reversal condition, only participants who could also correctly state the rule showed final above-chance performance; this suggests that learning the reversal pattern relied on some degree of explicit/conscious reasoning that the copying pattern did not. In further support of this distinction, correct syllable-reversal responses showed longer reaction times than correct copying responses. In a second variant of this experiment, the training phase was replaced with explicit instruction on the rule to apply; participants in the reduplication group still showed shorter reaction times. These results suggest that, to the extent that reversal patterns can be learned or applied at all, this is achieved more by conscious application of a rule rather than unconscious linguistic knowledge, in contrast to reduplication.

A significant aspect of this AGL study is that the stimuli used were auditory, "*purely phonological*", "*meaningless*" strings (Moreton *et al.* 2021, p. 9), chunks of which are identical. We take this to indicate that cognitively representable reduplication or reduplication-like patterns need not be realizations of meaning-changing operations: identity between sub-strings can contribute to the phonotactic well-formedness of a surface form, in ways that can be separated from any morphological paradigms in which that surface form appears. This aligns with the general tendency that Zuraw (2002) called *aggressive reduplication*: human phonological grammar is sensitive to output forms with

self-similar subparts, regardless of morphosyntactic or semantic cues. Such sensitivity is formalized as the constraint REDUP which requires string-to-string correspondence by coupling sub-strings together.[3]

Inadequacy of familiar language classes 2.1.2

Having established that the formal pattern *ww*, for unboundedly many strings *w*, is a reasonable model for reduplication, we can ask where this falls on the hierarchy of familiar language classes. The original Chomsky Hierarchy, shown in solid lines in Figure 2, classifies the *ww* pattern as properly context-sensitive; it is also included in the more recent *mildly context-sensitive* subclass (MCS; Joshi 1985; Stabler 2004), shown with a dashed line. This creates a puzzle with two parts.

The first part of the puzzle comes from the fact that reduplication is a counter-example to the otherwise overwhelming generalization that attested phonological and morphological patterns are regular. Aside from reduplication, it is very natural to hypothesize that the set of possible natural language word forms is regular (or even sub-regular). This is why the distinction above between bounded and unbounded copying is crucial: one way to save the regular hypothesis would be to demonstrate that reduplication is bounded, which would place it in the class of *finite* languages which is properly included in all of the classes shown in Figure 2. For example, Figure 3 shows a finite state automaton that successfully recognizes $\{ww \mid w \in S\}$ with a finite $S = \{aaa, aba, aab, abb, baa, bba, bab, bbb\}$. The finiteness

[3] Direct evidence supporting aggressive reduplication comes from pseudo-reduplication. A pseudo-reduplicated word has one portion identical to another portion. But the decomposed form cannot stand alone and thus does not bear proper morphosyntactic or semantic information. Zuraw (2002) studied the transparency of phonological rule application within pseudo-reduplicated words in Tagalog loan words. For example, stem-final mid vowels in Tagalog usually raise to high vowels when suffixed, as in [ka:l**o**s] *'grain leveller'* but [kal**u**s-in] *'to use a grain leveller on'*. However, within English and Spanish loans, mid vowel raising is less frequently applied when a preceding mid vowel is present: /tod**o** + in/ *'to include all'* surfaces as [tod**o**-in] but not [tod**u**-in]. The hypothesized motivation is that speakers preserve sub-string similarity between /to/ and /do/. A recent MEG study on visual inputs (Wray *et al.* 2022) further supports the reduplication-like representation for those pseudo-reduplicated words that fail to undergo a process due to similarity preservation.

makes it possible to essentially just memorize the desired list of surface forms.[4]

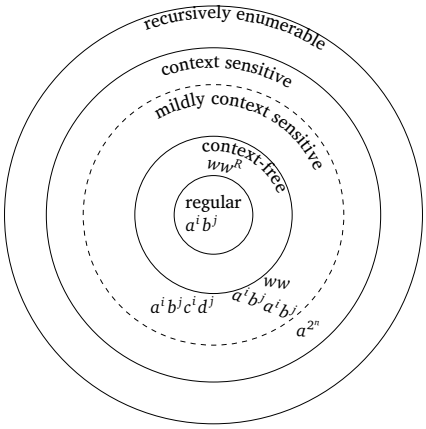


Figure 2: *Familiar language classes*

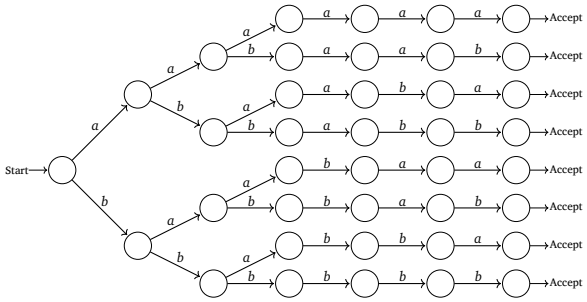


Figure 3: *A finite-state machine for whole-base copying with the set of bases* $=$ {*aaa, aab, aba, abb, baa, bab, bba, bbb*}

The second part of the puzzle comes from considering the classes in Figure 2 that do include *ww*. The most restrictive of these is the

[4] Of course one might also dispute whether Figure 3, with its explosion in the number of states (Roark and Sproat 2007; Dolatian and Heinz 2020), represents a linguistically adequate model of even a bounded copying construction. The distinction between arguing that Figure 3 is linguistically inadequate and arguing that copying is unbounded is subtle (Savitch 1993).

	linear/regular	nested	cross-serial
Morphology and Phonology	✓	✗	✓ restricted to symbol identity
Syntax	✓	✓	✓

Figure 4: *Attested types of dependencies in different language modules*

mildly context-sensitive class. This is not a good fit with natural language word forms because it also includes the ww^R pattern, which is unattested as discussed above; more generally, it includes *nesting* patterns as well as *crossing* patterns (recall Figure 1). But the problem is slightly more subtle than the simple distinction between nesting and crossing suggests: the MCS class includes very general crossing patterns such as $a^i b^j c^i d^j$, but reduplication represents a special case where the cross-serially dependent elements are identical symbols. MCS grammars are motivated by natural language syntax, where the more general kind of crossing patterns appear to be necessary[5] — the influential paper by Shieber (1985) on Swiss German appeals to exactly the aforementioned example $a^i b^j c^i d^j$ — but for the purposes of morphophonology, there is reason to distinguish crossing patterns that involve surface symbol identity (e.g. ww and $a^i b^j a^i b^j$) from those that do not. This situation is summarized in Figure 4. We return to the distinction between formalisms where symbol identity plays a role and those where it does not in Section 6.3.

2.2 *Language classes motivated by reduplication and queue automata*

In response to essentially the puzzle introduced above, Gazdar and Pullum (1985, p.287) 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

[5] And nesting patterns are at least as common as crossing patterns.

> it currently looks as if this class might be relevant to the characterization of NL [natural language] word-sets.

One such proposal is offered by Manaster-Ramer (1986, p.87), who introduces the idea — closely related to that underlying our own proposal below — as follows:[6]

> Rather than grudgingly clambering up the Chomsky Hierarchy towards Context-sensitive Grammars, we should consider going back down to Regular Grammars and striking out in a different direction. The simplest alternative proposal is a class of grammars which intuitively have the same relation to queues that CFGs have to stacks.

The *Context-free Queue Grammars* (CFQGs) that Manaster-Ramer proposes adopt the format of right-linear rewrite rules for regular grammars (i.e. valid rule forms are 'A → a B' and 'A → a'), with an additional queue-based memory in which a string of terminal symbols can be accumulated. The queue-based memory is implemented by the additional capability to write terminal symbols at the right end of the output string — not only to the right of the current nonterminal, but also any terminals previously added to this queue.

There are significant similarities between CFQGs and the FSBM formalism that we introduce in this paper. Manaster-Ramer illustrates CFQGs via an example that generates $\{ww \mid w \in \{\mathrm{a}, \mathrm{b}\}^*\}$, and conjectures that they cannot generate the corresponding mirror-image (ww^R) language, but there is no careful exploration of the formalism's capacity or limitations. Also, it is clear that CFQGs can generate more general crossing patterns such as $a^i b^j c^i d^j$ along with reduplication-like patterns, so FSBMs are more restricted in at least this (linguistically well-motivated) respect.

Along similar lines to Manaster-Ramer's proposal, Savitch (1989) introduced *Reduplication PDAs* (RPDAs), which are pushdown automata augmented with the ability to match reduplicated strings by

[6] Taken literally, this quotation seems to lead in the direction of unrestricted queue automata which are known to be equivalent to Turing machines. What Manaster-Ramer actually proposes is significantly more restricted. Also, see Kutrib *et al.* (2018) for a more complete review of the history of queue automata and investigations on restricted versions that computer scientists have conducted.

using a portion of the stack as a queue. RPDAs are more powerful than CFQGs, since the language class they define properly includes context-free languages, so they do not exclude nesting/mirror-image patterns. This aligns with the fact that the motivations Savitch discusses mainly involve crossing patterns found in syntax rather than identity-based reduplication which is our focus here. But the technical formulation of RPDAs has much in common with that of FSBMs below.

Finally, *Memory Automata* (MFAs; Schmid 2016; Freydenberger and Schmid 2019) introduce a kind of automata that is particularly similar to FSBMs. MFAs augment classical FSAs with a finite number of memory cells; each memory cell can store an unboundedly long sub-string of input, which can be matched against future input when it is recalled. The full class of MFAs can generate languages such as $\{a^i \mid i \text{ is not prime}\}$ (Câmpeanu *et al.* 2003, p.1013) and $\{a^{4^i} \mid i \geq 1\}$ (Freydenberger and Schmid 2019, p.21), and is therefore much too powerful to be suitable as a model for natural languages.[7] But these unusually "complex" languages all rely on either interactions between distinct memory cells, or the ability to recall a particular string from a memory cell more than once. The FSBM formalism that we introduce corresponds closely to a restricted version of MFAs where there is only one memory cell, and its contents are erased when recalled.

To summarize: our goal is to identify a formalism whose class of languages aligns with Gazdar and Pullum's motivating quotation above; RPDAs do not match this description because they extend upwards from the context-free languages, rather than the regular languages; CFQGs and MFAs do adopt the regular languages as the starting point, but extend too far and therefore overshoot the mark in different ways.

[7] MFAs were introduced to provide an automaton-based characterization of the languages generated by regular expressions extended with back-references (Câmpeanu *et al.* 2002; Câmpeanu *et al.* 2003; Carle and Narendran 2009). There are some differences between the various definitions of these "extended regular expressions" in the literature; see Freydenberger and Schmid (2019, pp. 36–37) for discussion. We would like to thank an anonymous reviewer for pointing out the relevant research on extended regular expressions, which in turn led us to the literature on MFAs.

This paper introduces FSBMs as a way of examining what minimal changes can be brought to regular languages to include string-sets with two copies of the same sub-strings, while excluding some typologically unattested context-free patterns, such as reversals, and crossing dependencies other than reduplication. We name the resulting class of languages *regular copying languages* (RCLs). The intended relation of this language class to other existing language classes is shown in Figure 5.

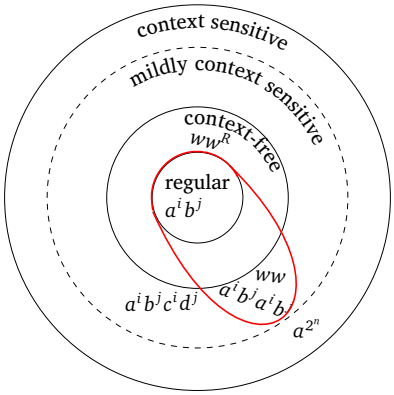


Figure 5: *The class of regular copying languages (oval shape) in the classical Chomsky Hierarchy*

2.3 *Other computational models motivated by reduplication*

Now we review other computational models motivated by reduplication, which can be categorized into two groups: those that limit attention to bounded copying, (Section 2.3.1), and those that consider transductions/mappings (Section 2.3.2).

2.3.1 Compact representations of bounded copying

The first line of work aims to improve upon the inelegant "memorization" strategy exemplified in Figure 3, while retaining the limitation to bounded copying. For example, Cohen-Sygal and Wintner (2006)

introduce *finite-state registered automata* (FSRAs), which augment standard FSAs with finitely many memory registers. This allows for a more space-efficient representation of copying patterns, without the "duplicating paths" of Figure 3, by storing the symbols to be matched in registers rather than in the machine's central state. But because the registers themselves provide only a finite amount of additional memory, FSRAs do not extend upon the generative capacity of standard FSAs, and therefore do not accommodate productive total reduplication (i.e. unbounded copying).

An analogous proposal is the *compile-replace* algorithm (Beesley and Karttunen 2000). This run-time technique first maps a lexical item to a regular expression representation for either morphological generation or analysis. Then the desired output is obtained by re-evaluating the output regular expression. Similarly, Walther (2000) added different types of transitions to represent the lexicon: *repeat* (for copying), *skip* (for truncation) and *self-loops* (for infixation). Then, intersecting these enriched lexical items with an FSA encoding language-specific reduplication rules would derive the surface strings. Last but not least, Hulden (2009) introduced an EQ function, a filter on a finite-state transduction which excludes input-output pairs where the output string does not meet a sub-string identity condition. In principle, this idea allows for an unbounded-copying output language such as $\{ww \mid w \in \{\mathrm{a}, \mathrm{b}\}^*\}$ to be specified, but in practice, Hulden's implementation restricts attention to cases where the equal sub-strings are bounded in length (p.125).

2-way Deterministic Finite-state Transducers 2.3.2

A finite-state method that computes unbounded copying elegantly and adequately is *2-way deterministic finite-state transducers* (2-way D-FSTs) (Dolatian and Heinz 2018a,b, 2019, 2020), which differ from conventional (1-way) FSTs in being able to move back and forth on the input.[8] 2-way D-FSTs have been proven to describe string transductions that are MSO-definable (Monadic Second-Order logic; Engelfriet and Hoogeboom 1999) and are equivalent to *streaming string transducers* (Alur and Černý 2010). In these formalisms, reduplication is modeled as a string-to-string *mapping* ($w \mapsto ww$). To avoid the mirror image function ($w \mapsto ww^R$), Dolatian and Heinz (2020) further studied sub-

classes of 2-way D-FSTs which cannot output anything during right-to-left passes over the input (cf. *rotating transducers*: Baschenis *et al.* 2017).

The issue addressed in Dolatian and Heinz (2020) is distinct from, but related to, the main concern of this paper: these transducers model reduplication as a function mapping underlying forms to surface forms ($w \mapsto ww$), while this paper aims to characterize only the identical-substrings requirement on the corresponding surface forms (ww). There are at least two reasons to address the string-set problem itself rather than considering only mappings between underlying and surface forms.

The first reason is a practical/strategic one, related to the problem of morphological *analysis* (rather than generation): the question of what kinds of transducers can implement the $ww \mapsto w$ mapping required for morphological analysis remains open, since 2-way D-FSTs (unlike standard 1-way FSTs) are not readily invertible as a class (Dolatian and Heinz 2020, p.235). Although we do not directly address the morphological analysis problem here, recognizing the reduplicated ww strings is plausibly an important first step: applying the mapping $ww \mapsto w$ to some string x requires at least *recognizing* whether x belongs to the ww string set.

The second reason stems from a full consideration of the linguistic facts surrounding reduplication: there is evidence supporting meaning-free, non-morphologically-generated reduplication-like structures, as mentioned in the discussion of aggressive reduplication above. This suggests that the phonological grammar involves a *phonotactic* constraint requiring sub-string identity, and the natural formal model for such a constraint is an automaton that generates/accepts the strings satisfying it. A constraint of this sort could play a role in mappings relating underlying forms to surface forms, so we may be missing a generalization if we only model those mappings directly.

[8] 2-way FSTs are still more restricted than Turing machines since they cannot move back and forth on the output tape, only the input tape.

FINITE-STATE BUFFERED MACHINES 3

The aim of proposing a new computing device is to add reduplication to FSAs and thereby gain a better understanding of the required computational operations. The new formalism is *finite-state buffered machines* (FSBMs), a summary of which is provided in Section 3.1. For each of exposition, we introduce the new formalism by first presenting the general case of FSBMs in Section 3.2, along with illustrative examples. A clearer understanding of the formalisms' capacity for copying comes from identifying a subset of FSBMs that we call *complete-path FSBMs*, in Section 3.3; we show that the languages recognized by FSBMs are precisely the languages recognized by complete-path FSBMs in Section 3.4.

FSBM in a nutshell 3.1

FSBMs are two-taped automata with finite-state core control.[9] One tape stores the input, as in normal FSAs; the other serves as an unbounded memory buffer, storing reduplicants temporarily for future string matching. An FSBM can be thought of as an extension to the FSRAs discussed above (Cohen-Sygal and Wintner 2006) but equipped with unbounded memory. FSBMs with a *bounded* buffer would be as expressive as FSRAs, and therefore also standard FSAs.

The interaction of the queue-like buffer with the input is restricted in two important ways. First, the buffer stores symbols from the same alphabet as the input, unlike the stack in a PDA, for example. Second, once one symbol is removed from the buffer, everything else must also be emptied from the buffer before symbols can next be added to it. These restrictions together ensure the machine will not generate string reversals or other non-reduplicative non-regular patterns.

Unlike a standard FSA, an FSBM works with two possible modes: in *normal* (N) mode, M reads symbols and transits between states, functioning as a normal FSA; and in *buffering* (B) mode, besides consuming symbols from the input and taking transitions among states, M

[9] The presented model here is a modified version of the proposal of Wang (2021a) and Wang (2021b).

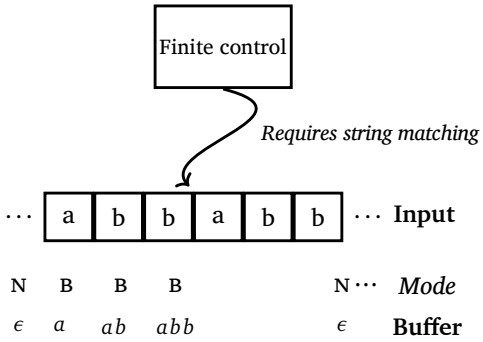


Figure 6: *Mode changes and input-buffer interaction of an FSBM M on "…abbabb…". The machine switches to* B *mode to temporarily store symbols in the queue-like buffer, and then at the point indicated by the arrow it compares the buffer contents against the remaining input. If the two strings match, the buffer is emptied, the matched input sub-string is consumed and the machine switches to* N *mode*

.

adds a copy of just-read symbols to the queue-like buffer. At a specific point, M exits buffering (B) mode, matching the stored string in the buffer against (a portion of) the remaining input. Provided this match succeeds, it switches back to normal (N) mode for another round of computation. Figure 6 provides a schematic diagram showing how the mode of an FSBM alternates when it determines the equality of substrings and how the buffer interacts with the input.

As presented here, FSBMs can only compute local reduplication with two adjacent, completely identical copies. They cannot handle non-local reduplication, multiple reduplication, or non-identical copies. We believe the current machinery can serve as the foundation for proposing different variants, and we discuss some potential modifications along these lines in Section 6.1.

Having introduced the important intuitions, we now turn to the formal definition of FSBMs.

Preliminaries & Definitions 3.2

For any finite alphabet Σ of symbols, we use Σ^* to denote the set of all finite strings over Σ. For a string w, $|w|$ denotes its length. ϵ is the null string and thus $|\epsilon| = 0$. We denote string union by '+', and denote string concatenation by simple juxtaposition, assuming implicit conversion between symbols and length-one strings where necessary. If $u = vw$, then $v\backslash u = w$; otherwise, $v\backslash u$ is undefined.

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

- Σ*: a finite set of symbols*
- 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 mode*
- $H \subseteq Q - G$*: states requiring string matching*
- δ*:* $Q \times (\Sigma \cup \{\epsilon\}) \times Q$*: transition relation*

The specification of the two sets of special states, G and H, serves to control what portions of a string are copied. To avoid intricacies, G and H are defined to be disjoint. The special case where $G = H = \emptyset$ corresponds to a standard FSA.

Definition 2. *A **configuration** of an FSBM is a four-tuple $(u, q, v, t) \in \Sigma^* \times Q \times \Sigma^* \times \{\text{N}, \text{B}\}$, where u is the input string; q is the current state; v is the string in the buffer; and t is the machine's current mode.*

Definition 3. *Given an FSBM $M = (\Sigma, Q, I, F, G, H, \delta)$, the relation $\vdash_M$ on configurations is the smallest relation such that, for any $u, v, w \in \Sigma^*$:*

- *For every transition $(q_1, x, q_2) \in \delta$*
  $(xu,\ q_1,\ \epsilon,\ \text{N}) \vdash_M (u,\ q_2,\ \epsilon,\ \text{N})$ if $q_1 \notin G$ and $q_2 \notin H$ $\vdash_{\text{N}}$
  $(xu,\ q_1,\ v,\ \text{B}) \vdash_M (u,\ q_2,\ vx,\ \text{B})$ if $q_1 \notin H$ and $q_2 \notin G$ $\vdash_{\text{B}}$
- *For every $q \in G$*
  $(u,\ q,\ \epsilon,\ \text{N}) \vdash_M (u,\ q,\ \epsilon,\ \text{B})$ $\vdash_{\text{N}\to\text{B}}$
- *For every $q \in H$*
  $(vw,\ q,\ v,\ \text{B}) \vdash_M (w,\ q,\ \epsilon,\ \text{N})$ $\vdash_{\text{B}\to\text{N}}$

Thus, $\vdash_M = \vdash_{\text{N}} \cup \vdash_{\text{B}} \cup \vdash_{\text{N}\to\text{B}} \cup \vdash_{\text{B}\to\text{N}}$. *When* $D_1 \vdash_M D_2$, *we say* D_1 ***yields*** D_2.

As is standard, $\vdash^*$ denotes the reflexive and transitive closure of $\vdash$, while $\vdash^+$ is the corresponding irreflexive closure.

Definition 4. *A* ***run*** *of* M *on* w *is a sequence of configurations* $D_0, D_1, D_2 \dots D_m$ *such that*

- $\exists q_0 \in I$, $D_0 = (w, q_0, \epsilon, \text{N})$
- $\exists q_f \in F$, $D_m = (\epsilon, q_f, \epsilon, \text{N})$
- $\forall\, 0 \leq i < m$, $D_i \vdash_M D_{i+1}$

Definition 5. *The language recognized by* $M = \langle \Sigma, Q, I, F, G, H, \delta \rangle$, *denoted by* $L(M)$, *is the set of all strings* $w \in \Sigma^*$ *such that there is a run of* M *on* w. *That is,* $L(M) = \{w \in \Sigma^* \mid (w, q_0, \epsilon, \text{N}) \vdash_M^* (\epsilon, q_f, \epsilon, \text{N}), q_0 \in I, q_f \in F\}$.

Notice that we do not impose any notion of determinism on the transitions of an FSBM. We return to some discussion of this point in Section 6.2.

Now, we give examples of FSBMs. In all illustrations, G states are drawn with diamonds and H states are drawn with squares.

3.2.1 Examples: Total reduplication

Figure 7 offers an FSBM M_1 for L_{ww}, with arbitrary strings over the alphabet $\Sigma = \{a, b\}$ as potential bases. The initial state q_1 is also a G state, and the only H state is q_3. The machine stores a copy of string computed in between q_1 and q_3 in the buffer and requires string matching at q_3. Since the states where the machine enters ($q_1 \in G$) and leaves ($q_3 \in H$) buffering mode are also the initial and final states respectively, this machine will recognize simple total reduplication. Table 3 gives a complete run of M_1 on the string *abbabb*. As we can see in Step 8, the string *abb* in the remaining input is consumed in one step.

For the rest of the illustration, we focus on the FSBM M_2 in Figure 8a. M_2 in Figure 8a recognizes the non-context-free language $\{a^i b^j a^i b^j \mid i, j \geq 1\}$. 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 8b). q_1 is an initial state and more importantly a G

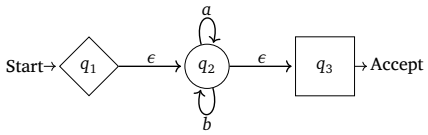
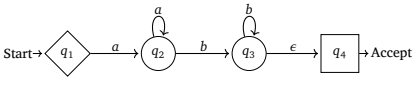


Figure 7: M_1 *with* $G = \{q_1\}$ *and* $H = \{q_3\}$. $L(M_1) = \{ww \mid w \in \{a, b\}^*\}$

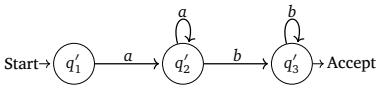
	Used arc or *state*	⊢ types	*Configuration* (input, state, buffer, mode)
1.	*N/A*		(*abbabb*, q_1, ϵ, N)
2.	$q_1 \in G$	$\vdash_{\text{N}\to\text{B}}$	(*abbabb*, q_1, ϵ, B)
3.	(q_1, ϵ, q_2)	$\vdash_{\text{B}}$	(*abbabb*, q_2, ϵ, B)
4.	(q_2, a, q_2)	$\vdash_{\text{B}}$	(*bbabb*, q_2, *a*, B)
5.	(q_2, b, q_2)	$\vdash_{\text{B}}$	(*babb*, q_2, *ab*, B)
6.	(q_2, b, q_2)	$\vdash_{\text{B}}$	(*abb*, q_2, *abb*, B)
7.	(q_2, ϵ, q_3)	$\vdash_{\text{B}}$	(*abb*, q_3, *abb*, B)
8.	$q_3 \in H$	$\vdash_{\text{B}\to\text{N}}$	(ϵ, q_3, ϵ, N)
	Accept		

Table 3: M_1 *in Figure 7 accepts* $abbabb$

state, forcing M_2 to enter B at the beginning of any run. Then M_2 in B mode always keeps a copy of consumed symbols until it proceeds to q_4, which is an H state and therefore requires M_2 to stop buffering and check for string identity to empty the buffer. Then, M_2 with a blank buffer can switch to N mode. It eventually ends at q_4, a legal final state. Table 4 shows one possible sequence of configurations of M_2 on *ababb*; this string is rejected because there is no way to reach a valid ending configuration.



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



(b) *An FSA M_0; $L(M_0) = \{a^i b^j \mid i, j \geq 1\}$*

Figure 8: *One example FSBM and the corresponding FSA for the base language*

	Used Arc or *state*	⊢ types	*Configuration* (input, state, buffer, mode)
1.	*N/A*		(*ababb*, q_1, ϵ, N)
2.	$q_1 \in G$	$\vdash_{\text{N}\to\text{B}}$	(*ababb*, q_1, ϵ, B)
3.	(q_1, a, q_2)	$\vdash_{\text{B}}$	(*babb*, q_2, *a*, B)
4.	(q_2, b, q_3)	$\vdash_{\text{B}}$	(*abb*, q_3, *ab*, B)
5.	(q_3, ϵ, q_4)	$\vdash_{\text{B}}$	(*abb*, q_4, *ab*, B)
6.	$q_4 \in H$	$\vdash_{\text{B}\to\text{N}}$	(*b*, q_4, ϵ, N)
	Reject		

Table 4: *M_2 in Figure 8a rejects $ababb$*

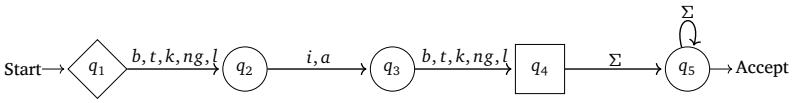


Figure 9: *An FSBM M_3 for Agta CVC-reduplicated plurals: $G = \{q_1\}$ and $H = \{q_4\}$*

3.2.2 Examples: Partial reduplication

Assuming $\Sigma = \{b, t, k, ng, l, i, a\}$, the FSBM M_3 in Figure 9 serves as a simple model of Agta CVC reduplicated plurals, as illustrated earlier 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 a consonant from the input 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

	Used Arc	$\vdash$ types	*Configuration*
1.	*N/A*		(*taktakki*, q_1, ϵ, N)
2.	$q_1 \in G$	$\vdash_{\text{N}\to\text{B}}$	(*taktakki*, q_1, ϵ, B)
3.	(q_1, t, q_2)	$\vdash_{\text{B}}$	(*aktakki*, q_2, *t*, B)
4.	(q_2, a, q_3)	$\vdash_{\text{B}}$	(*ktakki*, q_3, *ta*, B)
5.	(q_3, k, q_4)	$\vdash_{\text{N}\to\text{B}}$	(*takki*, q_4, *tak*, B)
6.	$q_4 \in H$	$\vdash_{\text{B}\to\text{N}}$	(ki, q_4, ϵ, N)
7.	(q_4, k, q_5)	$\vdash_{\text{N}}$	(i, q_5, ϵ, N)
8.	(q_5, i, q_5)	$\vdash_{\text{N}}$	(ϵ, q_5, ϵ, N)
		Accept	

Table 5: M_3 *in Figure 9 accepts* $taktakki$

	Used Arc	$\vdash$ types	*Configuration*
1.	*N/A*		(*tiktakki*, q_1, ϵ, N)
2.	$q_1 \in G$	$\vdash_{\text{N}\to\text{B}}$	(*tiktakki*, q_1, ϵ, B)
3.	(q_1, t, q_2)	$\vdash_{\text{B}}$	(*iktakki*, q_2, *t*, B)
4.	(q_2, i, q_3)	$\vdash_{\text{B}}$	(*ktakki*, q_3, *ti*, B)
5.	(q_3, k, q_4)	$\vdash_{\text{B}}$	(*takki*, q_4, *tik*, B)
	$q_4 \in H$*: checks for string identity and rejects*		

Table 6: M_3 *in Figure 9 rejects* $tiktakki$

state, requires M_3 to match this CVC sequence in the buffer with the remaining input. Then, M_3 with a blank buffer can switch to N mode at q_4. It transitions to q_5 to process the rest of the input via the normal loops on q_5. A successful run should end at q_5, the only final state. Table 5 gives a complete run of M_3 on the string "*taktakki*". Table 6 illustrates a case where the crucial step of returning from B mode to N mode is not possible, because of the non-matching sub-strings in "*tiktakki*"; this string is rejected by M_3.

3.3 *The copying mechanism and complete-path FSBMs*

The copying mechanism is realized by four essential components: 1) the unbounded memory buffer, which has queue-like storage; 2) added modalities; 3) added specifications of states requiring the machine to

buffer symbols into memory, namely states in G; 4) added specifications of states requiring the machine to empty the buffer by matching sub-strings, namely states in H.

As shown in the definitions of configuration changes and the examples in Section 3.2, the machine must end in N mode to accept 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 → N mode changes. Correspondingly, the resulting languages reflect either no copying (functioning as plain FSAs) or full copying.

In any specific run, it is the states that inform a machine 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 match strings. Hence, it is clear that 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 state later. Then the buffer has to be emptied for N mode at the point when a H state transits to a plain state. A template for those machines performing full copying can be seen in Figure 10.

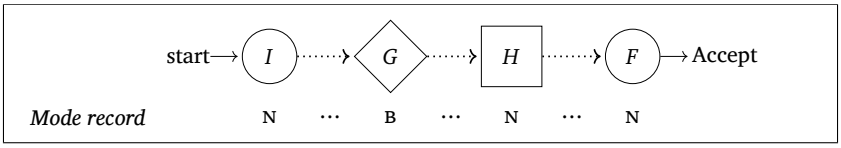


Figure 10: *The template for the implementation of the copying in FSBMs. Key components: G state, H states, and strict ordering between G and H. Dotted lines represent a sequence of transitions*

To allow us to reason about only the "useful" arrangements of G and H states, we impose an ordering requirement on G and H states in a machine. We define the *completeness restriction* on a path in Definition 7. We then identify those FSBMs in which all paths are complete as *complete-path FSBMs*. The machine M_1 in Figure 7, M_2 in Figure 8a and M_3 in Figure 9 are all complete-path FSBMs.

Definition 6. *A* ***path*** *from one state p_1 to another state p_n in an FSBM M is a sequence of states $p_1, p_2, p_3, \ldots p_n$ such that for each $i \in \{1, \ldots, n-1\}$, there is a transition $(p_i, x, p_{i+1}) \in \delta_M$.*

Definition 7. *A path in an FSBM M is **complete** if it is in the denotation of the regular expression $(P^*GP^*HP^* + P^*)^*$, where P represents any state in $Q - (G \cup H)$. A **complete-path FSBM** is an FSBM in which any path $p_1 \dots p_n$ with $p_1 \in I$ and $p_n \in F$ is complete.*

Definition 8. *A path is said to be **a copying path** if it is complete and there is at least one G state (or at least one H state).*

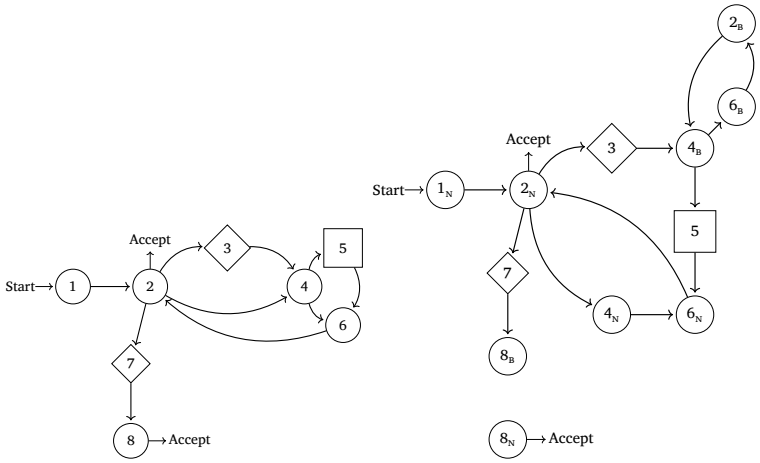
3.4 *The sufficiency of complete-path FSBMs*

Now, we show that the languages recognized by FSBMs are precisely the languages recognized by complete-path FSBMs; this will allow us to restrict attention to complete-path FSBMs when studying the formal properties of these machines below.

Proposition 1. *For any FSBM M, there exists a complete-path M' with $L(M) = L(M')$.*

Incomplete paths contribute nothing to the language generated by an FSBM, so showing this equivalence requires showing that, for any FSBM M_1, we can construct a new FSBM M_2 such that every path from an initial state to an accepting state in M_2 corresponds to some complete path from an initial state to an accepting state in M_1. The idea is that M_2 is a complete-path FSBM that keeps only those paths from M_1 that are indeed complete.

The non-obvious cases of this construction involve scenarios where some plain state in M_1 might be reached either in normal (N) mode or in buffering (B) mode, depending on the path by which that plain state is reached. In Figure 11a, for example, this is the case for states 2, 4 and 6: intuitively, a path from state 2 back to itself might contain a G state (3) or an H state (5), or both or neither. To construct an equivalent complete-path FSBM M_2, we "split" each plain state q into two distinct states q_{N} and q_{B}. Transitions from a G state to q and transitions from q to an H state (i.e. transitions that only make sense in buffering mode) are carried over in M_2 for q_{B} but not for q_{N}. Similarly, transitions from an H state to q and transitions from q to a G state are carried over in M_2 for q_{N} but not for q_{B}. And the status of q as an initial and/or accepting state is carried over for q_{N} but not for q_{B}. Figure 11b shows the resulting complete-path FSBM for this



(a) M_1 **(b)** M_2

Figure 11: *Construction of a complete-path FSBM M_2 that is equivalent to M_1.*

example. In addition to keeping track of the mode in which states 2, 4 and 6 are visited, notice that this construction also prevents state 7 from occurring in any path from an initial state to an accepting state, since 8_B is not an accepting state and 8_N is unreachable.

4 PUMPING LEMMA

We define the *Regular Copying Languages* (RCLs) to be the set of all languages accepted by some (complete-path) FSBM. To be able to prove that some languages are not RCLs, we present a pumping lemma in this section. The idea that is if an FSBM produces a string $urrv$ via a copying run, and r is sufficiently long, then some subpart of r will be pumpable in the manner of the familiar pumping lemma for regular languages; that is, r can be broken into $x_1x_2x_3$ such that $ux_1x_2^ix_3x_1x_2^ix_3w$ is also accepted.[10]

[10] This idea is largely inspired by Savitch (1989, p.256), who proposes a pumping lemma for context-free languages augmented with copying.

Theorem 1. *If $\mathscr{L}$ is a regular copying language, there is a positive integer k such that for every string $w \in \mathscr{L}$ with $|w| \geq 4k$, one of the following two conditions holds:*

1. *w can be rewritten as $w = xyz$ with*
   (a) $|y| \geq 1$
   (b) $|xy| \leq k$
   (c) $\forall i \geq 0, xy^i z \in \mathscr{L}$
2. *w can be rewritten as $w = ux_1x_2x_3x_1x_2x_3v$ such that*
   (a) $|x_2| \geq 1$
   (b) $|x_1x_2| \leq k$
   (c) $\forall i \geq 0, ux_1x_2^ix_3x_1x_2^ix_3v \in \mathscr{L}$

Proof. Since $\mathscr{L}$ is a regular copying language, there is a complete-path FSBM M that recognizes $\mathscr{L}$. Let k be the number of states in M. For an arbitrary string $w \in \mathscr{L}$ with $|w| > 4k$, there is at least one path through M that generates w. Let p be the shortest such path (or if there are ties, choose arbitrarily). Note that p does not contain any ϵ-*loops*; if it did, its length would not be minimal among all candidate paths.

Suppose first that p is not a copying path. The length of p is at least $|w|+1$, and so since $|w| \geq 4k > k$, some state must occur twice in p, in fact in the first $k+1$ elements of p. As in the standard pumping lemma for regular languages, this means that w can be rewritten as xyz, with $|xy| \leq k$, in such a way that M can also generate xy^iz by repeating the loop, and $y \neq \epsilon$ since p contains no ϵ-loops. So in this case, w satisfies Condition 1.

If $p = p_0p_1 \ldots p_n$ is a copying path, then the run that generates $w = urrv$ must have the form $(urrv, p_0, \epsilon, \text{N}) \vdash_M^* (rrv, p_i, \epsilon, \text{N}) \vdash_M (rrv, p_i, \epsilon, \text{B}) \vdash_M^* (rv, p_j, r, \text{B}) \vdash_M (v, p_j, \epsilon, \text{N}) \vdash_M^* (\epsilon, p_n, \epsilon, \text{N})$ with $p_0 \in I$, $p_i \in G$, $p_j \in H$ and $p_n \in F$. Since $|w| \geq 4k$, at least one of $|u|, |r|, |v|$ is greater than or equal to k.

- If $|r| \geq k$, then $|p_i \ldots p_j| \geq |r|+1 \geq k+1$, so at least one state must appear twice in the first $k+1$ elements of the sequence $p_i \ldots p_j$, i.e. there are ℓ and ℓ' such that $i \leq \ell < \ell' \leq j$ and $p_\ell = p_{\ell'}$, with $\ell' - i < k$. Then it must be possible to rewrite r as $x_1x_2x_3$, with $|x_1x_2| \leq k$, such that repeating the subpath $p_\ell \ldots p_{\ell'}$ results

in pumping x_2, and so any string of the form $x_1 x_2^i x_3$ can be consumed from the input and stored in the buffer in the course of moving from $p_i \in G$ to $p_j \in H$, i.e. $(x_1 x_2^i x_3 x_1 x_2^i x_3 v, p_i, \epsilon, \text{B}) \vdash_M^* (x_1 x_2^i x_3 v, p_j, x_1 x_2^i x_3, \text{B}) \vdash_M (v, p_j, \epsilon, \text{N})$. M will therefore generate all strings of the form $u x_1 x_2^i x_3 x_1 x_2^i x_3 v$, satisfying Condition 2.

- If $|u| \geq k$, then $|p_0 \dots p_i| \geq |u| + 1 \geq k + 1$, so at least one state must appear twice in the sequence $p_0 \dots p_i$, i.e. there are ℓ and ℓ' such that $0 \leq \ell < \ell' \leq i$ and $p_\ell = p_{\ell'}$, with $\ell' < k$. There are two cases to consider:
  - Suppose that M is in buffering mode throughout the part of the run from p_ℓ to $p_{\ell'}$. Therefore $p_\ell = p_{\ell'}$ is a plain state. Then it must be possible to rewrite u as $u' x_1 x_2 x_3 x_1 x_2 x_3 v'$, such that repeating the subpath $p_\ell \dots p_{\ell'}$ results in pumping x_2. And since the repeated state must occur in the first $k+1$ elements of p, $|u' x_1 x_2| \leq k$ and therefore $|x_1 x_2| \leq k$. M will therefore generate all strings of the form $u' x_1 x_2^i x_3 x_1 x_2^i x_3 v' r r v$, satisfying Condition 2.
  - Otherwise, it must be possible to rewrite u as $x_1 x_2 x_3$ such that repeating this loop pumps x_2; since M is a complete-path FSBM, repeating the loop cannot create incomplete paths. And since the repeated state must occur in the first $k+1$ elements of p, $|x_1 x_2| \leq k$. M will therefore generate all strings of the form $x_1 x_2^i x_3 r r v$, satisfying Condition 1.
- If $|v| \geq k$, an analogous argument shows that either Condition 1 or Condition 2 is satisfied.

□

Theorem 2. *$\mathscr{L}_{inv} = \{(a+b)^i c^j (a+b)^i c^j \mid i, j \geq 0\}$ is not an RCL.*

Proof. Suppose $\mathscr{L}_{inv}$ is an RCL. Let $w = a^k c^{k+1} b^k c^{k+1} \in \mathscr{L}_{inv}$, where k is the pumping length from Theorem 1. Given $|w| > 4k$, one of the conditions from Theorem 1 must hold.

1. Assume condition 1 holds. That is $w = xyz$ such that (i) $|y| \geq 1$, (ii) $|xy| \leq k$ and (iii) $\forall i \geq 0, xy^i z \in L$. Given $|xy| \leq k$, y must only contain as. Therefore $xyyz$ must have the form $a^{k+|y|} c^{k+1} b^k c^{k+1}$, so $xyyz \notin \mathscr{L}_{inv}$, a contradiction.

2. Assume condition 2 holds. Then, $w = ux_1x_2x_3x_1x_2x_3v$ such that (i) $|x_2| > 1$, (ii) $|x_1x_2| \leq k$ and (iii) $\forall i \geq 0$, $ux_1x_2^ix_3x_1x_2^ix_3v \in \mathscr{L}_{inv}$. The string x_1x_2 cannot contain the sub-string ac, because x_1x_2 occurs twice in w but ac does not; similarly, x_1x_2 cannot contain cb or bc. There remain three possible ways of choosing x_1x_2 with $|x_1x_2| \leq k$, each incurring a contradiction.

   (a) If x_1x_2 contains only as, then x_3 must also contain only as because it occurs in between the two occurrences of x_1x_2 in w. Therefore $ux_1x_2^2x_3x_1x_2^2x_3v$ must have the form $a^\ell c^{k+1}b^kc^{k+1}$ with $\ell > k$, and is therefore not in $\mathscr{L}_{inv}$; a contradiction.

   (b) Similarly, if x_1x_2 contains only bs, then $ux_1x_2^2x_3x_1x_2^2x_3v$ must have the form $a^kc^{k+1}b^\ell c^{k+1}$ with $\ell > k$, and is therefore not in $\mathscr{L}_{inv}$; a contradiction.

   (c) Finally, suppose x_1x_2 contains only cs. If x_3 did not contain only cs, then it would need to cover the sub-string b^k since it appears in between the two occurrences of x_1x_2 in w; but if x_3 covered the sub-string b^k then this sub-string would occur twice in w, which it does not. So x_3 must also contain only cs. Therefore $ux_1x_2^2x_3x_1x_2^2x_3v$ must have the form either $a^kc^\ell b^kc^{k+1}$ or $a^kc^{k+1}b^kc^\ell$, with $\ell > k+1$; a contradiction.

□

Example 1. *Some Non-RCL languages*

1. $\mathscr{L}_{SwissGerman} = \{a^ib^jc^id^j \mid i, j \geq 0\}$
2. $\mathscr{L} = \{a^nb^n \mid n \geq 0\}$
3. $\mathscr{L} = \{ww^R \mid w \in \Sigma^*\}$
4. $\mathscr{L} = \{www \mid w \in \Sigma^*\}$
5. $\mathscr{L} = \{w^{(2^n)} \mid n \geq 0\}$

To see that $\{w^{(2^n)} \mid n \geq 0\}$ is not an RCL, notice that the pumping lemma above requires that a constant-sized increase in the length of a string in the language can produce another string also in the language, but $w^{(2^n)}$ does not have this "constant growth" property (Joshi 1985).

5 CLOSURE PROPERTIES

The class of regular copying languages is closed under the following operations: intersection with a finite-state language (Section 5.1), some regular operations (union, concatenation, Kleene star; Section 5.2) and homomorphism (Section 5.3). But it is not closed under intersection, nor complementation (Section 5.4). More interestingly, it is not closed under inverse homomorphism (Section 5.5). In this section, we present proofs of these results.

5.1 *Closure under intersection with regular languages*

In this subsection, we write $\mathbf{0}$ for the zero matrix and $\mathbf{I}$ for the identity matrix, with the size of these matrices determined implicitly by context.

For any FSA $M = \langle Q, \Sigma, I, F, \delta \rangle$ and any symbol $x \in \Sigma$, $\mathbf{A}_x^M \in \{0,1\}^{|Q|\times|Q|}$ is the square matrix with rows and columns indexed by Q, whose (q_1, q_2) entry is 1 if $(q_1, x, q_2) \in \delta$ and is 0 otherwise. We will sometimes just write $\mathbf{A}_x$ where the FSA is clear from the context. We define $\mathbf{A}_\epsilon^M = \mathbf{I}$, and for any non-empty string $w = x_1 \dots x_n$ we define $\mathbf{A}_w^M = \mathbf{A}_{x_1}^M \dots \mathbf{A}_{x_n}^M$. Then it follows that the (q_1, q_2) entry of the matrix $\mathbf{A}_w^M$ is 1 if there is a path from q_1 to q_2 generating w, and is 0 otherwise.

We will assume, when we write any $\mathbf{A}_w^M$ in what follows, that the FSA M is supplemented with "sink states" as necessary to ensure that, for every $q_1 \in Q$ and every $x \in \Sigma$, there is at least one $q_2 \in Q$ such that $(q_1, x, q_2) \in \delta$. This ensures that, for any $w \in \Sigma^*$, there is at least one 1 on each row of $\mathbf{A}_w^M$, and therefore $\mathbf{A}_w^M \neq \mathbf{0}$.

We first define the relevant construction, then show below that it generates the desired intersection language. Without loss of generality, we assume that the FSA being intersected with the FSBM is ϵ-free.

Definition 9. *Given an FSBM $M_1 = \langle Q_1, \Sigma, I_1, F_1, G_1, H_1, \delta_1 \rangle$, and an FSA $M_2 = \langle Q_2, \Sigma, I_2, F_2, \delta_2 \rangle$, we define $M_1 \cap M_2$ to be the FSBM $\langle Q, \Sigma, I, F, G, H, \delta \rangle$, where*

- $Q = Q_1 \times Q_2 \times \{0,1\}^{|Q_2|\times|Q_2|}$
- $I = I_1 \times I_2 \times \{\boldsymbol{0}\}$

- $F = F_1 \times F_2 \times \{\boldsymbol{0}\}$
- $G = G_1 \times Q_2 \times \{\boldsymbol{A}_\epsilon^{M_2}\}$
- $H = H_1 \times Q_2 \times \{\boldsymbol{0}\}$
- $\delta = \delta_{\text{N}} \cup \delta_{\text{B}} \cup \delta_{\text{N}\to\text{B}} \cup \delta_{\text{B}\to\text{N}}$, *where*
  - *(a)* $((q_1, q_1', \boldsymbol{0}), x, (q_2, q_2', \boldsymbol{0})) \in \delta_{\text{N}}$ *iff* $(q_1, x, q_2) \in \delta_1$ *with* $q_1 \notin G_1$ *and* $q_2 \notin H_1$*, and either*
    - $(q_1', x, q_2') \in \delta_2$*, or*
    - $x = \epsilon$ *and* $q_1' = q_2'$*.*
  - *(b)* $((q_1, q_1', \boldsymbol{0}), \epsilon, (q_1, q_1', \boldsymbol{A}_\epsilon^{M_2})) \in \delta_{\text{N}\to\text{B}}$ *iff* $q_1 \in G_1$
  - *(c)* $((q_1, q_1', \boldsymbol{A}), x, (q_2, q_2', \boldsymbol{A}\boldsymbol{A}_x^{M_2})) \in \delta_{\text{B}}$ *iff* $\boldsymbol{A} \neq \boldsymbol{0}$ *and* $(q_1, x, q_2) \in \delta_1$ *with* $q_1 \notin H_1$ *and* $q_2 \notin G_1$*, and either*
    - $(q_1', x, q_2') \in \delta_2$*, or*
    - $x = \epsilon$ *and* $q_1' = q_2'$*.*
  - *(d)* $((q_1, q_1', \boldsymbol{A}), \epsilon, (q_1, q_2', \boldsymbol{0})) \in \delta_{\text{B}\to\text{N}}$ *iff* $q_1 \in H_1$ *and* $\boldsymbol{A} \neq \boldsymbol{0}$ *and the* (q_1', q_2') *entry of* $\boldsymbol{A}$ *is 1*

Notice that $|Q| = |Q_1| \times |Q_2| \times 2^{|Q_1|\times|Q_2|}$ is finite, since Q_1 and Q_2 are both finite.

The central challenge in setting up an FSBM to simulate the combination of an FSBM M_1 and an FSA M_2 is handling the effect on M_2 of $\vdash_{\text{B}\to\text{N}}$ transitions in M_1, where a string of arbitrary length is emptied from the buffer. Obviously the buffered string itself cannot be stored in the simulating FSBM's finite state. But, following an idea from Savitch (1989), any buffered string w determines a finite transition relation on the states of M_2, and it suffices to record this relation, which we encode in the form of the matrix $\mathbf{A}_w^{M_2}$.

The following lemma establishes the invariants that underpin the proof that this construction recognizes $L(M_1) \cap L(M_2)$.

Lemma 1. *Suppose a non-empty sequence of configurations* $D_1, \ldots D_m$ *is the initial portion of a successful run (of any string) on an intersection FSBM* $M = M_1 \cap M_2$*, with each* $D_i = (u_i, (q_i, q_i', \boldsymbol{A}_i), v_i, t_i)$*. Then one of the following is true:*

(i) $t_i = \text{N}$ *and* $\boldsymbol{A}_i = \boldsymbol{0}$

(ii) $t_i = \text{N}$ *and* $(q_i, q_i', \boldsymbol{A}_i) \in (G_1 \times Q_2 \times \{\boldsymbol{A}_\epsilon^{M_2}\}) = G$

(iii) $t_i = \text{B}$ *and* $\boldsymbol{A}_i = \boldsymbol{A}_{v_i}^{M_2}$

(iv) $t_i = \text{B}$ *and* $(q_i, q_i', \boldsymbol{A}_i) \in (H_1 \times Q_2 \times \{\boldsymbol{0}\}) = H$

Proof. By induction on the length m of the sequence. If $m = 1$, then $t_m = \text{N}$ and $(q_m, q'_m, \mathbf{A}_m) \in I = I_1 \times I_2 \times \{\mathbf{0}\}$, so $\mathbf{A}_m = \mathbf{0}$, satisfying (i). Now we consider a sequence $D_1 \dots D_m D_{m+1}$ where we assume that the requirement holds of D_m. Since $D_m \vdash_{M_1 \cap M_2} D_{m+1}$, there are four cases to consider.

- Suppose $D_m \vdash_{\text{N}} D_{m+1}$. Then $t_m = t_{m+1} = \text{N}$, $(q_m, q'_m, \mathbf{A}_m) \notin G$, and $(q_{m+1}, q'_{m+1}, \mathbf{A}_{m+1}) \notin H$. The inductive hypothesis therefore implies that $\mathbf{A}_m = \mathbf{0}$. Now there are four subcases, depending on the critical element of δ that licenses $D_m \vdash_{\text{N}} D_{m+1}$.
  - If the critical transition is in δ_{N}, then immediately $\mathbf{A}_{m+1} = \mathbf{0}$, satisfying (i).
  - If the critical transition is in $\delta_{\text{N}\to\text{B}}$, then $q_{m+1} \in G_1$ and $\mathbf{A}_{m+1} = \mathbf{A}_\epsilon^{M_2}$, satisfying (ii).
  - The critical transition cannot be in δ_{B}, since $\mathbf{A}_m = \mathbf{0}$.
  - The critical transition cannot be in $\delta_{\text{B}\to\text{N}}$, since $(q_{m+1}, q'_{m+1}, \mathbf{A}_{m+1}) \notin H$ which implies that either $q_{m+1} \notin H_1$ or $\mathbf{A}_{m+1} \neq \mathbf{0}$.
- Suppose $D_m \vdash_{\text{N}\to\text{B}} D_{m+1}$. Then $t_m = \text{N}$, $t_{m+1} = \text{B}$, $v_m = v_{m+1} = \epsilon$, and $(q_m, q'_m, \mathbf{A}_m) = (q_{m+1}, q'_{m+1}, \mathbf{A}_{m+1}) \in G = G_1 \times Q_2 \times \{\mathbf{A}_\epsilon^{M_2}\}$. Therefore $\mathbf{A}_{m+1} = \mathbf{A}_\epsilon^{M_2} = \mathbf{A}_{v_{m+1}}^{M_2}$, satisfying (iii).
- Suppose $D_m \vdash_{\text{B}} D_{m+1}$. Then $t_m = t_{m+1} = \text{B}$, $(q_m, q'_m, \mathbf{A}_m) \notin H$, $(q_{m+1}, q'_{m+1}, \mathbf{A}_{m+1}) \notin G$, and $v_{m+1} = v_m x$ for some $x \in \Sigma \cup \{\epsilon\}$. The inductive hypothesis therefore implies that $\mathbf{A}_m = \mathbf{A}_{v_m}^{M_2}$. Now there are four subcases, depending on the critical element of δ that licenses $D_m \vdash_{\text{B}} D_{m+1}$.
  - The critical transition cannot be in δ_{N}, since $\mathbf{A}_m = \mathbf{A}_{v_m}^{M_2} \neq \mathbf{0}$.
  - The critical transition cannot be in $\delta_{\text{N}\to\text{B}}$, since $(q_{m+1}, q'_{m+1}, \mathbf{A}_{m+1}) \notin G$ which implies that either $q_{m+1} \notin G_1$ or $\mathbf{A}_{m+1} \neq \mathbf{A}_\epsilon^{M_2}$.
  - If the critical transition is in δ_{B}, then $\mathbf{A}_{m+1} = \mathbf{A}_m \mathbf{A}_x^{M_2} = \mathbf{A}_{v_m}^{M_2} \mathbf{A}_x^{M_2} = \mathbf{A}_{v_m x}^{M_2} = \mathbf{A}_{v_{m+1}}^{M_2}$, satisfying (iii).
  - If the critical transition is in $\delta_{\text{B}\to\text{N}}$, then $q_{m+1} \in H_1$ and $\mathbf{A}_{m+1} = \mathbf{0}$, satisfying (iv).
- Suppose $D_m \vdash_{\text{B}\to\text{N}} D_{m+1}$. Then $t_m = \text{B}$, $t_{m+1} = \text{N}$, $v_{m+1} = \epsilon$, and $(q_m, q'_m, \mathbf{A}_m) = (q_{m+1}, q'_{m+1}, \mathbf{A}_{m+1}) \in H = H_1 \times Q_2 \times \{\mathbf{0}\}$. Therefore $\mathbf{A}_{m+1} = \mathbf{0}$, satisfying (i).

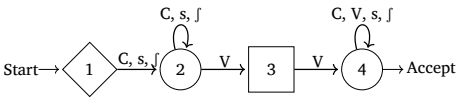
□

This lemma establishes that the matrix component of the constructed machine's state tracks the information necessary to determine the appropriate "jump" to make through M_2 when a string is emptied from the buffer: in a $\delta_{\text{B}\to\text{N}}$ transition from $(q_1, q_1', \mathbf{A})$ to $(q_1, q_2', \mathbf{0})$, the base FSBM M_1 is in state $q_1 \in H_1$ and therefore leaves buffering mode, and the matrix $\mathbf{A}$ determines the appropriate states q_2' for M_2 to jump to. The rest of the proof that $L(M_1 \cap M_2) = L(M_1) \cap L(M_2)$ is standard, but is provided in Appendix A.

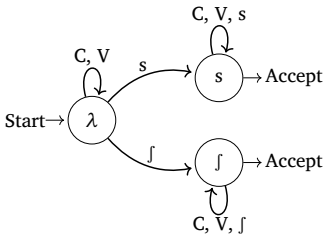
An example demonstrating how the intersection works can be found in Figure 12. The FSBM in Figure 12a computes the language that shows initial CC*V-copying. The FSA in Figure 12b, adapted from Heinz (2007, p.38), encodes Navajo sibilant harmony (Sapir and Hoijer 1967) on [anterior] features by banning *s…ʃ and *ʃ…s sequences. The intersection FSBM is shown in Figure 12c, which recognizes the language of strings obeying both restrictions.

That FSBM-recognizable languages are closed under intersection with regular languages is an important step in clarifying the potential role of FSBMs for phonological theory. The overwhelming majority of phonotactic constraints that are not concerned with sub-string identity are regular (Heinz 2018), and so any such constraint can be combined with an FSBM-enforcable identity constraint to yield another FSBM-recognizable language. In fact, since the regular languages are closed under intersection, FSBMs can also express the intersection of any *collection* of "normal" phonotactic constraints with any single FSBM-enforcable substring-identity constraint.

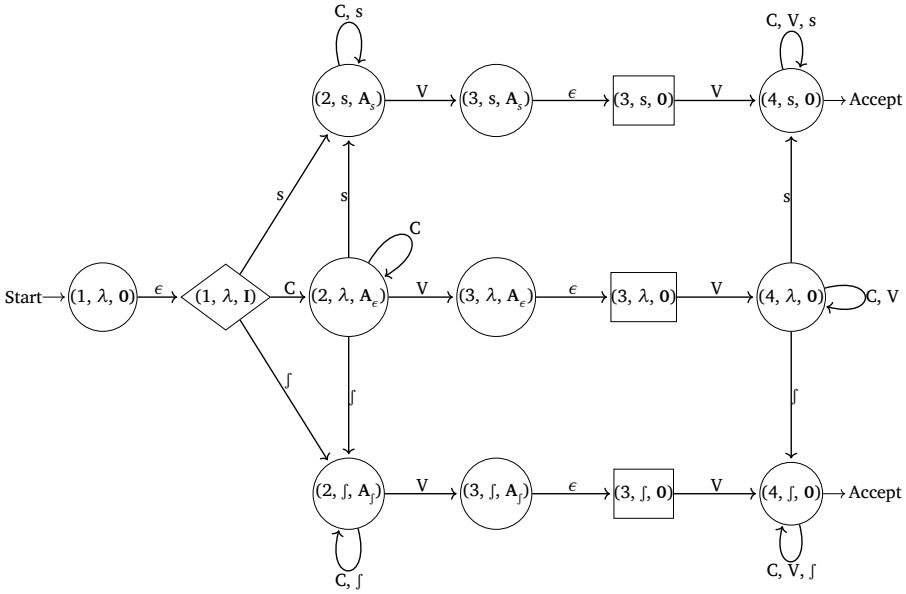
An important issue that we leave open for future work is developing an algorithm for intersecting an FSA with an FSBM that assigns *weights* to strings expressing degrees of well-formedness. This kind of intersection algorithm has been used to implement the notion of competition between candidates from Optimality Theory (Smolensky and Prince 1993), where violable constraints are expressed by weighted FSAs (Ellison 1994; Eisner 1997; Albro 1998; Riggle 2004b). Such an intersection algorithm for weighted FSBMs would allow for FSBM-defined reduplication constraints to be incorporated into implemented OT grammars. In other words, the point from the preceding paragraph might generalize beyond the special case of binary constraints which combine via simple intersection.



(a) *A complete-path FSBM M_1 recognizing initial CC*V-identity. $G = \{1\}$, $H = \{3\}$*



(b) *An FSA M_2 enforcing sibilant harmony. C indicates any non-sibilant consonant.*



(c) *The intersection FSBM $M_1 \cap M_2$, ignoring states from which no accepting state is reachable. $\boldsymbol{A}_\epsilon$ is the M_2 transition matrix for any string without any s or ʃ (equal to I); $\boldsymbol{A}_s$ is the transition matrix for all strings with at least one s and no ʃ; and $\boldsymbol{A}_ʃ$ is the transition matrix for all strings with at least one ʃ and no s*

Figure 12: *An example intersection construction*

5.2 *Closed under regular operations*

Noticeably, given complete-path FSBMs are finite-state machines with a copying mechanism, most of the proof ideas in this subsection are

similar to the standard proofs for FSAs, which can be found in Hopcroft and Ullman (1979) and Sipser (2013).

Theorem 3. *If L_1, L_2 are two FSBM-recognizable languages, then $L_1 \cup L_2$, $L_1 \circ L_2$ and L_1^* are also complete-path FSBM-recognizable languages.*

Proof. Assume there are complete-path FSBMs $M_1 = \langle \Sigma, Q_1, I_1, F_1, G_1, H_1, \delta_1 \rangle$ and $M_2 = \langle \Sigma, Q_2, I_2, F_2, G_2, H_2, \delta_2 \rangle$ such that $L(M_1) = L_1$ and $L(M_2) = L_2$, then …

Union One can construct a new FSBM M that accepts an input w if either M_1 or M_2 accepts w. $M = \langle \Sigma, Q, I, F, G, H, \delta \rangle$ such that

- $Q = Q_1 \cup Q_2 \cup \{q_0\}$
- $I = \{q_0\}$
- $F = F_1 \cup F_2$
- $G = G_1 \cup G_2$
- $H = H_1 \cup H_2$
- $\delta = \delta_1 \cup \delta_2 \cup \{(q_0, \epsilon, q') \mid q' \in (I_1 \cup I_2)\}$

The construction of M keeps M_1 and M_2 unchanged, but adds a new state q_0. q_0 is the only initial state, branching into those previous initial states in M_1 and M_2 with ϵ-arcs. q_0 is a non-G, non-H plain state, so the constructed automaton is a complete-path FSBM.

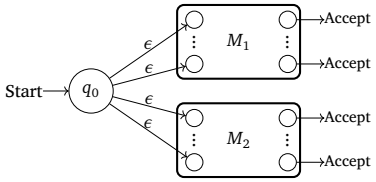


Figure 13: *The construction used in the union of two FSBMs*

Concatenation There is a complete-path FSBM M that can recognize $L_1 \circ L_2$ by the normal concatenation of two automata. The new machine $M = \langle \Sigma, Q, I, F, G, H, \delta \rangle$ satisfies $L(M) = L_1 \circ L_2$.

- Q $= Q_1 \cup Q_2 \cup \{q_0\}$
- I $= \{q_0\}$
- F $= F_2$
- $G = G_1 \cup G_2$

- $H = H_1 \cup H_2$
- $\delta = \delta_1 \cup \delta_2 \cup \{(p_f, \epsilon, q_i) \,|\, p_f \in F_1, q_i \in I_2\} \cup \{(q_0, \epsilon, p_i) \,|\, p_i \in I_1\}$

The new machine adds a new plain state q_0 and makes it the only initial state, branching into those previous initial states in M_1 ϵ-arcs. q_0 is not in H, nor in G. All final states in M_2 are the only final states in M. M also adds ϵ-arcs from all old final states in M_1 to all initial states in M_2.

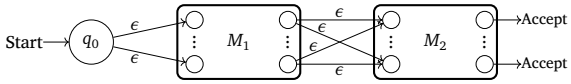


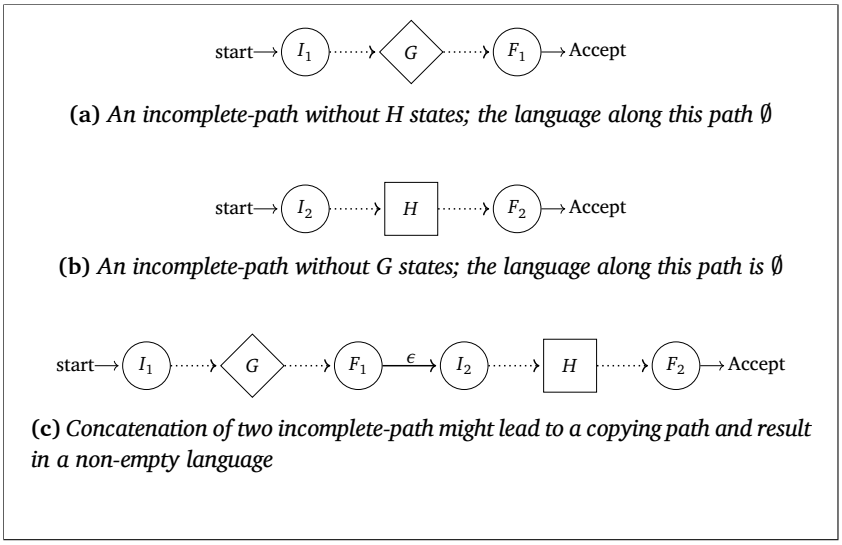
Figure 14: *The construction used in the concatenation of two FSBMs*

For this construction to work, it is important that we assume that M_1 and M_2 are complete-path FSBMs. Incomplete paths in two arbitrary machines might create a complete copying path, thus overgenerating under the construction of concatenation mentioned here. For example, as illustrated in Figure 15, imagine one path in M_1 only has G states but no H states, and another path in M_2 contains only H states. They both recognize the empty language $L_\emptyset = \emptyset$. Therefore, the concatenation of these two languages should also be $L_\emptyset$. The assumption that M_1 and M_2 are complete-path FSBMs ensures that the construction has this result.

Kleene Star $(L_1)^*$ is a complete-path FSBM-recognizable language. The new machine $M = \langle \Sigma, Q, I, F, G, H, \delta \rangle$ satisfies $L(M) = (L_1)^*$.

- $Q = Q_1 \cup \{q_0\}$
- $I = \{q_0\}$
- $F = F \cup \{q_0\}$
- $G = G_1$
- $H = H_1$
- $\delta = \delta_1 \cup \{(p_f, \epsilon, q_i) \,|\, p_f \in F_1, q_i \in I_1\} \cup \{(q_0, \epsilon, q_i) \,|\, q_i \in I_1\}$

M is similar to M_1 with a new initial state q_0. q_0 is also a final state, branching into 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



(a) *An incomplete-path without H states; the language along this path* $\emptyset$

(b) *An incomplete-path without G states; the language along this path is* $\emptyset$

(c) *Concatenation of two incomplete-path might lead to a copying path and result in a non-empty language*

Figure 15: *Problems arise in the concatenation of two incomplete paths. Dotted lines represent a sequence of transitions*

make sure M can jump back to an initial state after it hits a final state, ϵ transitions from any final state to any old initial states are added. Since all paths in M_1 are complete, concatenations of these paths do not overgenerate. □

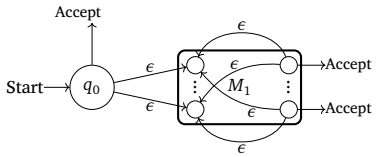


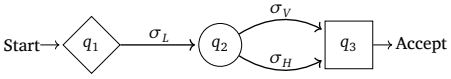
Figure 16: *The construction used in the star operation*

Closed under homomorphism 5.3

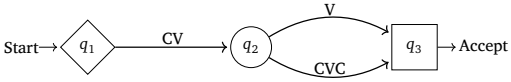
Theorem 4. *The class of languages recognized by complete-path FSBMs is closed under homomorphisms.*

Proof. That complete-path FSBM languages are closed under homomorphism can be proved by constructing a new machine M_h based on the base machine M, such that $L(M_h) = h(L(M))$. The construction goes as follows. Relabel each transition that emits x in M with the string $h(x)$, and add states to split the transitions so that there is only one symbol or ϵ on each arc in M_h. States added for this purpose are not included in G or H. All paths in M_h are complete since the construction does not affect the arrangements G and H states in paths. □

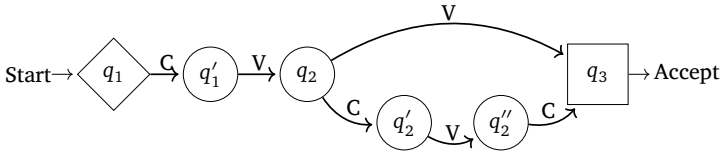
This construction is illustrated in Figure 17. The FSBM M uses the alphabet $\Sigma = \{\sigma_H, \sigma_L, \sigma_V\}$, and recognizes the finite language $\{\sigma_L\sigma_H\sigma_L\sigma_H, \sigma_L\sigma_V\sigma_L\sigma_V\}$. The constructed machine M_h recognizes the image of this finite language under the homomorphism $h : \Sigma^* \to \{C, V\}^*$ defined by $h(\sigma_L) = CV$, $h(\sigma_V) = V$, and $h(\sigma_H) = CVC$.



(a) $L(M) = \{\sigma_L\sigma_H\sigma_L\sigma_H, \sigma_L\sigma_V\sigma_L\sigma_V\}$



(b) $h(\sigma_L) = CV, h(\sigma_V) = V, h(\sigma_H) = CVC$. *The intermediate step when the arcs are relabeled with mapped strings*



(c) *States* q_1', q_2', q_2'' *are added to split the arcs.* $L(M_h) = \{CVVCVV, CVCVCCVCVC\}$

Figure 17: *Constructions used for the homomorphic language*

The fact that FSBMs are closed under homomorphism allows theorists to perform analyses at convenient levels of abstraction.

Not closed under intersection and complementation 5.4

Theorem 5. *The class of languages recognized by complete-path FSBMs is not closed under intersection, and thus not closed under complementation.*

Proof. $L_1 = \{wwx \mid w, x \in \{a, b\}^*\}$ and $L_2 = \{xww \mid w, x \in \{a, b\}^*\}$ are FSBM-recognizable languages. However, $L_1 \cap L_2 = \{www \mid w \in \{a, b\}^*\}$ is not an FSBM-recognizable language. Given FSBM is closed under union but is not closed under intersection, by De Morgan's law, FSBM is not closed under complementation. □

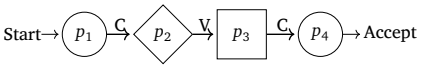
Not closed under inverse homomorphism 5.5

It is trivial to see that the class of languages recognized by complete-path FSBMs is closed under *one-to-one* alphabetic inverse homomorphism. One can directly relabel every mapped symbol in an FSBM to construct a new FSBM. But it is not closed under general inverse alphabetic homomorphisms and thus inverse homomorphism. Therefore, RCLs are not a trio.

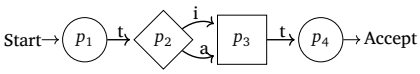
Consider the complete-path FSBM-recognizable language $L = \{a^i b^j a^i b^j \mid i, j \geq 1\}$ (see Figure 7a), and an alphabetic homomorphism $h : \{0, 1, 2\}^* \to \{a, b\}^*$ such that $h(0) = a$, $h(1) = a$ and $h(2) = b$. Then, the inverse homomorphic image of L is $h^{-1}(L) = \{(0+1)^i 2^j (0+1)^i 2^j \mid i, j \geq 1\}$, which is not an RCL by Theorem 2.

Even though RCLs are not closed under inverse homomorphisms, analyzing exactly why this is not the case highlights something that distinguishes the languages of FSBMs from many other well-known language classes. The pivotal point comes from the one-to-many mapping. At first glance, one might try to apply the conventional construction for showing closure under inverse homomorphism of FSAs, i.e. build a new machine M', which reads any symbol x in the new alphabet and simulates M on $h(x)$, as shown in Figure 18.

But this construction fails to generate the full language $h^{-1}(L(M))$: the constructed machine M' still imposes an identity requirement, and therefore fails to accept strings such as 'tait' where the two occurrences of V are mapped by h^{-1} to distinct symbols. The application of an inverse homomorphism — unlike the application of a homomorphism — can "disrupt" sub-string identity relationships that the construction of a new FSBM will necessarily maintain.



(a) $L(M) = \{CVVC\}$



(b) $h : \{a, i, t\} \to \{C, V\}^*$ *with* $h(a) = V$, $h(i) = V$ *and* $h(t) = C$. $L(M') = \{taat, tiit\}$ *but* $h^{-1}(L) = \{taat, tiit, tait, tiat\}$

.

Figure 18: *The conventional construction of the inverse homomorphic image undergenerates*

5.6 *An equivalent extension of regular expressions*

The standard class of regular languages can be defined either via FSAs or via regular expressions. FSBMs constitute a minimal enrichment of FSAs that allow for copying. Here we present a corresponding way to enrich regular expressions that leads to the same class of languages as FSBMs. This provides an alternative characterization of the RCL class in terms of language-theoretic closure properties.

Definition 10. *Let Σ be an alphabet. The regular copying expressions (RCEs) over Σ and the languages they denote are defined as follows.*

- *$\emptyset$ is an RCE and $\mathscr{L}(\emptyset) = \emptyset$*
- *ϵ is an RCE and $\mathscr{L}(\epsilon) = \{\epsilon\}$*
- *$\forall a \in \Sigma$, a is an RCE and $\mathscr{L}(a) = \{a\}$*
- *If R_1 and R_2 are RCEs, then $R_1 + R_2$, R_1R_2, and R_1^* are RCEs, and $\mathscr{L}(R_1 + R_2) = \mathscr{L}(R_1) \cup \mathscr{L}(R_2)$, $\mathscr{L}(R_1R_2) = \{uv \mid u \in \mathscr{L}(R_1), v \in \mathscr{L}(R_2)\}$, and $\mathscr{L}(R_1^*) = (\mathscr{L}(R_1))^*$.*
- *(new copying operator) If R_1 is a **regular** expression, R_1^C is an RCE and $\mathscr{L}(R_1^C) = \{ww \mid w \in \mathscr{L}(R_1)\}$*

RCEs introduce two modifications to regular expressions. First, a $\cdot^C$ expression operator for the copying-derived language is added. Then, the closure under other regular operations is extended to all RCEs. Therefore, languages denoted by regular copying expressions are closed under concatenation, union and Kleene star. Second, the copying operation is only granted access to regular expressions, namely to regular sets formed *without* the use of copying. In other words, the

languages denoted by RCEs are not closed under copying, thus restricting the denoted languages by excluding w^{2^n}.

Given Σ^* is a regular language, an RCE for the simplest copying language $L_{ww} = \{ww \mid w \in \Sigma^*\}$ with $\Sigma = \{a, b\}$ would be $((a + b)^*)^C$. Assume $\Sigma = \{C, V\}$, a naive RCE describing Agta plurals after CVC-reduplication without considering the rest of the syllable structures could be $(CVC)^C(V + C)^*$. This denotes a regular language, unlike $((a+b)^*)^C$. Note, $((CVC)^C(V+C)^*)^C$ is not a regular copying expression, because the copying operator cannot apply to the expressions containing copying.

As noted in footnote 7, there are a number of definitions of "extended regular expressions" in the literature that incorporate some form of back-references (e.g. Câmpeanu *et al.* 2002; Câmpeanu *et al.* 2003; Carle and Narendran 2009), and these motivated the development of Memory Automata (MFAs; Schmid 2016; Freydenberger and Schmid 2019). Just as FSBMs can be seen as a restricted special case of MFAs, RCEs correspond to a special case of extended regular expressions: essentially, an RCE of the form R^C is equivalent to $(R)\backslash 1$, where the back-reference necessarily immediately follows the captured group.

For further details of the equivalence of RCEs and FSBMs, see Appendix B.

DISCUSSION & FURTHER IMPLICATIONS 6

Typology of reduplication 6.1

Here we briefly consider some more complicated kinds of reduplication that are beyond the capacity of FSBMs as formulated here. We sketch some possible ways in which FSBMs might provide a starting point for future work that aims for a proper treatment of the full range of natural language reduplication phenomena.

Non-local Reduplication

Non-local reduplication is the case when the surface phonological strings have non-adjacent copies, incurring non-local correspondence among symbols.[11] A more comprehensive typology and linguistic analysis on non-local reduplication can be found in Riggle (2004a). Examples from Creek are shown in Table 7.

Non-local reduplication Creek plural		
Gloss	*Singular*	*plural*
'precious'	a-cá:k-i:	a-cá:**ca**k-í:
'clean'	hasátk-i:	hasat**ha**k-í:
'soft'	lowáck-i:	lowac**lo**k-i:

Table 7: *Creek plural; CV-copying placed before the final consonant of the root (Booker 1979; Riggle 2004a)*

Marantz (1982) described the adjacency between the reduplicant and the base as a general typological trend. There were proposals (e.g. Nelson 2005) arguing the inviolability of Marantz's generalization, either classifying some patterns as non-reduplicative copying but due to drives to satisfy templates, or suggesting that copying is still local and deletion motivated by other phonological constraints complicates the situation. Riggle (2004a) used the Creek words in Table 7 to argue for true non-local correspondence relations.

FSBMs' current limitation to local reduplication comes from the requirement that B-mode computation has to be directly followed by the buffer-emptying process, and a filled buffer is not allowed in N mode. A possible modification to allow non-local reduplication would be to allow the buffer to be filled in N mode and encode such a possibility in another kind of special states, say J, which stops the machine from buffering, with the buffer only being matched against input and

[11] Bambara 'Noun o Noun' illustrates a particularly simple kind of non-local reduplication where the intervening string is *always* the fixed string 'o'. This could be relatively easily handled by specifying an fixed string to each H state, to be inserted between the two copies when the buffer is emptied. The examples discussed in the main text are when the intervening elements are variable, different from the Bambara-like examples in important ways.

emptied when an H state is encountered. The transitions leading from a G state to a J state would consume symbols in the input tape and buffer symbols in the queue-like buffer. Then, if there is no adjacent H following the end of buffering, the machine can use plain transitions to plain states for only input symbols. The buffer with symbols in it should be kept unchanged. Ultimately, the machine has to encounter some H states to empty the buffer to accept the string, since no final configuration allows symbols on the buffer.

Such a modification might not affect much of the proof ideas of the theorems constructed so far. Regarding the pumping lemma, Condition 2 can be modified by including a sub-string of intervening segments in between two copies. That is, $w \in L$ with $w > 5k$ can be rewritten as $w = ux_1x_2x_3\mathbf{y}x_1x_2x_3v$ such that $\forall i \in \mathbb{N}, ux_1x_2^ix_3\mathbf{y}x_1x_2^ix_3v \in L$. It is worth pointing out that if the generalization in Creek is productive, the sub-string of intervening segments between copies could be unboundedly long.

Multiple Reduplication Here, multiple reduplication refers to the cases when two or more different reduplicative patterns appear in one word. One string can have multiple sub-strings identical to each other. Examples from Nlaka'pamux (previously known as Thompson), a Salish language, are listed in Table 8. See Zimmermann (2019) for a complete typological survey and classification.

Multiple reduplication Nlaka'pamux (Broselow 1983, p.329)	
Gloss	Strings
calico	sil
DIM-calico	sí-sil'
DIST-calico	sil-síl
DIST-DIM-calico	sil-sí-sil'

Table 8: *Multiple reduplication in Nlaka'pamux*

While the computational nature of multiple reduplication in natural language phonology and morphology remains an open question,[12] FSBMs could be relatively easily modified to include multiple copies

of the same base form ($\{w^n \mid w \in \Sigma^*, n \in \mathbb{N}\}$), where n might be tied to the number of copying operations in a language. Given a natural number n, an appropriate modification of FSBMs might allow for the buffered symbols to not be emptied until they have been matched n times against the input.

On the other hand, FSBMs cannot be easily modified to recognize the language $\{w^{2^n} \mid w \in \Sigma^*, n \in \mathbb{N}\}$, where ww strings are themselves copied (i.e. $\{w, ww, wwww, \dots\}$, excluding www).

It is worth carefully distinguishing between the sense of "copying" instantiated by ww and w^n on the one hand, and the sense instantiated by $w^{(2^n)}$ on the other. The former sense highlights the fact that certain portions of a string are identical to certain other portions, whereas the latter is a natural interpretation of the idea that there is a copying *operation* that can apply to *its own outputs*. The kind of recursive copying exhibited by $w^{(2^n)}$ means that this language does not have the constant growth property that Joshi (1985) identified as a criterion for mild context-sensitivity. Excluding this recursive copying from phonology seems relatively well-justified, on the grounds that triplication is attested (Zimmermann 2019; Rawski *et al.* 2023). But the situation may be different for syntax, where Kobele (2006), for example, has argued for recursive copying of the $w^{(2^n)}$ sort on the basis of Yoruba relativized predicates. See also Clark and Yoshinaka (2014) on the relationship between parallel multiple context-free grammars (PMCFGs) and multiple context-free grammars (MCFGs); and Stabler (2004) on the comparison between what he calls *generating grammars* and *copying grammars*.

Reduplication with non-identical copies In natural languages, non-identical copies are prevalent. There are cases where other phonological processes apply to the base or the reduplicant to create nonidentical copies, such as onset cluster simplification in Tagalog partial reduplication (Zuraw 1996), e.g. *'X is working'* [nag-ta-tɾabahoh], mapped from [tɾabahoh]. Another type of non-identical copies involves a fixed, memorized segment/sub-string (Alderete *et al.* 1999). Examples are

[12] For recent phonological analyses, see Zimmermann (2021a) and Zimmermann (2021b). For a more detailed discussion on the string-to-string function version of this problem, see Rawski *et al.* (2023).

given in Mongolian, illustrated in Table 9, where whole stems are copied to create forms with the meaning 'X and such things'. However, the initial consonant is always rewritten as [m].[13]

Non-identical copies Mongolian Noun Reduplication (Svantesson *et al.* 2005, pp.60)		
Gloss	*root*	*X and such things*
'gown'	teeɮ	teeɮ-meeɮ
'beard'	t^haɮx	t^haɮx-m^{h}aɮx
'eye'	nut	nut-mut

Table 9: *Non-identical copies in Mongolian*

One way to modify FSBMs to accommodate non-identical copies would be to allow the machine to either store or empty not exactly the same input symbols, but the image of the inputs symbols under some alphabetic mapping, or finite-state transduction, f. For example, to account for the fixed consonant in Mongolian, we can introduce a finite state transduction $f_{C_1 \to m}$ that rewrites the first consonant to [m]. To empty the buffer, instead of checking the identity relation, it determines whether $f_{C_1 \to m}(x) = y$ where x is in the buffer and y is a prefix of the remaining input.

If no restrictions at all are imposed on the transduction, then the modified automata would recognize the context-free $\{a^n b^n \mid n \in \mathbb{N}\}$ with $f(a) = b$ in a manner that (unlike a context-free grammar) associates the first 'a' with the first 'b' and so on, though still excluding string reversals. Moreover, the resulting language set would also include $\{a^i b^j c^i d^j \mid i, j \geq 1\}$ with $f(a) = c, f(b) = d$. It could be fruitful for further studies to examine possible restrictions on the transduction.

6.2 *A note (and a conjecture) regarding determinism*

A natural question to consider is whether the non-determinism that we have allowed in FSBMs is essential.[14] A proper treatment of this

[13] When the stem form starts with [m], it is always rewritten to [c]. For example, the reduplicated form of [maɮ] 'cattle' is [maɮ-caɮ]

issue turns out to be more subtle than it might initially appear, but we offer some initial observations here.

The FSBM in Figure 19 is non-deterministic in the sense that the string aa might lead the machine either to q_2 or to q_3. This familiar kind of non-determinism brings no additional expressive power in the case of standard FSAs, where the subset construction can be used to determinize any FSA. But this method for determinization cannot be straightforwardly applied to FSBMs, because of the distinguished status of G and H states. Applying the construction to the FSBM in Figure 19 would yield a new state corresponding to $\{q_2, q_3\}$, and then the question arises of whether this new state should be an H state (like q_3) or not (like q_2). Neither answer is sufficient: in the new machine, the string aa will deterministically lead to the state $\{q_2, q_3\}$, but the prefix aa may or may not be the entire string that needs to be buffered and copied.

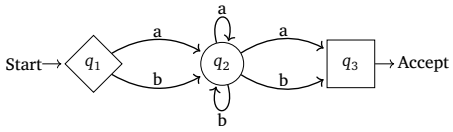


Figure 19: *An FSBM illustrating nondeterminism*

Stated slightly more generally, the subset construction can eliminate non-determinism *between states* ("state-nondeterminism"), but in FSBMs there is also the possibility of nondeterminism *between modes* ("mode-nondeterminism"). The state-nondeterminism indicated in (5) could be eliminated, in a sense, by applying the subset construction to yield a new machine M' with transitions as in (6).

(5) a. $(aa\ldots, q_1, \textsc{n}, \epsilon) \vdash_M^* (\ldots, q_2, \textsc{b}, aa)$

b. $(aa\ldots, q_1, \textsc{n}, \epsilon) \vdash_M^* (\ldots, q_3, \textsc{b}, aa)$

(6) $(aa\ldots, \{q_1\}, \textsc{n}, \epsilon) \vdash_{M'}^* (\ldots, \{q_2, q_3\}, \textsc{b}, aa)$

But the two configurations reached in (5) differ in whether M will stop buffering after this prefix aa, and we suspect that there is no way to eliminate this kind of nondeterminism between modes. To bring out

[14] Thanks to two reviewers for drawing our attention to this.

this important additional distinction, consider the transition sequences in (7) for the longer prefix $aaaa$.

(7) a. $(aaaa\ldots, q_1, \textsc{n}, \epsilon) \vdash_M^* (aa\ldots, q_2, \textsc{b}, aa) \vdash_M^* (\ldots, q_2, \textsc{b}, aaaa)$
b. $(aaaa\ldots, q_1, \textsc{n}, \epsilon) \vdash_M^* (aa\ldots, q_3, \textsc{b}, aa) \vdash_M (\ldots, q_3, \textsc{n}, \epsilon)$

So there is something distinctive about the kind of non-determinism in Figure 19, which lies not in the fact that the prefix aa might lead to either state q_2 or state q_3, but rather the fact that the prefix $aaaa$ might lead to either state q_2 in mode B, or state q_3 in mode N.

The following definition makes a first attempt at pinpointing the distinctive kind of non-determinism in Figure 19.

Definition 11. *An FSBM M is mode-deterministic if there do not exist three configurations $C = (w, q, m, v)$, $C_1 = (\epsilon, q_1, m_1, v_1)$ and $C_2 = (\epsilon, q_2, m_2, v_2)$, such that*

- $C \vdash_M^* C_1$ *and* $C \vdash_M^* C_2$,
- $C_1 \nvdash_M^* C_2$ *and* $C_2 \nvdash_M^* C_1$, *and*
- $m_1 \neq m_2$.

The FSBM in Figure 20, for example, is mode-deterministic in this sense, whereas (7) demonstrates that the FSBM in Figure 19 is not. We conjecture that the mode-deterministic FSBMs are properly less powerful than the full class of FSBMs, and in particular that there is no mode-deterministic FSBM that generates the same language as the FSBM in Figure 19.

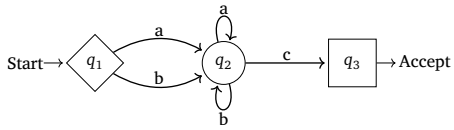


Figure 20: *An FSBM illustrating mode-determinism*

6.3 *The role of symbol identity*

A noteworthy trait of the RCL class is its non-closure under inverse homomorphisms. This distinguishes the RCL class from many of the

familiar language classes that have played a role in the analysis of natural languages: the regular class and the context-free class are each closed under both homomorphisms and inverse homomorphisms, as are prominent classes in the "mildly context sensitive" region, such as the tree-adjoining languages and multiple context-free languages (Joshi 1985; Kallmeyer 2010).

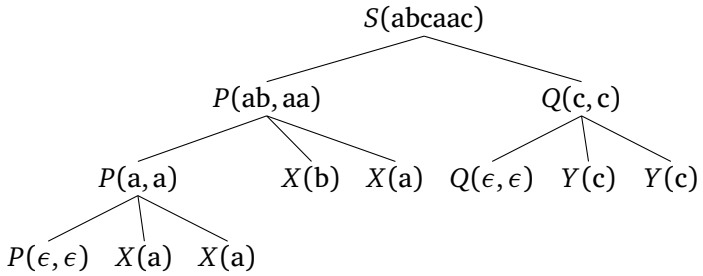
To illustrate, consider the relationship between the following two languages:

$$L_1 = (\mathrm{a}+\mathrm{b})^i \mathrm{c}^j (\mathrm{a}+\mathrm{b})^i \mathrm{c}^j$$
$$L_2 = \mathrm{a}^i \mathrm{c}^j \mathrm{a}^i \mathrm{c}^j$$

We showed above that L_1 is not an RCL, whereas L_2 obviously is. This sets the RCL class apart from the regular and context-free classes, which contain neither L_1 nor L_2, and from the tree-adjoining and multiple context-free classes, which contain both; recall Figure 5. For all these other formalisms, the surface differences between L_1 and L_2 are essentially irrelevant. For example, a multiple context-free grammar (MCFG; Seki *et al.* 1991; Kallmeyer 2010) for L_1 is given in (8), and (9) shows an illustrative derivation for the string 'abcaac'. This grammar uses the nonterminals P and Q to control the assembly of (discontinuous) '$(\mathrm{a}+\mathrm{b})^i \ldots (\mathrm{a}+\mathrm{b})^i$' and '$\mathrm{c}^j \ldots \mathrm{c}^j$' portions respectively; P-portions can grow via the addition of X elements, and Q-portions can grow via the addition of Y elements.

(8)
$$S(u_1 v_1 u_2 v_2) \rightarrow P(u_1, u_2)\ Q(v_1, v_2)$$
$$P(\epsilon, \epsilon)$$
$$P(u_1 v, u_2 w) \rightarrow P(u_1, u_2)\ X(v)\ X(w)$$
$$Q(\epsilon, \epsilon)$$
$$Q(u_1 v, u_2 w) \rightarrow Q(u_1, u_2)\ Y(v)\ Y(w)$$
$$X(\mathrm{a})$$
$$X(\mathrm{b})$$
$$Y(\mathrm{c})$$

(9)



Notice that to generate L_2 instead of L_1, we would simply omit the rule $X(\mathrm{b})$ from (8). What this highlights is that for either L_1 or L_2, the significant work is done by the rules that arrange the yields of the nonterminals X and Y appropriately, and this work can be dissociated from the rules that specify the terminal symbols that can appear as the yields of X and Y. The nonterminals provide a grammar-internal mechanism for doing the book-keeping necessary to enforce the abstract pattern shared by L_1 and L_2, and the relationship between these grammar-internal symbols and the terminal symbols that make up the generated strings is opaque.

In an FSBM, on the other hand, the machinery that extends the formalism beyond the regular languages has no analogous grammar-internal book-keeping mechanism that can be dissociated from surface symbols: the non-regular effects of an FSBM's string-buffering mechanism are inherently tied to the identity of certain surface symbols. This is what underlies the crucial difference between L_1 and L_2 for FSBMs, and the non-closure under inverse homomorphisms of RCLs.[15]

[15] Of course the states of an FSBM are grammar-internal symbols in the relevant sense, and this is in effect what allows FSAs to be closed under both homomorphisms and inverse homomorphisms. But the point of the discussion here is to look at the distinctive additional capacities of FSBMs, which are brought out by considering a non-regular language such as L_2.

A comparison with Savitch's RPDAs (discussed above; Savitch 1989) is informative: RPDAs, while similar in some respects to FSBMs, generate a class of languages that *is* closed under both inverse homomorphism and homomorphism (in fact, under any finite-state transduction). This difference stems from the fact that an RPDA's queue-like memory arises from relaxing restrictions on a standard PDA's stack, and so the queue-like memory uses a distinct alphabet of "stack symbols" rather than surface symbols. These stack symbols are grammar-internal book-keeping devices whose relationship to surface symbols can be specified by

To put a label on this distinction, we might say that FSBMs are "symbol-oriented" (where by "symbol" we mean surface/terminal symbol), in contrast to the other formalisms mentioned above. Suppose, to make this precise, we say that a formalism (or a language class) is **symbol-oriented** iff it fails to be closed under both homomorphisms and inverse homomorphisms.

It is interesting to note that, while the symbol-oriented nature of FSBMs sets them apart from formalisms (such as MCFGs) motivated by the kinds of non-context-free cross-serial dependencies observed in syntax, this property of FSBMs is shared by other formalisms that have been argued to align well with observed phonological patterns. Many of the sub-regular language classes discussed by Heinz (2007), are also symbol-oriented in this sense. An easy example (Mayer and Major 2018; De Santo and Graf 2019) comes from the Strictly 2-Local (SL_2) languages: $(ab)^*$ is an SL_2 language, but applying the homomorphism h defined by $h(a) = c, h(b) = c$ yields $(cc)^*$, which is not an SL_2 language. So the SL_2 languages are not closed under homomorphisms.

The fact that the SL languages lack closure under homomorphisms, whereas the RCL class lacks closure under *inverse* homomorphisms, reflects the different role that symbol identity plays for the two formalisms. The move from $(ab)^*$ to $(cc)^*$ eliminates distinctions between surface symbols, which removes information that the SL_2 grammar for $(ab)^*$ was using to ensure that the length of each generated string was even. The move from L_2 to L_1, on the other hand, *introduces* distinctions between surface symbols which are incompatible with the string-buffering mechanism of an FSBM. [16]

But the broader point we wish to draw attention to here is the distinction between (i) the context-free class and various mildly context-sensitive classes, which are closed under both homomorphisms and inverse homomorphisms, and (ii) the RCL and SL classes, which are not and therefore exhibit a degree of sensitivity to surface symbol identity. It is intriguing that the insensitivity to surface symbol identity seems to be necessary for many important patterns found in natural

---

the grammar-writer, as in the case of MCFGs such as (8) above.

[16] For similar reasons, the languages of regular expressions extended with back-references are also not closed under inverse homomorphism (Câmpeanu *et al.* 2003).

language syntax — for example, the classic cross-serial dependencies in Swiss German (Shieber 1985) correspond to $a^i b^j c^i d^j$, rather than $a^i b^j a^i b^j$ — whereas many phonological patterns that have been studied computationally are compatible with symbol-oriented formalisms. This includes both the sub-regular patterns that motivate formalisms such as SL grammars, and the non-regular reduplication patterns that motivate FSBMs.

A complication to this clear picture may come from copying patterns in syntax, for example the Yoruba constructions discussed by Kobele (2006), mentioned above in Section 6.1. The languages generated by parallel multiple context-free grammars (PMCFGs) are not closed under inverse homomorphisms (Nishida and Seki 2000, p. 145, Corollary 12), for reasons analogous to what we have seen for FSBMs, and so this is an example of a symbol-oriented formalism that has been argued to be appropriate for syntax. But it is clear that syntax requires at least some *non*-symbol-oriented mechanisms to generate the well-known cross-serial dependencies of the Swiss-German sort ($a^i b^j c^i d^j$), whereas those cross-serial dependencies that we do observe in phonology are compatible with the more restricted, symbol-oriented notion of cross-serial dependencies that appear in reduplication.

CONCLUSION 7

This paper has looked at formal computational properties of unbounded copying on regular languages, including the simplest copying language L_{ww} where w can be any arbitrary string over an alphabet. We have proposed a new computational device: finite-state buffered machines (FSBMs), which add copying to regular languages by adding an unbounded queue-structured memory buffer, with specified states restricting how this memory buffer is used. As a result, we introduce a new class of languages, which is incomparable to context-free languages, named regular copying languages (RCLs).

This class of languages extends regular languages with *unbounded* copying but excludes non-reduplicative non-regular patterns. Context-free string reversals are excluded since the buffer is queue-like, and the

mildly context-sensitive Swiss-German cross-serial dependency pattern, 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.

We have also surveyed the class's closure properties and proved a pumping lemma. This language set is closed under union, concatenation, Kleene Star, homomorphism, and intersection with regular languages. It is not closed under copying, inhibiting the recursive application of copying and excluding non-semilinear $w^{(2^n)}$. This class is also not closed under intersection, nor complementation. Finally, it is not closed under inverse homomorphism, given it cannot recover the possibility of non-identity among corresponding segments when the mapping is many-to-one (and the inverse homomorphic image is one-to-many); we suggested that this might reflect an important difference between the string-generating mechanisms of phonology and syntax.

One potential direction for future research is to connect FSBMs with the 2-way D-FSTs studied by Dolatian and Heinz (2018a,b, 2019, 2020), which successfully model unbounded copying as *function*s while excluding mirror image mappings. We briefly mention two possibilities along these lines. First, it will be interesting to compare the RCL class of languages with the image of the functions studied by Dolatian and Heinz (2020). Second, it is natural to consider adding to FSBMs another tape for output strings, extending from acceptors (as presented here) to finite-state buffered transducers (FSBTs). The morphological analysis ($ww \mapsto w$) problem is claimed to be difficult for 2-way D-FSTs, since they are not invertible. Our intuition is that FSBTs might help solve this issue: after reading the first w in input and buffering this string in memory, the machine can write ϵ to the output tape when it matches the buffered string against the contents of the input tape. But a more detailed and rigorous study is required in this direction.

We are currently investigating the learning and learnability of FSBMs and copying in sub-regular phonology. The RCL class itself cannot be identified in the limit, since it properly contains the regular class (Gold 1967). However, we take positive learning results from Clark and Yoshinaka (2014) and Clark *et al.* (2016) on PMCFGs with copying, and from Dolatian and Heinz (2018b) on Concatenated Output Strictly Local functions for reduplication, as suggestions for future

directions towards learning results for FSBMs. In particular, one of the most attractive properties of the sub-regular classes is their Gold-learnability (e.g. Garcia *et al.* 1990; Heinz 2010; Chandlee *et al.* 2014; Jardine and Heinz 2016). We hope to explore whether the learnability property still holds once copying is added to these sub-regular classes.

Last but not least, the current class of languages excludes non-adjacent copies, multiple reduplication and reduplication with non-identical copies. We briefly sketched some possible modifications and their potential effects. We hope that our proposal here provides a useful framework for better understanding the formal issues raised by these more complex reduplication phenomena, and guiding empirical research into their typology.

ACKNOWLEDGMENT

We would like to thank Dylan Bumford, Hossep Dolatian, Bruce Hayes, Ed Keenan, Claire Moore-Cantwell, Kie Zuraw, the audience at the UCLA Phonology Seminar, the reviewers and audience at the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, the ESSLLI 2021 Student Session, and the 2021 Annual Meeting on Phonology for helpful input and discussion. Thanks also to the three anonymous reviewers for their valuable feedback. All errors remain our own.

REFERENCES

Daniel M Albro (1998), *Evaluation, implementation, and extension of Primitive Optimality Theory*, Master's thesis, UCLA.

John Alderete, Jill Beckman, Laura Benua, Amalia Gnanadesikan, John McCarthy, and Suzanne Urbanczyk (1999), Reduplication with Fixed Segmentism, *Linguistic Inquiry*, 30(3):327–364, ISSN 0024-3892, doi:10.1162/002438999554101, https://doi.org/10.1162/002438999554101.

Rajeev ALUR and Pavol ČERNÝ (2010), Expressiveness of streaming string transducers, in Kamal LODAYA and Meena MAHAJAN, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010)*, volume 8 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pp. 1–12, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, ISBN 978-3-939897-23-1, ISSN 1868-8969, doi:10.4230/LIPIcs.FSTTCS.2010.1, `http://drops.dagstuhl.de/opus/volltexte/2010/2853`.

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)*, pp. 1–12, doi:10.1109/LICS.2017.8005138.

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*, pp. 191–198, Association for Computational Linguistics, Hong Kong, doi:10.3115/1075218.1075243, `https://www.aclweb.org/anthology/P00-1025`.

Karen M. BOOKER (1979), *Comparative Muskogean: Aspects of Proto-Muskogean Verb Morphology*, Ph.D. thesis, University of Kansas.

Ellen I. BROSELOW (1983), Salish double reduplications: Subjacency in morphology, *Natural Language & Linguistic Theory*, 1:317–346.

Cezar CÂMPEANU, Kai SALOMAA, and Sheng YU (2002), Regex and Extended Regex, in Jean-Marc CHAMPARNAUD and Denis MAUREL, editors, *Implementation and Application of Automata, 7th International Conference, CIAA 2002, Tours, France, July 3-5, 2002, Revised Papers*, volume 2608 of *Lecture Notes in Computer Science*, pp. 77–84, Springer, doi:10.1007/3-540-44977-9_7, `https://doi.org/10.1007/3-540-44977-9_7`.

Cezar CÂMPEANU, Kai SALOMAA, and Sheng YU (2003), A formal study of practical regular expressions, *International Journal of Foundations of Computer Science*, 14(06):1007–1018.

Benjamin CARLE and Paliath NARENDRAN (2009), On extended regular expressions, in *Language and Automata Theory and Applications: Third International Conference, LATA 2009, Tarragona, Spain, April 2-8, 2009. Proceedings 3*, pp. 279–289, Springer.

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.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz (2014), Learning Strictly Local Subsequential Functions, *Transactions of the Association for Computational Linguistics*, 2:491–504, doi:10.1162/tacl_a_00198, https://aclanthology.org/Q14-1038.

Jane Chandlee and Jeffrey Heinz (2012), Bounded copying is subsequential: Implications for metathesis and reduplication, in *Proceedings of the Twelfth Meeting of the Special Interest Group on Computational Morphology and Phonology*, pp. 42–51, Association for Computational Linguistics, Montréal, Canada, https://www.aclweb.org/anthology/W12-2306.

Alexander Clark, Makoto Kanazawa, Gregory M Kobele, and Ryo Yoshinaka (2016), Distributional learning of some nonlinear tree grammars, *Fundamenta Informaticae*, 146(4):339–377.

Alexander Clark and Ryo Yoshinaka (2014), Distributional Learning of Parallel Multiple Context-Free Grammars, *Mach. Learn.*, 96(1–2):5–31, ISSN 0885-6125, doi:10.1007/s10994-013-5403-2, https://doi.org/10.1007/s10994-013-5403-2.

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.

Aniello De Santo and Thomas Graf (2019), Structure Sensitive Tier Projection: Applications and Formal Properties, in Raffaella Bernardi, Greg Kobele, and Sylvain Pogodalla, editors, *Formal Grammar*, pp. 35–50, Springer Berlin Heidelberg, ISBN 978-3-662-59648-7.

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 Olgierd Unold, Witold Dyrka, and Wojciech Wieczorek, editors, *Proceedings of the 14th International Conference on Grammatical Inference*, volume 93 of *Proceedings of Machine Learning Research*, pp. 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*, pp. 66–77, Association for Computational Linguistics, Brussels, Belgium, doi:10.18653/v1/W18-5807.

Hossep Dolatian and Jeffrey Heinz (2019), RedTyp: A Database of Reduplication with Computational Models, in *Proceedings of the Society for Computation in Linguistics*, volume 2, article 3.

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*, pp. 313–320, Association for Computational Linguistics, Madrid, Spain, doi:10.3115/976909.979657, https://www.aclweb.org/anthology/P97-1040.

T. Mark Ellison (1994), Phonological Derivation in Optimality Theory, in *Proceedings of the 15th Conference on Computational Linguistics - Volume 2*, COLING '94, pp. 1007–1013, Association for Computational Linguistics, USA, doi:10.3115/991250.991312, https://doi.org/10.3115/991250.991312.

Joost Engelfriet and Hendrik Jan Hoogeboom (1999), MSO definable string transductions and two-way finite state transducers, doi:10.48550/ARXIV.CS/9906007, https://arxiv.org/abs/cs/9906007.

Dominik D Freydenberger and Markus L Schmid (2019), Deterministic regular expressions with back-references, *Journal of Computer and System Sciences*, 105:1–39.

Pedro Garcia, Enrique Vidal, and José Oncina (1990), Learning Locally Testable Languages in the Strict Sense., in *Proceedings of the Workshop on Algorithmic Learning Theory*, pp. 325–338.

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

David Gil (1996), How to speak backwards in Tagalog, in *Pan-Asiatic Linguistics, Proceedings of the Fourth International Symposium on Language and Linguistics, January 8-10, 1996, Institute of Language and Culture for Rural Development, Mahidol University at Salaya*, volume 1, pp. 297–306.

E Mark Gold (1967), Language identification in the limit, *Information and control*, 10(5):447–474.

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 (2010), String Extension Learning, in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pp. 897–906, Association for Computational Linguistics, Uppsala, Sweden.

Jeffrey Heinz (2018), The computational nature of phonological generalizations, in Larry Hyman and Frans Plank, editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pp. 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*, pp. 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, `http://hdl.handle.net/10150/196112`.

Riny HUYBREGTS (1984), The weak inadequacy of context-free phrase structure grammars, *Van periferie naar kern*, pp. 81–99.

Sharon INKELAS (2008), The dual theory of reduplication, 46(2):351–401, doi:doi:10.1515/LING.2008.013, `https://doi.org/10.1515/LING.2008.013`.

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.

Adam JARDINE and Jeffrey HEINZ (2016), Learning Tier-based Strictly 2-Local Languages, *Transactions of the Association for Computational Linguistics*, 4:87–98, ISSN 2307-387X, `https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/694`.

Aravind K. JOSHI (1985), *Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions?*, pp. 206–250, Studies in Natural Language Processing, Cambridge University Press, doi:10.1017/CBO9780511597855.007.

Laura KALLMEYER (2010), *Parsing Beyond Context-Free Grammars*, Cognitive Technologies, Springer, ISBN 978-3-642-14845-3, doi:10.1007/978-3-642-14846-0, `https://doi.org/10.1007/978-3-642-14846-0`.

Ronald M. KAPLAN and Martin KAY (1994), Regular Models of Phonological Rule Systems, *Computational Linguistics*, 20(3):331–378, ISSN 0891-2017.

Gregory M. KOBELE (2006), *Generating Copies: An investigation into structural identity in language and grammar.*, Ph.D. thesis, University of California, Los Angeles.

Martin KUTRIB, Andreas MALCHER, and Matthias WENDLANDT (2018), *Queue Automata: Foundations and Developments*, pp. 385–431, Springer International Publishing, Cham, ISBN 978-3-319-73216-9, doi:10.1007/978-3-319-73216-9_19, `https://doi.org/10.1007/978-3-319-73216-9_19`.

Alexis MANASTER-RAMER (1986), Copying in Natural Languages, Context-Freeness, and Queue Grammars, in *Proceedings of the 24th Annual Meeting on Association for Computational Linguistics*, ACL '86, pp. 85–89, Association for Computational Linguistics, USA, doi:10.3115/981131.981145, https://doi.org/10.3115/981131.981145.

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

Gary F. MARCUS, Sugumaran VIJAYAN, Shoba Bandi RAO, and Peter M. VISHTON (1999), Rule Learning by Seven-Month-Old Infants, *Science*, 283(5398):77–80, ISSN 00368075, 10959203, http://www.jstor.org/stable/2897195.

Connor MAYER and Travis MAJOR (2018), A Challenge for Tier-Based Strict Locality from Uyghur Backness Harmony, in Annie FORET, Greg KOBELE, and Sylvain POGODALLA, editors, *Formal Grammar 2018*, pp. 62–83, Springer Berlin Heidelberg, ISBN 978-3-662-57784-4.

Edith A. MORAVCSIK (1978), Reduplicative Constructions, in GREENBERG, J. H., and ET AL., editors, *Universals of Human Language. Volume 3: Word Structure*, pp. 297–334, Stanford University Press, Stanford.

Elliott MORETON, Brandon PRICKETT, Katya PERTSOVA, Joshua FENNELL, Joe PATER, and Lisa SANDERS (2021), Learning Reduplication, but not Syllable Reversal, in Ryan BENNETT, Richard BIBBS, Mykel Loren BRINKERHOFF, Stephanie Rich MAX J. KAPLAN, Amanda RYSLING, Nicholas Van HANDEL, and Maya Wax CAVALLARO, editors, *Supplemental Proceedings of the 2020 Annual Meeting on Phonology*, https://journals.linguisticsociety.org/proceedings/index.php/amphonology/article/view/4912/4634.

Nicole NELSON (2005), *Wrong side reduplication is epiphenomenal: Evidence from Yoruba*, pp. 135–160, De Gruyter Mouton, Berlin, Boston, doi:doi:10.1515/9783110911466.135, https://doi.org/10.1515/9783110911466.135.

Taishin Y. NISHIDA and Shigeko SEKI (2000), Grouped partial ET0L systems and parallel multiple context-free grammars, *Theoretical Computer Science*, 246(1):131–150, ISSN 0304-3975, doi:https://doi.org/10.1016/S0304-3975(99)00076-6, https://www.sciencedirect.com/science/article/pii/S0304397599000766.

Jonathan RAWSKI, Hossep DOLATIAN, Jeffrey HEINZ, and Eric RAIMY (2023), Regular and polyregular theories of reduplication, *Glossa: a journal of general linguistics*, 8(1).

Jason RIGGLE (2004a), Nonlocal Reduplication, in *Proceedings of the 34th Meeting of the North-East Linguistics Society (NELS 34)*, pp. 485–496, GLSA, University of Massachusetts, USA, https://doi.org/doi:10.7282/T3GT5PZF.

Jason Alan Riggle (2004b), *Generation, recognition, and learning in finite state Optimality Theory*, University of California, Los Angeles.

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

Carl Rubino (2005), *Reduplication: Form, function and distribution*, pp. 11–30, De Gruyter Mouton, Berlin, Boston, doi:doi:10.1515/9783110911466.11, https://doi.org/10.1515/9783110911466.11.

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, https://wals.info/chapter/27.

Edward Sapir and Harry Hoijer (1967), *The Phonology and Morphology of the Navaho Language*, number v. 50-51 in The Phonology and Morphology of the Navaho Language, University of California Press.

Walter Savitch (1989), A Formal Model for Context-Free Languages Augmented with Reduplication, *Computational Linguistics*, 15(4):250–261, https://aclanthology.org/J89-4003.

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

Markus L Schmid (2016), Characterising REGEX languages by regular languages equipped with factor-referencing, *Information and Computation*, 249:1–17.

Hiroyuki Seki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami (1991), On multiple context-free grammars, *Theoretical Computer Science*, 88(2):191–229, ISSN 0304-3975, doi:https://doi.org/10.1016/0304-3975(91)90374-B, https://www.sciencedirect.com/science/article/pii/030439759190374B.

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

Michael Sipser (2013), *Introduction to the Theory of Computation*, Course Technology, Boston, MA, third edition, ISBN 113318779X.

Paul Smolensky and Alan Prince (1993), Optimality Theory: Constraint interaction in generative grammar, *Optimality Theory in phonology*, 3.

Edward P. Stabler (2004), Varieties of crossing dependencies: Structure dependence and mild context sensitivity, *Cognitive Science*, 93(5):699–720.

Jan-Olof Svantesson, Anna Tsendina, Anastasia Karlsson, and Vivan Franzén (2005), *The phonology of Mongolian*, OUP Oxford.

Rachelle Waksler (1999), Cross-Linguistic Evidence for Morphological Representation in the Mental Lexicon, *Brain and Language*, 68(1):68–74, ISSN

0093-934X, doi:https://doi.org/10.1006/brln.1999.2117, https://www.sciencedirect.com/science/article/pii/S0093934X9992117X.

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*, https://www.aclweb.org/anthology/A00-2039.

Yang WANG (2021a), Recognizing Reduplicated Forms: Finite-State Buffered Machines, in *Proceedings of the 18th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, pp. 177–187, Association for Computational Linguistics, Online, doi:10.18653/v1/2021.sigmorphon-1.20, https://aclanthology.org/2021.sigmorphon-1.20.

Yang WANG (2021b), *Regular languages extended with reduplication: Formal models, proofs and illustrations*, Master's thesis, University of California, Los Angeles.

Samantha WRAY, Linnaea STOCKALL, and Alec MARANTZ (2022), Early Form-Based Morphological Decomposition in Tagalog: MEG Evidence from Reduplication, Infixation, and Circumfixation, *Neurobiology of Language*, 3(2):235–255, ISSN 2641-4368, doi:10.1162/nol_a_00062, https://doi.org/10.1162/nol_a_00062.

Fujimura YUKO (2001), Reduplication in standard Malay and Japanese, *Journal of Modern Languages*, 13(1):65–92.

Eva ZIMMERMANN (2019), Database of the DFG project Multiple Reduplication: Typology and Theory, online available at http://www.evazimmermann.org/multiple-reduplication-data.html.

Eva ZIMMERMANN (2021a), Faded copies: Reduplication as distribution of activity, *Glossa: a journal of general linguistics*, 6(1).

Eva ZIMMERMANN (2021b), A phonological account of unfaithful multiple reduplication, *The Linguistic Review*, 38(3):537–572, doi:doi:10.1515/tlr-2021-2075, https://doi.org/10.1515/tlr-2021-2075.

Kie ZURAW (1996), *Floating Phonotactics: Infixation and Reduplication in Tagalog Loanwords*, Master's thesis, UCLA.

Kie ZURAW (2002), Aggressive reduplication, *Phonology*, 19(3):395–439, doi:10.1017/S095267570300441X.

A PROOF OF THEOREM 1

Lemma 2. *For any string w, if $w \in L(M_1 \cap M_2)$, then $w \in L(M_1)$ and $w \in L(M_2)$.*

Proof. Assume $M_1 = \langle Q_1, \Sigma, I_1, F_1, G_1, H_1, \delta_1 \rangle$ and $M_2 = \langle Q_2, \Sigma, I_2, F_2, \delta_2 \rangle$.

Let the run on $M_1 \cap M_2$ that generates w be $D_0, D_1, \ldots, D_m$, where each $D_i = (u_i, (p_i, q_i, \mathbf{A}_i), v_i, m_i)$. We define a sequence $C_0, C_1, \ldots, C_m$ of configurations of M_1, and a sequence $B_0, B_1, \ldots, B_m$ of configurations of M_2, as follows:

$$C_i = (u_i, p_i, v_i, m_i)$$

$$B_i = \begin{cases} (v_i \backslash u_i, q_i) & \text{if } m_i = \text{B and } (p_i, q_i, \mathbf{A}_i) \in (H_1 \times Q_2 \times \{\mathbf{0}\}) = H \\ (u_i, q_i) & \text{otherwise} \end{cases}$$

For the initial configuration $D_0 = (w, (p_0, q_0, \mathbf{A}_0), \epsilon, \text{N})$, we know that $(p_0, q_0, \mathbf{A}_0) \in I$, so $p_0 \in I_1$ and $q_0 \in I_2$. Therefore $C_0 = (w, p_0, \epsilon, n)$ is a valid starting configuration for a run of w on M_1, and $B_0 = (w, q_0)$ is a valid starting configuration for a run of w on M_2.

For the final configuration $D_m = (\epsilon, (p_m, q_m, \mathbf{A}_m), \epsilon, \text{N})$, we know that $(p_m, q_m, \mathbf{A}_m) \in F$, so $p_m \in F_1$ and $q_m \in F_2$. Therefore $C_m = (\epsilon, p_m, \epsilon, \text{N})$ is a valid ending configuration for a run on M_1, and $B_m = (\epsilon, q_m)$ is a valid ending configuration for a run on M_2.

To use the sequences $C_0, \ldots, C_m$ and $B_0, \ldots, B_m$ to establish that $w \in L(M_1)$ and $w \in L(M_2)$, we will show that, for every $i \in \{0, \ldots, m-1\}$, $C_i \vdash^*_{M_1} C_{i+1}$ and $B_i \vdash^*_{M_2} B_{i+1}$.

For each $i \in \{0, \ldots, m-1\}$, we know that $D_i \vdash_{M_1 \cap M_2} D_{i+1}$, so there are four cases to consider:

- Suppose $D_i \vdash_{\text{N}} D_{i+1}$. Then $D_i = (xu_{i+1}, (p_i, q_i, \mathbf{A}_i), \epsilon, \text{N})$ and $D_{i+1} = (u_{i+1}, (p_{i+1}, q_{i+1}, \mathbf{A}_{i+1}), \epsilon, \text{N})$, with $((p_i, q_i, \mathbf{A}_i), x, (p_{i+1}, q_{i+1}, \mathbf{A}_{i+1})) \in \delta$, $(p_i, q_i, \mathbf{A}_i) \notin G$, and $(p_{i+1}, q_{i+1}, \mathbf{A}_{i+1}) \notin H$. Then $C_i = (xu_{i+1}, p_i, \epsilon, \text{N})$, $C_{i+1} = (u_{i+1}, p_i, \epsilon, \text{N})$, $B_i = (xu_{i+1}, q_i)$ and $B_{i+1} = (u_{i+1}, q_{i+1})$. We want to show that $C_i \vdash^*_{M_1} C_{i+1}$ and that $B_i \vdash^*_{M_2} B_{i+1}$.
  - Suppose the critical transition is in δ_{N}. Then $(p_i, x, p_{i+1}) \in \delta_1$ and $p_i \notin G_1$ and $p_{i+1} \notin H_1$, so $C_i \vdash_{\text{N}} C_{i+1}$. Also either $(q_i, x, q_{i+1}) \in \delta_2$, or $x = \epsilon$ and $q_i = q_{i+1}$; so $B_i \vdash^* B_{i+1}$.
  - Suppose the critical transition is in $\delta_{\text{N}\to\text{B}}$. Then $x = \epsilon$, and $p_i = p_{i+1}$ and $q_i = q_{i+1}$. Therefore $C_i = C_{i+1}$ and $B_i = B_{i+1}$.
  - The critical transition cannot be in δ_{B}, because Lemma 1 implies that $\mathbf{A}_i = \mathbf{0}$.
  - The critical transition cannot be in $\delta_{\text{B}\to\text{N}}$, because Lemma 1 implies that $\mathbf{A}_i = \mathbf{0}$.

- Suppose $D_i \vdash_{\text{N}\to\text{B}} D_{i+1}$. Then $D_i = (u_i, (p_i, q_i, \mathbf{A}_i), \epsilon, \text{N})$ and $D_{i+1} = (u_i, (p_i, q_i, \mathbf{A}_i), \epsilon, \text{B})$, with $(p_i, q_i, \mathbf{A}_i) \in G$ and therefore $p_i \in G_1$. So $C_i = (u_i, p_i, \epsilon, \text{N})$ and $C_{i+1} = (u_i, p_i, \epsilon, \text{B})$, and therefore $C_i \vdash_{\text{N}\to\text{B}} C_{i+1}$. Furthermore $B_i = B_{i+1} = (u_i, q_i)$, since $\mathbf{A}_i = \mathbf{A}_\epsilon \neq \mathbf{0}$, so $B_i \vdash^* B_{i+1}$.
- Suppose $D_i \vdash_{\text{B}} D_{i+1}$. Then $D_i = (xu_{i+1}, (p_i, q_i, \mathbf{A}_i), v_i, \text{B})$ and $D_{i+1} = (u_{i+1}, (p_{i+1}, q_{i+1}, \mathbf{A}_{i+1}), v_i x, \text{B})$, with $((p_i, q_i, \mathbf{A}_i), x$ $, (p_{i+1}, q_{i+1}, \mathbf{A}_{i+1})) \in \delta$, $(p_i, q_i, \mathbf{A}_i) \notin H$ and $(p_{i+1}, q_{i+1}, \mathbf{A}_{i+1}) \notin G$. So $C_i = (xu_{i+1}, p_i, v_i, \text{B})$ and $C_{i+1} = (u_{i+1}, p_{i+1}, v_i x, \text{B})$, but B_i and B_{i+1} will depend on the sub-cases below. There are four sub-cases to consider.
  - The critical transition cannot be in δ_{N}, since Lemma 1 implies that $\mathbf{A}_i = \mathbf{A}_{v_i}^{M_2} \neq \mathbf{0}$.
  - The critical transition cannot be in $\delta_{\text{N}\to\text{B}}$, since Lemma 1 implies that $\mathbf{A}_i = \mathbf{A}_{v_i}^{M_2} \neq \mathbf{0}$.
  - Suppose the critical transition is in δ_{B}. Then $(p_i, x, p_{i+1}) \in \delta_1$ and $p_i \notin H_1$ and $p_{i+1} \notin G_1$. Therefore $C_i \vdash_{\text{B}} C_{i+1}$. Now consider B_i and B_{i+1}. Since $(p_i, q_i, \mathbf{A}_i) \notin H$ we know that $B_i = (xu_{i+1}, q_i)$. Also, we know $\mathbf{A}_{i+1} = \mathbf{A}_i \mathbf{A}_x \neq \mathbf{0}$, so $(p_{i+1}, q_{i+1}, \mathbf{A}_{i+1}) \notin H$ and $B_{i+1} = (u_{i+1}, q_{i+1})$. Finally, either $(q_i, x, q_{i+1}) \in \delta_2$, or $x = \epsilon$ and $q_i = q_{i+1}$; so in either case $B_i \vdash^* B_{i+1}$.
  - Suppose the critical transition is in $\delta_{\text{B}\to\text{N}}$. Then $x = \epsilon$ and $p_i = p_{i+1}$, so $C_i = C_{i+1}$. Also $\mathbf{A}_i \neq \mathbf{0}$, so $B_i = (u_{i+1}, q_i)$. Furthermore, $p_{i+1} \in H_1$ and $\mathbf{A}_{i+1} = \mathbf{0}$, so $B_{i+1} = (v_i \backslash u_{i+1}, q_{i+1})$. And we know that $v_i \backslash u_{i+1}$ is defined, because the configuration D_{i+1} is part of a successful run and its state $(p_{i+1}, q_{i+1}, \mathbf{A}_{i+1}) \in H$, so the step to D_{i+2} must involve matching an initial portion of the string u_{i+1} against the buffered string v_i. Finally, we also know from the definition of $\delta_{\text{B}\to\text{N}}$ that the (q_i, q_{i+1}) entry of $\mathbf{A}_i = \mathbf{A}_{v_i}^{M_2}$ is 1, so $q_{i+1} \in \delta_2^*(q_i, v_i)$. Therefore $B_i = (u_{i+1}, q_i) \vdash_{M_2}^* (v_i \backslash u_{i+1}, q_{i+1}) = B_{i+1}$.
- Suppose $D_i \vdash_{\text{B}\to\text{N}} D_{i+1}$. Then $D_i = (vu_{i+1}, (p_i, q_i, \mathbf{A}_i), v, \text{B})$ and $D_{i+1} = (u_{i+1}, (p_i, q_i, \mathbf{A}_i), \epsilon, \text{N})$, with $(p_i, q_i, \mathbf{A}_i) \in H$. Therefore $C_i = (vu_{i+1}, p_i, v, \text{B})$ and $C_{i+1} = (u_{i+1}, p_i, \epsilon, \text{N})$, and $p_i \in H_1$, so $C_i \vdash_{\text{B}\to\text{N}} C_{i+1}$. Since $(p_i, q_i, \mathbf{A}_i) \in H$, $B_i = (v \backslash vu_{i+1}, q_i) = (u_{i+1}, q_i)$. But also $B_{i+1} = (u_{i+1}, q_i)$. So $B_i = B_{i+1}$.

Therefore $C_0 \vdash_{M_1}^* C_m$, so $w \in L(M_1)$. Similarly, $B_0 \vdash_{M_2}^* B_m$, so $w \in L(M_2)$.

□

Lemma 3. *For any string w, if $w \in L(M_1)$ and $w \in L(M_2)$, then $w \in L(M_1 \cap M_2)$.*

Proof. Assume $w = x_1x_2x_3 \dots x_n \in L_1$ and $w \in L_2$, N.T.S that $w \in L_M$.
$\because w \in L_1$ and $w \in L_2$
$\therefore$ there exists a sequence of configurations $C_0, C_1, C_2 \dots C_m$ with

- $C_0 = (w, p_0, \epsilon, \text{N})$ with $p_0 \in I_1$
- $C_m = (\epsilon, p_m, \epsilon, \text{N})$ with $p_m \in F_1$
- $\forall 0 \leq i < m$, $C_i \vdash_{M_1} C_{i+1}$

and there's a function f : SUFFIX$(w) \to Q_2$ such that $f(w) \in I_2$ and $f(\epsilon) \in F_2$ and $\forall x \in \Sigma, v \in \Sigma^*$, $(f(xv), x, f(v)) \in \delta_2$.

For each $i \in \{0, \dots, m\}$, we take $C_i = (u_i, p_i, v_i, m_i)$, and define D_i to be a configuration of $M_1 \cap M_2$ as follows:

$$D_i = \begin{cases} (u_i, (p_i, f(u_i), \mathbf{0}), v_i, \text{N}) & \text{if } m_i = \text{N} \\ (u_i, (p_i, f(u_i), \mathbf{A}_{v_i}^{M_2}), v_i, \text{B}) & \text{if } m_i = \text{B} \end{cases}$$

First, notice that $D_0 = (w, (p_0, f(w), \mathbf{0}), \epsilon, \text{N})$, where $p_0 \in I_1$ and $f(w) \in I_2$, so D_0 is a valid starting configuration for a run of w on $M_1 \cap M_2$. Similarly, $D_m = (\epsilon, (p_m, f(\epsilon), \mathbf{0}), \epsilon, \text{N})$, where $p_m \in F_1$ and $f(\epsilon) \in F_2$, so D_m is a valid ending configuration for a run on $M_1 \cap M_2$. To show that $w \in L(M_1 \cap M_2)$, we will show that for each $i \in \{0, \dots, m-1\}$, $D_i \vdash_{M_1 \cap M_2}^* D_{i+1}$, which implies that $D_0 \vdash_{M_1 \cap M_2}^* D_m$.

For each $i \in \{0, \dots, m-1\}$, we know that $C_i \vdash_{M_1} C_{i+1}$, so there are four cases to consider.

- Suppose $C_i \vdash_{\text{N}} C_{i+1}$. Then $C_i = (xu_{i+1}, p_i, \epsilon, \text{N})$ and $C_{i+1} = (u_{i+1}, p_{i+1}, \epsilon, \text{N})$ where $(p_i, x, p_{i+1}) \in \delta_1$ and $p_i \notin G$ and $p_{i+1} \notin H$. Therefore $D_i = (xu_{i+1}, (p_i, f(xu_{i+1}), \mathbf{0}), \epsilon, \text{N})$ and $D_{i+1} = (u_{i+1}, (p_{i+1}, f(u_{i+1}), \mathbf{0}), \epsilon, \text{N})$, with $(f(xu_{i+1}), x, f(u_{i+1})) \in \delta_2$. So $D_i \vdash_{\text{N}} D_{i+1}$, since $(p_i, f(xu_{i+1}), \mathbf{0}) \notin G$ and $(p_{i+1}, f(u_{i+1}), \mathbf{0}) \notin H$.
- Suppose $C_i \vdash_{\text{N}\to\text{B}} C_{i+1}$. Then $C_i = (u, p, \epsilon, \text{N})$ and $C_{i+1} = (u, p, \epsilon, \text{B})$, where $p \in G_1$. Therefore $D_i = (u, (p, f(u), \mathbf{0}), \epsilon, \text{N})$ and $D_{i+1} = (u, (p, f(u), \mathbf{A}_{\epsilon}^{M_2}), \epsilon, \text{B})$, and we need to show that $D_i \vdash_{M_1 \cap M_2}^* D_{i+1}$.

  - Since $p \in G_1$, the automaton $M_1 \cap M_2$ has a transition $((p, f(u), \mathbf{0}), \epsilon, (p, f(u), \mathbf{A}_\epsilon^{M_2})) \in \delta_{\text{N}\to\text{B}}$. Therefore $D_i \vdash_{\text{N}} (u, (p, f(u), \mathbf{A}_\epsilon^{M_2}), \epsilon, \text{N})$.
  - Since $p \in G_1$, we know that $(p, f(u), \mathbf{A}_\epsilon^{M_2}) \in G$, and therefore $(u, (p, f(u), \mathbf{A}_\epsilon^{M_2}), \epsilon, \text{N}) \vdash_{\text{N}\to\text{B}} (u, (p, f(u), \mathbf{A}_\epsilon^{M_2}), \epsilon, \text{B}) = B_{i+1}$.

  Therefore $D_i \vdash^*_{M_1\cap M_2} D_{i+1}$.
- Suppose $C_i \vdash_{\text{B}} C_{i+1}$. Then $C_i = (xu_{i+1}, p_i, v_i, \text{B})$ and $C_{i+1} = (u_{i+1}, p_{i+1}, v_i x, \text{B})$, with $p_i \notin H_1$ and $p_{i+1} \notin G_1$. Therefore $D_i = (xu_{i+1}, (p_i, f(xu_{i+1}), \mathbf{A}_{v_i}^{M_2}), v_i, \text{B})$ and $D_{i+1} = (u_{i+1}, (p_{i+1}, f(u_{i+1}), \mathbf{A}_{v_i x}^{M_2}), v_i x, \text{B})$, with $(f(xu_{i+1}), x, f(u_{i+1})) \in \delta_2$. Since $p_i \notin H_1$ and $p_{i+1} \notin G_1$ and $\mathbf{A}_{v_i}^{M_2} \neq \mathbf{0}$, the automaton $M_1 \cap M_2$ has a transition $((p_i, f(xu_{i+1}), \mathbf{A}_{v_i}^{M_2}), x, (p_{i+1}, f(u_{i+1}), \mathbf{A}_{v_i}^{M_2}\mathbf{A}_x^{M_2})) \in \delta_{\text{B}}$.
- Suppose $C_i \vdash_{\text{B}\to\text{N}} C_{i+1}$. Then $C_i = (v_i u_{i+1}, p, v_i, \text{B})$ and $C_{i+1} = (u_{i+1}, p, \epsilon, \text{N})$, with $p \in H_1$. Therefore $D_i = (v_i u_{i+1}, (p, f(v_i u_{i+1}), \mathbf{A}_{v_i}^{M_2}), v_i, \text{B})$ and $D_{i+1} = (u_{i+1}, (p, f(u_{i+1}), \mathbf{0}), \epsilon, \text{N})$, with $f(u_{i+1}) \in \delta_2^*(f(v_i u_{i+1}), v_i)$. We need to show that $D_i \vdash^*_{M_1\cap M_2} D_{i+1}$.
  - Since $p \in H_1$ and the $(f(v_i u_{i+1}), f(u_{i+1}))$ entry of the matrix $\mathbf{A}_{v_i}^{M_2}$ must be 1, we know that the automaton $M_1 \cap M_2$ has a transition $((p, f(v_i u_{i+1}), \mathbf{A}_{v_i}^{M_2}), \epsilon, (p, f(u_{i+1}), \mathbf{0})) \in \delta_{\text{B}\to\text{N}}$. Therefore $D_i \vdash_{\text{B}} (v_i u_{i+1}, (p, f(u_{i+1}), \mathbf{0}), v_i, \text{B})$.
  - Since $p \in H_1$, we know that $(p, f(u_{i+1}), \mathbf{0}) \in H$, and therefore $(v_i u_{i+1}, (p, f(u_{i+1}), \mathbf{0}), v_i, \text{B}) \vdash_{\text{B}\to\text{N}} (u_{i+1}, (p, f(u_{i+1}), \mathbf{0}), \epsilon, \text{N}) = D_{i+1}$.

  Therefore $D_i \vdash^*_{M_1\cap M_2} D_{i+1}$.

Therefore $D_0 \vdash^*_{M_1\cap M_2} D_m$, i.e. $(w, (p_0, f(w), \mathbf{0}), \epsilon, \text{N}) \vdash^*_{M_1\cap M_2} (\epsilon, (p_m, f(\epsilon), \mathbf{0}), \epsilon, \text{N})$, and so $w \in L(M_1 \cap M_2)$.

□

B EQUIVALENCE OF REGULAR-COPYING EXPRESSIONS TO FSBMS

We show here that RCEs and FSBMs are equivalent in terms of expressivity: namely, the languages accepted by FSBMs are precisely the

languages denoted by RCEs. We prove this statement in two directions: 1) every RCE has a corresponding FSBM; 2) every language recognized by FSBMs can be denoted by an RCE.

Theorem 6. *Let R be a regular copying expression. Then, there exists an FSBM that recognizes $\mathscr{L}(R)$.*

Proof. We complete our proof by induction on the number of operators in R.

Base case: zero operators R must be ϵ, $\emptyset$, a for some symbol a in Σ. Then, standard method to construct corresponding FSAs, thus FSBMs, meet the requirements.

Inductive step: One or more operators In induction, we assume this theorem holds for RCEs with less than n operators with $n \geq 1$. Let R have n operators. There are two cases: 1): $R = R_1^C$; 2): $R \neq R_1^C$;

- Case 1: $R = R_1^C$. Then, we know R_1 must be a regular expression and we can construct an FSA for R_1. Assume there's an FSA $M_0 = \langle Q', \Sigma, I', F', \delta' \rangle$ that recognizes $L(R_1)$. Let $M = \langle Q, \Sigma, I, F, \delta, G, H \rangle$ with
  - $Q = Q' \cup \{q_0, q_f\}$
  - $G = I = \{q_0\}$
  - $H = F = \{q_f\}$
  - $\delta = \delta' \cup \{(q_0, \epsilon, q) \mid q \in I'\} \cup \{(q, \epsilon, q_f) \mid q \in F'\}$

  As part of this construction, we add another initial state q_0 and a final state q_f and use them as the *only* initial and final states in the new machine. We add ϵ-arcs 1) from the new initial state q_0 to the previous initial states, and 2) from the previous final states to the new final state q_f. The key component is to add the copying mechanism: G, H, and special arcs. Let G contain only the initial state q_0, which would put the machine to B mode before it takes any transitions. Let H contain only the final state q_f, which stops the machine from buffering and sends it to string matching. Thus, if w is in $L(R_1)$, ww must be in the language accepted by this complete-path FSBM and nothing beyond. Figure 21 shows such a construction. The proof showing L(M) = L(R) is suppressed here.

- Case 2: when $R \neq R_1^C$ for some R_1, we know it has to be made out of the three operations: for some R_1 and R_2, $R = R_1 + R_2$,

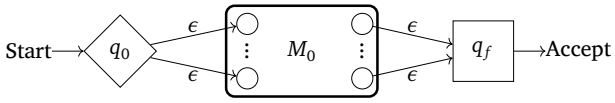


Figure 21: *The construction used in converting the copy expression R_1^C to a finite-state buffered machine. $L(M_0) = L(R_1)$.*

or $R = R_1R_2$ or $R = R_1^*$. Because R_1 and R_2 have operators less than i, from the induction hypothesis, we can construct FSBMs for R_1 and R_2 respectively. Using the constructions mentioned in the main text, we can construct the new FSBM for R.

□

Theorem 7. *If a language L is recognized by an FSBM, then L could be denoted by a RCE.*

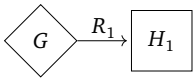
Instead of diving into proof details, we introduce the most crucial fragments to the full FSBM-to-RCE conversion: how the copying mechanism in a complete-path FSBM is converted into a copy expression. We leave out parts that use basic ideas of FSA-to-RE conversion, which can be found in Hopcroft and Ullman (1979, pp. 33–34).

The previous discussion on the realization of the copying mechanism in complete-path FSBMs concluded with three aspects 1) the specification of G states, 2) the specification of H states, and 3) the *completeness restriction* which imposes ordering requirements on G and H. Thus, to start with, we want to concentrate on the areas selected by G states and H states in a machine, as they are closely related to the copying mechanism.

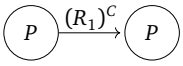
The core is to treat any G state and H state pair as an small FSA: if the paths along the pair do not cross other special states, borrow the FSA-to-RE conversion to get a regular expression R_1, denoting the languages possible to be stored in the buffer temporarily. Importantly, there are only finitely many (G, H) pairs. Iterating through all possible paths between these two states and getting a general RE R_1 by union, we use two plain states with the RCE R_1 along the arc to denote the languages from that specific G to H. Then we plug them back into the starting FSBM.

All special states are eliminated. Thus, we get an intermediate representation with only plain states. Similar ideas as FSA-to-RE conver-

sion could be applied again to get the final regular copying expression for this FSBM. The described conversion of the copying mechanism in a machine to a copy expression is depicted in Figure 22.



(a) *Goal for the possible* (G, H) *in the first steps of the FSBM-to-RCE conversion*



(b) *Next step after Figure 22a*

Figure 22: *The conversion of the copying mechanism in an FSBM to a corresponding RCE. P represents the plain, non-H, non-G states*

Yang Wang
0000-0003-0575-2626
yangwangx@g.ucla.edu

Department of Linguistics
University of California, Los Angeles
Los Angeles, CA, USA

Tim Hunter
0000-0003-1587-9049
timhunter@ucla.edu

Department of Linguistics
University of California, Los Angeles
Los Angeles, CA, USA

Yang Wang and Tim Hunter (1970), *On regular languages*, Journal of Language Modelling, $i^2(e^{i\pi})$:1–67
https://dx.doi.org/10.15398/jlm.vi^2i$e^{i\pi}$.342