Regular languages extended with reduplication: formal models, proofs and illustrations

Yang Wang · June 28, 2021

Abstract

Total reduplication is common in natural language phonology and morphology. However, productive total reduplication requires computational power beyond context-free, while other phonological and morphological patterns are regular, or even sub-regular. Thus, existing language classes characterizing reduplicated strings inevitably include typologically unattested context-free patterns, such as reversals. This thesis introduces two ways of extending regular languages to incorporate reduplication. Firstly, we add copying as an expression operator and define regular copying expressions (RCEs) in a more restricted way. Secondly, we augment finite-state machinery with the ability to recognize copied strings and introduce a new computational device: finite-state buffered machine (FSBMs). As a result, the class of regular languages and languages derived from them through a primitive copying operation is characterized, named *regular+copying* languages (RCLs). We then examine and discuss the closure properties of this language class. As suggested by previous literature (Gazdar and Pullum, 1985, 278), *regular+copying* languages should approach the correct characterization of natural language word sets.

Keywords: Total reduplication, Unbounded Copying, Formal Language Theory, Finite State Buffered Machines, Regular Copying Expressions

Contents

1 Introduction **7**

1.1 Background 9

1.1.1 The puzzle of reduplication 9

1.1.2 Previous computational work on reduplication 14

1.2 Main questions addressed in this thesis 16

1.3 The organization of this thesis 18

2 Defining regular + copying languages **19**

2.1 Notations: regular languages 19

2.2 Regular Copying Expressions 22

2.2.1 Closure: homomorphism 23

2.3 Finite-State Buffered Machine 27

2.3.1 Finite-state buffered machines in a nutshell 27

2.3.2 Mathematical definitions and examples 29

2.3.3 Implementation of the copying mechanism and complete-path FSBMs . . 34

2.3.4 The equivalence between general FSBMs and complete-path FSBMs 38

2.3.5 Closure properties of complete-path FSBMs 39

2.4 The equivalence between RCEs and FSBMs 48

2.5 Regular + copying languages 53

3 The linguistic relevance of the formal methods **54**

4 Discussion **57**

4.1 Typology of reduplication 57

4.1.1 Non-local Reduplication 57

4.1.2 Multiple Reduplication 58

4.1.3 Reduplication with non-identical copies 59

5 Conclusion **61**

A Mathematical preliminaries **70**

B A proof of Theorem 2: closure under the intersection with regular languages **72**

C A proof of Theorem 5: the construction for the copying expression operator **81**

List of Figures

1 The four-level Chomsky Hierarchy 7

2 A finite-state machine for whole-base copying with the set of bases = {aaa, aba, aab, abb, baa, bba, bab, bbb} 12

3 Crossing dependencies in Dyirbal total reduplication *'midi-midi'* (top) versus nesting dependencies in unattested string reversal *'midi-idim'* (bottom) 13

4 The class of regular with copying languages in the Chomsky Hierarchy. The red oval shape represents the regular+copying languages. 17

5 Mode changes and input-buffer interaction of an FSBM M on "…abbababbab…". Assume M is armed with sufficient input consuming and symbol matching apparatus. The machine switches to B mode to temporarily store symbols in queue-like buffer. At the breaking point, it shifts to E mode for symbol matching between what's in the buffer and what's in the input. After all symbols matched, the buffer is emptied and the machine further switches to N mode. 28

6 An FSBM M_1 with G = $\{q_1\}$ (diamond) and H = $\{q_3\}$ (square); dashed arcs are used only for the emptying process. $L(M_1) = \{ww|w \in \{a,b\}^*\}$ 30

7 One example FSBM and the corresponding FSA for the base language 31

8 M_2 in Figure 7a accepts *abbabb* 31

9 M_2 in Figure 7a rejects *ababb* 32

10 An FSBM M_3 for Agta CVC-reduplicated plurals: G = $\{q_1\}$ and H = $\{q_4\}$ 32

11 M_2 in Figure 10 accepts *taktakki* 33

12 M_2 in Figure 10 rejects *tiktakki* 33

13 The template for the implementation of the copying in FSBMs. Key components: G state, H states, transitions between H states, and strict ordering between G and H. Solid lines represent a transition in one step. Dotted lines represent a sequence of normal transitions. Black dotted lines replace plain non-G non-H states. H states in between H states are replaced by red dashed lines. 35

14 Possible paths in a machine failing on the completeness requirement. Dotted lines represent a sequence of normal transitions. Dashed lines are special transitions between H states in one step. 37

15 An incomplete FSBM M_4 with $G = \varnothing$ and $H = \{q_2, q_4\}$; $L(M_4) = \{abba\}$ 37

16 An FSA (or an FSBM with $G = \varnothing$ and $H = \varnothing$) whose language is equivalent as M_3 in Figure 15 38

17 Example for the intersection construction 40

18 Constructions used for the homomorphic language in Theorem 3. 42

19 Under-generation of the conventional construction of the inverse homomorphic image 43

20 The construction used in the union of two FSBMs 44

21 The construction used in the concatenation of two FSBMs 45

22 Problems arise in the concatenation of two incomplete paths. Dotted lines represent a sequence of normal transitions. Red dashed lines represent a sequence of special transitions 46

23 The construction used in the star operation 47

24 FSBMs for the base step in Theorem 5. All have $G = \varnothing; H = \varnothing$ 48

25 The construction used in converting the copy expression R_1^C to a finite state buffered machine. $L(M_0) = L(R_1)$. 49

26 The conversion of the copying mechanism in an FSBM to RCE. P represents the plain, non-H, non-G states 51

27 The example conversion of the copying mechanism to a copy expression 52

28 The linguistic relevance of the closure under homomorphism 56

List of Tables

1 Total reduplication:Dyirbal plurals (top); partial reduplication:Agta plurals (bottom). 9
2 Reduplication and bounded/unbounded copying. “Boundedness” here indicates whether the corresponding language is regular. 12
3 Different cases for G and H states along a path 38
4 Surveyed closure properties of RCLs 53
5 Chukchee absolutive singular: copies the first CVC sequence to the end of the word 57
6 Multiple reduplication in Thompson 58
7 Non-identical copies in Javanese 60

1 Introduction

Formal language theory (FLT) provides computational mechanisms characterizing different classes of abstract languages. Pursuing formal language theory in the study of human languages, researchers would hope to discover a hierarchy of grammar formalisms that matches empirical findings on human languages: more complex languages in such a hierarchy are supposed to be 1) less common in natural language typology; 2) harder for learners to learn and/or 3) harder for speakers to process.

The classical Chomsky Hierarchy (CH) arranges formal languages in four levels with increasing complexity: regular, context-free, context-sensitive, recursively enumerable (Chomsky, 1956, 1959; Jäger and Rogers, 2012). This relation is visualized in Figure 1. Essentially, every regular language is a context-free language, and every context-free language is a context-sensitive language, and so on. An example shows that not all context-free languages are regular is the string reversal language ww^{R}. Similarly, some languages are context-sensitive or recursively enumerable but not context-free. Those patterns sitting higher in the CH are regarded as more *complex* than those in a lower position, since they demand more computational resources: the automata/grammars fully capturing the more complex patterns are more powerful and less restricted.

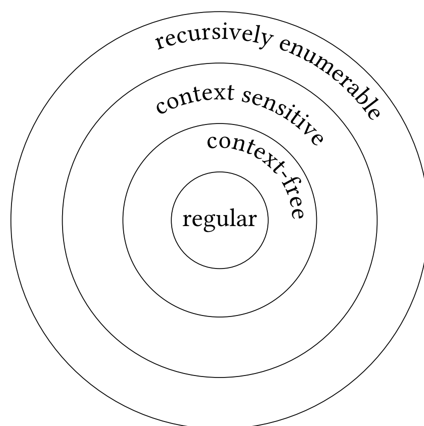


Figure 1: *The four-level Chomsky Hierarchy*

Does the CH notion of formal complexity have the desired empirical correlates in natural languages? Several findings suggest that those four levels do not align with natural languages

precisely, some leading to major refinements of the Chomsky Hierarchy. First, the unbounded crossing dependencies in Swiss-German case marking (Shieber, 1985) facilitated various attempts to characterize mildly context-sensitive languages (MCS), which extend context-free languages (CFLs) but still preserve some useful properties of CFLs (e.g., Joshi, 1985; Seki et al., 1991; Steedman, 1996; Stabler, 1997). Secondly, it is generally accepted that phonology, with the exception of reduplication which is the focus of this thesis, is regular (e.g. Johnson, 1972; Kaplan and Kay, 1994). However, being regular is argued to be an insufficiently restrictive property for phonological knowledge of well-formedness, because some generalizations falling into the regular set are typologically unattested. For example, hardly any languages have their words sensitive to an even or odd number of certain sounds in a natural class (Heinz, 2018). With typological evidence, the sub-regular hierarchy (e.g McNaughton and Papert, 1971; Simon, 1975) was further developed, which continues to be an active area of research (e.g. Heinz, 2007; Heinz et al., 2011; Chandlee, 2014; Graf, 2017; Heinz, 2018).[1]

This thesis analyzes another mismatch between existing well-known language classes and empirical findings: reduplication, which involves copying operations (Inkelas and Zoll, 2005) on certain base forms. The reduplicated phonological strings are either of total identity (*total reduplication*) or of partial identity (*partial reduplication*) to the base forms. Table 1 provides examples showing the difference between total reduplication and partial reduplication. In Dyirbal, the pluralization of nominals is realized by fully copying the singular stems, while in Agta examples, plural forms only copy the first CVC sequence of the corresponding singular forms (Healey, 1960; Marantz, 1982). [2]

[1]One might wonder why this thesis adds copying to the regular set but not a more restricted set. Firstly, the *strictly local* (SL), the *strictly piecewise* (SP) and the *tier-based strictly local* classes are all reasonable candidates. Graf (2017) further supplements domain restrictions to strictly piecewise grammars and finds the resulting *interval-based strictly piecewise* languages contain SL, SP, and TSL languages. Moreover, Jardine (2016) argues tonal patterns as mappings are fully regular processes while segmental patterns are maximally weakly deterministic and thus less complex. However, McCollum et al. (2020) challenges this typological asymmetry by advancing that segmental processes require the same expressivity as tonal processes, which has further implications on the sub-regular hypothesis. As the debate continues, for now, we remain agnostic about which set should be the most restrictive one to add the copying operation. Secondly, the regular set is one of the most well-studied classes with practically useful grammars/automata. Thus, starting with the regular set is an important and reasonable step to pursue.

[2]Here, we adopt a simplistic analysis. 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.

Total reduplication: Dyirbal plurals (Dixon, 1972, 242; Inkelas, 2008,352)			
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, 1960,7)			
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).*

1.1 Background

1.1.1 The puzzle of reduplication

Reduplication in natural languages Enough evidence supports the productivity of reduplication patterns in natural languages. Waksler (1999) presented code-switching data from Tagalog, as in Example 1. Zuraw (1996, 9) also provided some examples of extending Tagalog CV-reduplication to novel English forms, such as *thank you* and the form [mag-θɛ-θæŋkju]. Clearly, Tagalog speakers could apply verb reduplication to novel forms.

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

Reduplication is common cross-linguistically. In the sample reported by Rubino (2013) and 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. As a comparison, context-free string reversals are rare in phonology and morphology (Marantz, 1982) and appear to be confined to language games (Bagemihl, 1989), whose status of phonology is unclear.

Additionally, in a few experiments that, either directly or indirectly, study the learnability of identity-based patterns, reduplication appears to be prominent and easy to learn. Past empirical

studies (Marcus et al., 1999, 2007) show infants can extract rules from reduplicated patterns. One recent artificial grammar learning study (Moreton et al., 2021) directly compares the learning of reduplication with the learning of a syllable-level reversal. In two experiments, adult learners were trained to recognize either a reduplication or a syllable reversal pattern. In the first experiment, participants did not know the rules and conditions beforehand. They were required to induce the generalizations by themselves and apply what they learned. Participants in the reduplication group showed above-chance performance whether they could state the rule or not. However, for the syllable-reversal condition, only participants who could correctly state the rule at the end showed above-chance performance, which hints the explicit reasoning in learning reversals. In addition, those participants who learned the reversal pattern showed a longer reaction time, suggesting more non-automatic computation involved. In the second experiment, participants were informed of the generalization they should apply. Participants in the reduplication group still showed better performance. These results support the hypothesis that the reversal pattern is more complex than the reduplication pattern and requires more effortful computation. To the extent that reversal was learned, it was achieved by a more explicit rule rather than unconscious linguistic knowledge.

An interesting aspect of this AGL study is that the stimuli used were auditory, *purely phonological*, *meaningless* strings, chunks of which are identical. This seems to suggest that reduplication or reduplication-like patterns are not all morphologically relevant. Moreover, Zuraw (2002) proposes *aggressive reduplication* in phonology: speakers are sensitive to adjacent internal phonological similarity within words, and reduplication-like structures are attributed to those words. Direct evidence supporting aggressive reduplication comes from some pseudo-reduplication patterns. Pseudo-reduplicated words have one portion (the pseudo-reduplicant) identical to another portion (the pseudo-base). However, the pseudo-base does not bear proper morphosyntactic or semantic information and cannot stand alone. Zuraw (2002) studied Tagalog loanwords from English and Spanish and used them for illustrations. Tagalog mid vowel raising does not apply when a preceding mid vowel is present and the hypothesized motivation is to preserve sub-string similarity. For

example, the stem-ultima vowel /o/ is realized as [u] in [kal**u**s-in] *'to use a grain leveller on'* but [o] in [tod**o**-in] *'to include all'*.

It has to be admitted that there are too few relevant processing studies to provide solid conclusions on the complexity of reduplication through the lens of online cognitive activity. However, the prevalence of reduplication in typology and learning results in experimental settings converge to one point: in an ideal formal complexity hierarchy that matches empirical findings, reduplication ought to be less complex than patterns exhibiting nesting dependencies, such as string reversals.

Reduplication in the Chomsky Hierarchy This subsection aims to show that while most of phonological and morphological patterns are regular, reduplication is not. It starts by investigating the computational issue that arises by assuming reduplication is fully productive and using *unbounded copying* as a model.

What is unbounded copying? The term *unbounded* means 'no upper bound'. Unbounded copying, therefore, is copying without upper bounds on reduplicants. On the other hand, bounded copying, imposing a limit on the number of segments copied, is proved to be regular and can be captured by finite-state machinery (Chandlee and Heinz, 2012). As for their empirical coverage, there is no doubt that bounded copying covers the cases of partial reduplication in natural languages. The situation for total reduplication is more nuanced. Some previous studies use "unbounded copying" interchangeably with *total reduplication*, because total reduplication copies the whole forms of certain bases, which might lead people to easily equate total reduplication to copying with no limits on the length of the reduplicants. However, some researchers (e.g. Clark and Yoshinaka, 2014; Chandlee, 2017) differentiated unrestricted, productive total reduplication from total reduplication on a *finite* set of bases in terms of their computational nature. Formally, for $\{ww \mid w \in L\}$, the distinction is whether L is finite or infinite. When reduplication is only applied to a finite set of lexemes, whether partial or total, the resulting set of reduplicated forms is always *finite*, and therefore technically within *regular*. In the total reduplicated cases, even

though the whole bases are copied, if the set of bases L is finite, the length of the longest string could still be an upper bound, which invalidates unproductive total reduplication as unbounded copying. Modelers therefore can take the easy way out by forcing the reduplicated pattern into the regular class. More specifically, one can construct a 1-way finite-state automata/transducer with additional states and memorized arcs and represent the memorization of each corresponding reduplicated form. For example, in Figure 2, by full listing, the finite state machine recognizes $\{ww \mid w \in \mathrm{L}\}$ with a finite $\mathrm{L} = \{aaa, aba, aab, abb, baa, bba, bab, bbb\}$.

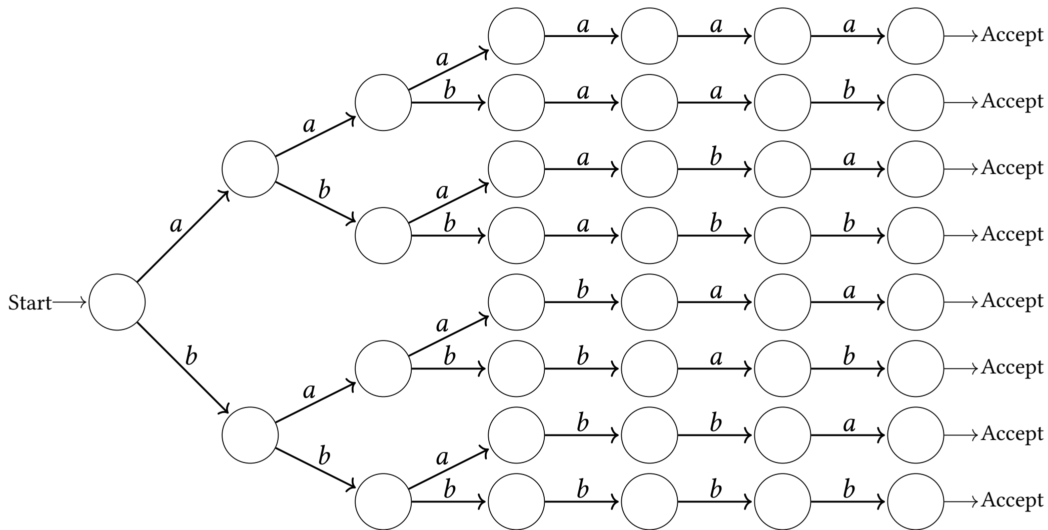


Figure 2: *A finite-state machine for whole-base copying with the set of bases = {aaa, aba, aab, abb, baa, bba, bab, bbb}*

The feasibility of such construction indeed reveals computational difference between total reduplication on finite bases versus productive total reduplication, demonstrated in Table 2. Such

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

Table 2: *Reduplication and bounded/unbounded copying. "Boundedness" here indicates whether the corresponding language is regular.*

distinction is more form a mathematical, definition-wise perspective. But it uncovers the core aspect of two treatments and raises a question about the nature of reduplication: whether we should employ a *memorization* account of copying or a fully productive copying operation, which

itself does not impose any number restriction on the set of bases?

As we discussed earlier, natural languages exhibit productive reduplication patterns. Additionally, from a computational angle, Roark and Sproat (2007) and further Dolatian and Heinz (2020) challenge the memorization account. They argue 1-way finite-state approach not only is incapable of capturing productive total reduplication , but also would lead to a great increase in the number of states or state explosions for bounded copying. Indeed, one can directly see this from the unwieldy machine aiming to recognize the simple ww language with only eight possible bases of length three in Figure 2. Following them, this thesis focuses on productive total reduplication and unbounded copying.

Unbounded copying in the Chomsky Hierarchy The simplest yet structurally dull language representing unbounded copying is $\mathrm{L}_{ww} = \{ww \mid w \in \Sigma^*\}$, with any arbitrary strings made out of an alphabet Σ as candidates of bases. L_{ww} is a well-known non-context free language (Hopcroft and Ullman, 1979). Its non-context-freeness comes from the incurred cross-serial dependencies among segments, similar to Swiss-German case marking constructions as $\{a^i b^j c^i d^j \mid i, j \geq 1\}$. For instance, in $\mathrm{L} = \{a^i b^j a^i b^j \mid i, j \geq 1\}$, a subset of L_{ww}, as always precede bs in both halves. Then, the first i many of as must correspond with the second i many of as segment by segment, while the first j many of bs must correspond with the second j many of bs segment by segment. Natural languages do possess patterns similar to L, such as Noun *o* Noun construction in Bambara (Culy, 1985). However, the typologically-rare string reversal ww^{R} is an instance of nesting dependencies, which are context-free. Figure 3 is an illustration of cross-serial dependency of Dyirbal plurals 'midi-midi' and nesting dependency of made-up string reversal 'midi-idim'.

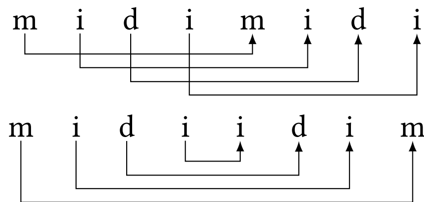


Figure 3: *Crossing dependencies in Dyirbal total reduplication 'midi-midi' (top) versus nesting dependencies in unattested string reversal 'midi-idim' (bottom)*

Syntactic patterns show nesting dependencies at least as often as crossing dependencies. Mildly context sensitive formalisms, which were argued to be adequate for natural language syntax (Joshi, 1985; Stabler, 2004), can describe L_{ww}. Joshi et al. (1990, 13) provides a tree adjoining grammar for this language. A minimalist grammar can be found in Graf (2013, 119). Multiple context-free grammars (MCF) are used to implement reduplication in Primitive Optimality Theory according to the base-reduplicant correspondence theory (Albro, 2000). A parallel multiple context free grammar (PMCFG) for L_{ww} is in Clark and Yoshinaka (2014, 13). However, the class of languages recognized by those computationally powerful formalisms would include all context-free sets, thus inevitably including typologically unattested patterns, such as reversals, as described earlier.

To summarize, reduplication is productive, common cross-linguistically and easy to learn. However, in the existing hierarchy of formal languages, it belongs to a language class higher than the context-free set and requires at least the power of mildly context-sensitive languages to recognize even the simplest copy language L_{ww}. The extended formal power would include phonologically and morphologically unattested nesting dependencies, such as string reversal patterns.

1.1.2 Previous computational work on reduplication

This section reviews previous computational works which model reduplication but not nesting dependencies. There are essentially two lines. The first is to develop better designs than the memorization account but still limit copying to only a *finite set of bases.* The second is to extend only a bit beyond conventional finite-state technology and intentionally exclude non-reduplicative processes.

Previously, we discussed how modeling unproductive total reduplication could be achieved by the bulky full listing account. Many approaches from the first direction aim to reduce redundancy with more linguistically sound and computationally efficient finite-state techniques. However, these approaches do not add formal power and cannot model productive total reduplication. One example is finite-state registered machines in Cohen-Sygal and Wintner (2006) (FSRAs) augmented

with finitely many registers as its memory. Even though it brings extensions to standard finite-state machines, it does not add any computational power to them. The important equivalence between the augmented machine and standard finite-state machines is guaranteed by the finite number of registers. Besides this, some other examples are the *compile-replace* algorithm in Beesley and Karttunen (2000), the intersection of a reduplication finite-state machine and the enriched lexical representations with different types of transitions: *repeat, skip* and *self-loops* in Walther (2000) and the EQ function in Hulden (2009).

The state-of-art finite-state method that computes *unbounded copying* elegantly and adequately is 2-way finite-state transducers (2-way FSTs) (Dolatian and Heinz, 2018a,b, 2019, 2020), which use the similar idea as conventional 1-way FSTs but can move back and forth on the input. 2-way FSTs capture reduplication as a string-to-string mapping ($w \rightarrow ww$). To avoid the mirror image function ($w \rightarrow ww^{R}$), Dolatian and Heinz (2020) further developed sub-classes of 2-way FSTs which cannot output anything during right-to-left passes over the input (cf. rotating transducers: Baschenis et al., 2017).

It should be noted that the issue addressed by 2-way FSTs is a different one from the topic of this thesis: reduplication is modeled as a function ($w \rightarrow ww$), while this thesis aims to characterize a set of strings containing the copy languages with identical sub-strings (ww). The string set question and the corresponding membership problem are non-trivial and difficult ones for reasons of both formal aspects and the linguistic motivation. Firstly, since 2-way FSTs are not readily invertible, the inverse relation $ww \rightarrow w$ for the morphological analysis problem remains an open question, as acknowledged in Dolatian and Heinz (2020). Although we do not directly address the morphological analysis problem, recognizing the reduplicated ww strings is the important first step. To determine if a string $x = ww$ can be mapped to w seems to require at least *recognizing* whether x belongs to the ww language. Secondly, given that 2-way finite-state transducers extend the expressive power of 1-way finite-state transducers, one might be tempted to look at languages recognized by 2-way finite-state *automata* for the non-regular copy languages. However, any 2-way finite-state automaton is equivalent to some 1-way finite-state automaton, proved independently

in Rabin and Scott (1959) and Shepherdson (1959). In other words, any languages recognized by a 2-way finite state *automaton* are in the regular set. For recognizing copy languages, there is no direct help from looking at the closely related 2-way finite state techniques.

As for the theoretical aspects, as discussed before, there are evidences supporting meaning-free, non-morphologically-generated reduplication patterns. *If* the phonology module ever needs constraints requiring phonological strings of certain forms to be reduplicated, or *if* the phonological grammar needs a way to construct segmental correspondence between bases and reduplicants for surface reduplicated strings and/or sub-string correspondence by Redup in the aggressive reduplication account, such tasks could be achieved computationally by models with the ability to recognize the copy languages. Whether those conditions hold is beyond the scope of this thesis. Overall, it would be desirable to have models that detect sub-string identity from surface strings whose bases are in the regular set. Of course, more powerful grammars, such as minimalist grammars and multiple context-free grammars, can help and achieve these goals. But they appear to be much too powerful for this purpose.

1.2 Main questions addressed in this thesis

Given that most phonological and morphological patterns are regular, how can one fit in reduplicated strings without including reversals? At the same time, how can one rule out the non-reduplicative, Swiss-German type of crossing dependencies, which appear to be unattested in phonology? Gazdar and Pullum (1985) made the remark that

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

There are three levels of questions raised in this quote:

- How can a characterization of such class of languages be given?

- After this language class is proposed, what languages are there in this class? What computational properties does this class possess?

- How does this language class align with natural language patterns?

This thesis aims to provide answers to those questions. It examines what minimal changes can be brought to regular languages to include stringsets with two copies of the same sub-strings, while excluding some typologically unattested context-free patterns, such as reversals, and other crossing dependencies other than reduplication. We name the class of languages with regular languages and added copied stringsets as *regular + copying languages* (RCLs). The intended relation of this language class to other existing language classes is shown in Fig. 4.

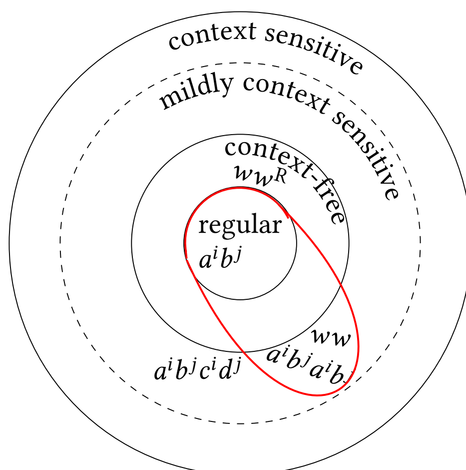


Figure 4: *The class of regular with copying languages in the Chomsky Hierarchy. The red oval shape represents the regular+copying languages.*

Formally, regular+copying languages are defined in two ways in this thesis. First, it adds a copying expression operator to the existing collection of operations whose closure defines the regular languages (namely union, concatenation and Kleene star). The set of regular +copying languages is the closure under this new expanded set of operations. Second, this thesis introduces a new computational device: finite-state buffered machine (FSBMs). These are two-taped finite state automata, with the ability to store symbols from the input tape into an unbounded memory

buffer and compare them to later input symbols, hence able to detect identity between sub-strings. It turns out that these two formal approaches yield the same class of languages. After introducing regular+copying languages, we inspect the corresponding closure properties and conclude with a discussion on their performance in modeling natural language reduplication.

1.3 The organization of this thesis

- Section 2 first gives an overview of the notations adopted in this thesis. Then, it gives definitions and examples of regular copying expressions and finite-state buffered machines and shows their equivalence. The last subsection defines regular+copying languages and scrutinizes the closure properties of this language class.
- Section 3 shows the linguistic relevance of the formal methods and properties.
- Section 4 examines (in)capabilities of the current formal models on natural language reduplication.
- Section 5 concludes the thesis and proposes directions for future research.

2 Defining regular + copying languages

Section 2.1 reviews basic mathematical definitions relevant in defining regular + copying languages. We focus on existing characterizations of the regular class, because regular+copying languages arise from modifications of these characterizations. The rest of this section proposes new expressions and a new computing device: the finite-state buffered machines (FSBMs) that extend finite-state machines.

2.1 Notations: regular languages

We provide a brief overview of the most fundamental mathematical concepts necessary in understanding the subsequent sections, namely the relevant notions in the regular class. Basic knowledge by the reader (sets, relations and functions, alphabets and definitions for formal languages) is assumed. Readers not familiar with those terms can first read through Appendix A, or consult other useful sources such as Hopcroft and Ullman (1979) and Sipser (2013).

The notations are largely standard ones and modifications are emphasized. We tried to keep the notations consistent across the whole paper.

Language classes A *class of languages* is a set of languages. Probing the computational properties of different language classes can be achieved by analyzing different classes of *grammars* or *automata* that represent a possibly infinite language in finite means. Specifically, understanding the operations, the mechanisms and computational resources available in a corresponding class of grammars is instructive to identify the characteristics of languages within a class.

Regular languages Regular languages are sets of strings recognized by finite state automata (FSAs) and described by regular expressions, whose definitions are provided below. The limited amount of memory used in an FSA informs us of the constant bound on memory for computing any regular sets. Meanwhile, regular expressions offer a "bottom-up" view and allow one to see how a language is decided by the closure of certain operations on an alphabet.

Definition 1. *A finite-state machine (FSA) is a 5-tuple* $\langle Q, \Sigma, \delta, I, F \rangle$ *where*

1. *Q is a finite set of states*
2. Σ *is a finite alphabet*
3. $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ *is the transition relation.*
4. $I \subseteq Q$ *is a set of initial states*
5. $F \subseteq Q$ *is a set of final states*

Definition 1. defines a non-deterministic finite automata (NFAs). Moreover, it extends the standard definition of NFA to have a set of initial states but not just one initial state. However, such extension does not bring more power to the machine, as it is easy to construct a corresponding FSA with only one initial state ϵ-branching into the previous initial states.

Definition 2. *A* ***configuration*** *of a finite state machine* $C = (w, q) \in \Sigma^* \times Q$ *where* w *is the input string and* q *is the current state the machine is in.*

Definition 3. *Given an FSA* A*, for any* $x \in (\Sigma \cup \{\epsilon\}), w \in \Sigma^*$ *and* $q_1, q_2 \in Q$*, we say a configuration* (xw, q_1) ***yields*** *a configuration* (w, q_2) *if and only if* $(q_1, x, q_2) \in \delta$*, denoted* $(xw, q_1) \vdash_A (w, q_2)$*.*

Definition 4. *A* ***run*** *of an FSA* A *on the input* w *is a sequence of configurations* $C_0, C_1, \ldots C_n$ *such that 1)* $\exists q_0 \in I, C_0 = (q_0, w)$*; 2)* $\exists q_f \in F, C_n = (q_f, \epsilon)$ *and* $\forall 0 \leq i < n, C_i \vdash_A C_{i+1}$*.* A *accepts* w *iff there is a run on* A *of* w*. The language recognized by an FSA* A*, denoted by* $L(A)$*, is the set of strings over* Σ^* *accepted by* A*.*

Now, we switch to the definition of regular expressions by starting with regular operations on given languages. For any two languages L_1 and L_2, *union*, *concatenation* and *star* are three regular operations. The union is $L_1 \cup L_2 = \{u \mid u \in L_1 \text{ or } u \in L_2\}$. The concatenation is $L_1 \circ L_2 = \{uv \mid u \in L_1 \text{ and } v \in L_2\}$. The Kleene star operation on a language L_1 is defined as $L_1^* = \{x_1 x_2 \ldots x_k \mid k \geq 0 \text{ and each } x_i \in L_1\}$ with $\epsilon \in L_1^*$.

Definition 5. *Let Σ be an alphabet. The regular expression (RE) over Σ and the languages they denote are defined as follows.*

- $\varnothing$ *is a regular expression and* $\mathcal{L}(\varnothing) = \varnothing$ *the null set*
- ϵ *is a regular expression and* $\mathcal{L}(\epsilon) = \{\epsilon\}$ *the null string*
- $\forall a \in \Sigma$, a *is a regular expression and* $\mathcal{L}(a) = \{a\}$
- *If* R_1 *and* R_2 *are regular expressions,* $R_1{+}R_2$, R_1R_2, $R_1{}^*$ *are regular expressions such that* $\mathcal{L}(R_1{+}R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$; $\mathcal{L}(R_1R_2) = \mathcal{L}(R_1) \circ \mathcal{L}(R_2)$; $\mathcal{L}(R_1{}^*)$ *is the Kleene star on* $\mathcal{L}(\mathrm{R}_1)$.

For example, the value of $(\mathtt{ab+c})^*$ is the language $\{\epsilon, ab, c, cab, abc, abab, cc \ldots\}$, which contains all possible strings by attaching any number of *ab* or *c*.

The languages accepted by finite state machines are precisely the languages denoted by regular expressions, since there always exists an FSA that accepts the language denoted by any arbitrary regular expressions, and vice versa (see Hopcroft and Ullman, 1979, 29-34).

Besides the three regular operations, a generalized regular expression (GRE) also includes $\mathrm{R}_1 \times \mathrm{R}_2$ and $\overline{\mathrm{R}_1}$ in its recursive definition, describing the intersection and complement operations respectively. Specifically, $\mathcal{L}(\mathrm{R}_1 \times \mathrm{R}_2) = \mathcal{L}(\mathrm{R}_1) \cap \mathcal{L}(\mathrm{R}_2)$ and $\mathcal{L}(\overline{\mathrm{R}_1}) = \Sigma^* - \mathcal{L}(\mathrm{R}_1)$. Because regular languages are closed under intersection and complementation, generalized regular expressions are equivalent to standard regular expressions in terms of their expressivity.

To conclude, for any $\mathrm{L} \in \Sigma^*$, L is regular iff $\exists$ an FSA M such that $\mathcal{L}(\mathrm{M}) = \mathrm{L}$, iff $\exists$ an RE r such that $\mathcal{L}(r) = \mathrm{L}$, iff $\exists$ a GRE r' such that $\mathcal{L}(r') = \mathrm{L}$.

For incorporating reduplication, we will introduce regular copying expressions (RCEs) below by extending the standard definition of REs. However, proving the equivalence between the new computing device and regular copying expressions, we will take advantage of generalized regular expressions, especially the expression denoting the intersection operation. Details can be found in the following sections.

2.2 Regular Copying Expressions

The goal is to add copying to the set of operations whose closure defines regular languages and examine the consequences of such addition. The method of treating copying as an expression operator, adopted here, is not initiated in a vacuum. Several works touched on a similar idea but restricted themselves to only copying finite languages. The 'compile-replace' system in Beesley and Karttunen (2000, 2003), intending to construct finite-state machinery to generate reduplicated forms, uses a copying-like expression as the intermediate step. In detail, it takes a lexical item w in a language L and creates an intermediate expression of $\{w\}$^2 (more generally, $\{w\}$^n), denoting a fixed number of concatenations of w with itself. Then, the system compiles this expression in run-time and outputs the surface string form ww (more generally, $ww \dots w$ with n many of ws). Importantly, the system only operates on a *finite* L and thus targets bounded copying. Similarly, Hulden (2009), seeking to model bounded copying in natural languages in finite-state machinery, mentions a regular expression operator *Copy(L)*, which takes a *finite* language L and outputs another *finite* language $L' = ww$ when w is a string in L. [3]

To describe languages involving unbounded copying and to examine their mathematical properties, a regular copying expression incorporates a copying operation on some infinite language, or precisely, regular languages. Definitions are as follows.

Definition 6. *Let Σ be an alphabet. The regular copying expression (RCE) over Σ and the languages they denote are defined as follows.*

- *$\varnothing$ is a regular copying expression and $\mathcal{L}(\varnothing) = \varnothing$*
- *ϵ is a regular copying expression and $\mathcal{L}(\epsilon) = \{\epsilon\}$*

[3]However, Hulden (2009) noticed that in natural languages, phonological changes complicate the issue by introducing complex cases such as the partial prosodically governed reduplication found in Warlpiri. To ultimately capture natural language reduplication elegantly, Hulden (2009) did not further pursue the COPY operator but designed an EQ operator into finite-state calculus, which checks whether certain types of sub-strings are equal in content. One of the main goals of this thesis is to analyze the *formal properties* of the class of languages which involves copying. Thus, viewing reduplication "by way of string copying" suffices for this purpose given it provides the desired class of languages. Finite-state buffered machines introduced in the next section check sub-string identity and thus could potentially be extended to model phonological complications as the next step.

- *$\forall a \in \Sigma$, a is a regular copying expression and $\mathcal{L}(a) = \{a\}$*
- *If R_1 and R_2 are regular copying expressions, R_1+R_2, R_1R_2, $R_1{}^*$ are regular copying expressions such that $\mathcal{L}(R_1+R_2) = \mathcal{L}(R_1) \cup \mathcal{L}(R_2)$; $\mathcal{L}(R_1R_2) = \mathcal{L}(R_1) \circ \mathcal{L}(R_2)$; $\mathcal{L}(R_1{}^*) = (\mathcal{L}(R_1))^*$.*
- *(new copying operator) If R_1 is a* ***regular*** *expression, R_1^C is a regular copying expression and $\mathcal{L}(R_1^C) = \{ww \mid w \in \mathrm{L}(\mathrm{R}_1)\}$*

Regular copying expressions introduce two modifications to regular expressions. Firstly, a R^{C} expression operator for the copying-derived language is added. Then, the closure of other recursive operations is extended to all regular copying expressions. Therefore, languages denoted by regular copying expressions are closed under concatenation, union and Kleene star. Secondly, the copying operation is only granted access to regular expressions, namely to regular sets without copying operation applied previously. Another way of phrasing it would be that the languages denoted by regular copying expressions are not closed under copying, thus restricting the denoted languages by excluding copies of copies (w^{2^n}).

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

2.2.1 Closure: homomorphism

In this section, we prove the set of languages denoted by regular copying expressions are closed under any homomorphisms. We begin with the definitions of a homomorphism and inverse homomorphism.

Definition 7. *A (string) homomorphism is a function mapping one alphabet to strings of another alphabet, written* $h : \Sigma \to \Delta^*$*. We can extend* h *to operate on strings over* Σ^* *such that*

1. $h(\epsilon_\Sigma) = \epsilon_\Delta$
2. *for* $w = a_1 a_2 \dots a_n \in \Sigma^*$, $h(w) = h(a_1)h(a_2)\dots h(a_n)$ *where each* $a_i \in \Sigma$

Definition 8. *An alphabetic homomorphism* h_0 *is a special homomorphism with* h_0 *maps each symbol in previous alphabet to another symbol in the new alphabet. In other words,* $\forall a \in \Sigma$, $h(a) \in \Delta$.

For example, $h : \{\mathrm{H}, \mathrm{L}\} \to \{\mathrm{C}, \mathrm{V}\}^*$ with $h(\mathrm{H}) = \mathrm{CVC}$ and $h(\mathrm{L}) = \mathrm{CV}$. Then, $h(\mathrm{LLH}) = \mathrm{CVCVCVC}$. Let $h_0 : \{p, a\} \to \{\mathrm{C}, \mathrm{V}\}^*$ be an alphabetic homomorphism with $h_0(p) = \mathrm{C}$ and $h_0(a) = \mathrm{V}$. Then, $h_0(papa) = \mathrm{CVCV}$.

Additionally, we define the homomorphism h operation on a language, as well as the inverse homomorphic image of a language.

Definition 9. *Given a homomorphism* $h\colon \Sigma \to \Delta^*$ *and* $\mathrm{L}_1 \subseteq \Sigma^*$, $\mathrm{L}_2 \subseteq \Delta^*$, *define* $h(\mathrm{L}_1) = \{h(w) \mid w \in \mathrm{L}_1\} \subseteq \Delta^*$ *and* $h^{-1}(\mathrm{L}_2) = \{w \mid h(w) = v \in \mathrm{L}_2\} \subseteq \Sigma^*$.

For example, let $h : \{\mathrm{H}, \mathrm{L}\} \to \{\mathrm{C}, \mathrm{V}\}^*$ with $h(\mathrm{H}) = \mathrm{CVC}$ and $h(\mathrm{L}) = \mathrm{CV}$. Then, for a language $\mathrm{L}_1 = \{(\mathrm{LH})^n \mid n \in \mathbb{N}\}$, $h(\mathrm{L}_1) = \{(\mathrm{CVCVC})^n \mid n \in \mathbb{N}\}$. Given a language $\mathrm{L}_2 = \{(\mathrm{CV})^n(\mathrm{CVC})^n\}$, $h^{-1}(\mathrm{L}_2) = \{\mathrm{L}^n\mathrm{H}^n\}$.

Theorem 1. *A language described by a regular copying expression is closed under homomorphism.*

Proof. (*adapted from Hopcroft and Ullman, 1979, 60*) Assume an arbitrary $\mathrm{L} \subseteq \Sigma^*$ is described by some regular copying expression R_1. Given an arbitrary homomorphism $h : \Sigma \to \Delta^*$, for each $a \in \Sigma$, $\exists u \in \Delta^*$ such that $h(a) = u$. Select the regular copying expression for u and assume it's R_u. Replace each occurrence of a in the RCE R_1 by R_u. After each symbol in the alphabet is replaced, we can get another expression R_2.

To prove R_2 denote $h(\mathrm{L}(\mathrm{R}_1))$, use proof by induction on the operations in R_1. In the inductive step, we only provide a complete proof showing the homomorphism of a copying closure is the

copying closure of the homomorphism. Proofs of the other three regular operations follow the same idea.

- Base cases:
  - $R_1 = \varnothing$, $L(R_1) = \varnothing$, then $h(L(R_1)) = \varnothing$. Under construction, $R_2 = \varnothing$.
    Thus, $L(R_2) = h(L(R_1))$
  - $R_1 = \epsilon$, $L(R_1) = \epsilon$, then $h(L(R_1)) = \epsilon$. Under construction, $R_2 = \epsilon$.
    Thus, $L(R_2) = h(L(R_1))$
  - $\exists a \in \Sigma, R_1 = a$, $L(R_1) = \{a\}$, then $h(L(R_1)) = \{h(a)\}$. Under construction, $R_2 = R_u$ and $L(R_u) = \{u\} = \{h(a)\}$. Thus, $L(R_2) = h(L(R_1))$
- Induction step: we assume $L(r_2) = h(L(r_1))$ holds for any regular copying expression r_1 with less than n operators ($n \geq 1$). Let us consider the case when R_1 have n operators.
  - If $R_1 = r_1^C$, $L(R_1) = \{ww \mid w \in L(r_1)\}$.
    Now assume r_2 is the regular copying expression after replacing each symbol occurring in r_1. By induction hypothesis, $L(r_2) = h(L(r_1))$. We also know $R_2 = r_2^C$. Now, we are left to show $L(R_2) = h(L(R_1))$

$$
\begin{aligned}
h(L(R_1)) &= h(\{ww \mid w \in L(r_1)\}) \\
&= \{h(ww) \mid w \in L(r_1)\} \\
&= \{h(w)h(w) \mid w \in L(r_1)\}
\end{aligned}
$$

$$
\begin{aligned}
L(R_2) &= L(r_2^C) \\
&= \{ss \mid s \in L(r_2)\} \\
&= \{ss \mid s \in h(L(r_1))\} \\
&= \{h(w)h(w) \mid w \in L(r_1)\}
\end{aligned}
$$

- Other cases ($R_1 = r_1 r_2$, $R_1 = r_1 + r_2$, $R_1 = r_1^*$) follow the similar idea.

□

We conjecture that the set of languages denoted by regular copying expressions is not closed under inverse homomorphism. Here, we provide a brief account and more detailed discussions can also be found later. Consider the mildly context-sensitive language $L = \{a^i b^j a^i b^j \mid i, j \geq 0\}$. A regular copying expression denoting this language is $(a^* b^*)^C$. Now, with an alphabetic homomorphism $h : \{0, 1, 2\} \to \{a, b\}^*$ such that $h(0) = a$, $h(1) = a$ and $h(2) = b$, the inverse homomorphic image of L is $h^{-1}(L) = \{(0+1)^i 2^j (0+1)^i 2^j \mid i, j \geq 0\}$, including the copying language $\{w 2^j w 2^j \mid w \in \{0, 1\}^*, j \geq 0\}$. However, this set of strings also includes non-identical symbol correspondences, such as 1202 and 11020002. $h^{-1}(L)$ appears to beyond the abilities of regular copying expressions. The other operators fail at the incurred crossing dependencies while the new copying operator only concerns with *identity*.

2.3 Finite-State Buffered Machine

The aim of proposing a new computing device is to add reduplication to FSAs, implement a copying mechanism and analyze the nature of such operation by deconstructing it into indispensable components. The new formalism is *finite-state buffered machines* (FSBMs), a summary of which is provided in Section 2.3.1. To avoid any potential confusion by technical complications, we introduce the new formalism by first presenting the general cases of *finite-state buffered machines* in Section 2.3.2. Examples are provided in later of Section 2.3.2 to make the system concrete and comprehensible. To better understand the copying mechanism, complete-path FSBMs are highlighted in Section 2.3.3. We present that the languages recognized by FSBMs are precisely the languages recognized complete-path FSBMs in Section 2.3.4. Section 2.3.5 scrutinizes the closure properties of the set of languages defined by complete-path FSBMs, which should hold for the general cases.

2.3.1 Finite-state buffered machines in a nutshell

Finite-state buffered machines are two-taped automata with finite-state core control. One tape stores the input, as in normal FSAs; the other serves as an unbounded memory buffer, storing reduplicants temporarily for future string matching. Intuitively, a finite-state buffered machine is an extension to finite-state registered machines (Cohen-Sygal and Wintner, 2006) but equipped with unbounded memory. In theory, finite-state buffered machines with a *bounded* buffer would be as expressive as a finite state registered automaton and therefore can be converted to an FSA.

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

There are three possible modes for the newly proposed machine M when processing an input:

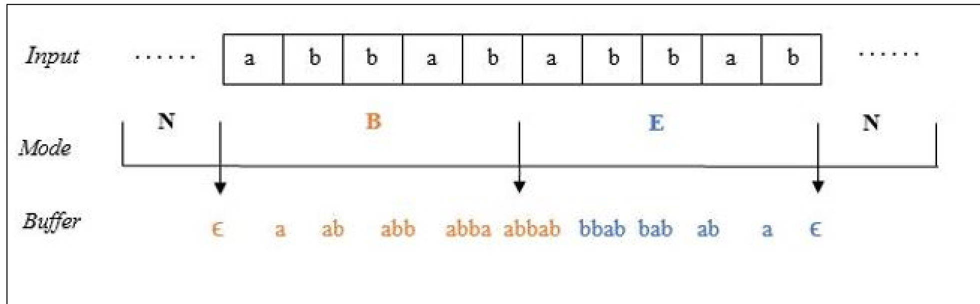


Figure 5: *Mode changes and input-buffer interaction of an FSBM* M *on "...abbababbab...". Assume* M *is armed with sufficient input consuming and symbol matching apparatus. The machine switches to* B *mode to temporarily store symbols in queue-like buffer. At the breaking point, it shifts to* E *mode for symbol matching between what's in the buffer and what's in the input. After all symbols matched, the buffer is emptied and the machine further switches to* N *mode.*

1) in normal (N) mode, M reads symbols and transits between states, functioning as a normal FSA; 2) in buffering (B) mode, besides consuming symbols from the input and taking transitions among states, it adds a copy of just-read symbols to the queue-like buffer, until it exits buffering (B) mode; 3) after exiting buffering (B) mode, M enters emptying (E) mode, in which M matches the stored symbols in the buffer against input symbols. When all buffered symbols have been matched, M switches back to normal (N) mode for another round of computation. Figure 5 provides a schematic diagram manifesting how the mode of a machine alternates when it determines the equality of sub-strings and how the buffer interacts with the input. Under the current augmentation, FSBMs can only capture local reduplication with two adjacent, completely identical copies. It cannot handle non-local reduplication, nor multiple reduplication.

So far, we have only introduced the basic characteristics and how different modes lead to different actions and behaviors in a finite state buffered machine. Then, one question on hold stands out: how does a machine know when to switch to which mode? The control of mode changes in those machines will be easier to grasp after introducing the formal definitions; details will be discussed later in this section.

2.3.2 Mathematical definitions and examples

Definition 10. *A **Finite-State Buffered Machine** is a 7-tuple* $\langle \Sigma, Q, I, F, G, H, \delta \rangle$ *where*

- *Q: a finite set of states*
- $I \subseteq Q$*: initial states*
- $F \subseteq Q$*: final states*
- $G \subseteq Q$*: states where the machine must enter buffering* (B) *mode*
- $H \subseteq Q$*: states requiring string matching and* $G \cap H = \varnothing$
- δ*:* $Q \times (\Sigma \cup \{\epsilon\}) \times Q$*: the state transitions according to a specific symbol*

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

Definition 11. *A **configuration** of an FSBM* $D = (u, q, v, t) \in \Sigma^* \times Q \times \Sigma^* \times \{\text{N}, \text{B}, \text{E}\}$*, where* u *is the input string;* v *is the string in the buffer;* q *is the current state and* t *is the current mode the machine is in.*

Definition 12. *Given an FSBM M and* $x \in (\Sigma \cup \{\epsilon\})$*,* $u, w, v \in \Sigma^*$*, we define a configuration* D_1 ***yields*** *a configuration* D_2 *in* M ($D_1 \vdash_M D_2$) *as the smallest relation such that:*

- *For every transition* (q_1, x, q_2) *with at least one state of* $q_1, q_2 \notin H$

  $(xu, q_1, \epsilon, \text{N}) \vdash_M (u, q_2, \epsilon, \text{N})$ *with* $q_1 \notin G$ — *"normal" actions*

  $(xu, q_1, v, \text{B}) \vdash_M (u, q_2, vx, \text{B})$ *with* $q_2 \notin G$ — *"buffering" actions*

- *For every transition* (q_1, x, q_2) *and* $q_1, q_2 \in H$

  $(xu, q_1, xv, \text{E}) \vdash_M (u, q_2, v, \text{E})$ — *"emptying" actions*

- *For every* $q \in \mathrm{G}$

  $(u, q, \epsilon, \textsc{n}) \vdash_{\mathrm{M}} (u, q, \epsilon, \textsc{b})$ *mode-changing actions*

- *For every* $q \in \mathrm{H}$

  $(u, q, v, \textsc{b}) \vdash_{\mathrm{M}} (u, q, v, \textsc{e})$ *mode-changing actions*

  $(u, q, \epsilon, \textsc{e}) \vdash_{\mathrm{M}} (u, q, \epsilon, \textsc{n})$ *mode-changing actions*

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

Definition 13. *A **run** of* M *on* w *is a sequence of configurations* $\mathrm{D}_0, \mathrm{D}_1, \mathrm{D}_2 \ldots \mathrm{D}_m$ *such that 1)* $\exists\, q_0 \in \mathrm{I}$, $\mathrm{D}_0 = (w, q_0, \epsilon, \textsc{n})$*; 2)* $\exists\, q_f \in \mathrm{F}$, $\mathrm{D}_m = (\epsilon, q_f, \epsilon, \textsc{n})$*; 3)* $\forall\, 0 \le i < m$, $\mathrm{D}_i \vdash_{\mathrm{M}} \mathrm{D}_{i+1}$*. The language recognized by M is denoted by* $\mathrm{L(M)}$, $w \in \mathrm{L(M)}$ *iff there's a run of* M *on* w.

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

Example 1. Total reduplication Figure 6 offers an FSBM M_1 for L_{ww}, with any arbitrary strings made out of an alphabet $\Sigma = \{a, b\}$ as candidates of bases.

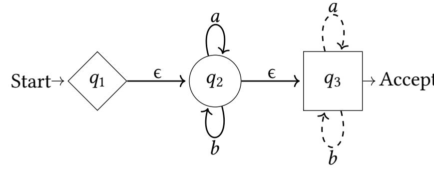
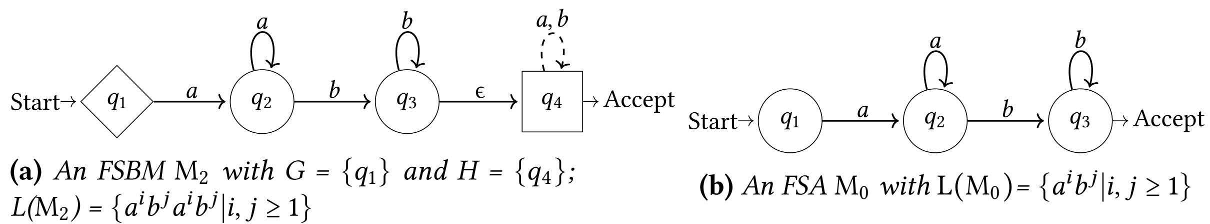


Figure 6: *An FSBM* M_1 *with* $G = \{q_1\}$ *(diamond) and* $H = \{q_3\}$ *(square); dashed arcs are used only for the emptying process.* $L(\mathrm{M}_1) = \{ww | w \in \{a, b\}^*\}$

For the rest of the illustration, we focus on the FSBM M_2 in Figure (7a). M_2 in Figure (7a) recognizes the non-context free language $\{a^i b^j a^i b^j \mid i, j \ge 1\}$. This language can be viewed as total reduplication added to the regular language $\{a^i b^j \mid i, j \ge 1\}$ (recognized by the FSA M_0 in Figure 7b). State q_1 is an initial state and more importantly a G state, forcing M_2 to enter B mode before it takes any arcs and transits to other states. Then M_2 in B mode always keeps a copy of consumed symbols until it proceeds to State q_4, an H state. State q_4 requires M_2 to stop buffering

and switch to E mode in order to empty the buffer by checking for string identity. Using the special transitions between H states (in this case, *a* and *b* loops on State q_4), M_2 checks whether the stored symbols in the buffer matches the remaining input. If so, after emitting out all symbols in the buffer, M_2 with a blank buffer can switch to N mode. It eventually ends at State q_4, a legal final state. Figure 8 gives a complete run of M_2 on the string "*abbabb*". Figure 9 shows M_2 rejects the non-total reduplicated string "*ababb*" since a final configuration cannot be reached.



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

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

Figure 7: *One example FSBM and the corresponding FSA for the base language*

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

Figure 8: M_2 *in Figure 7a accepts abbabb*

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

Figure 9: M_2 *in Figure 7a rejects ababb*

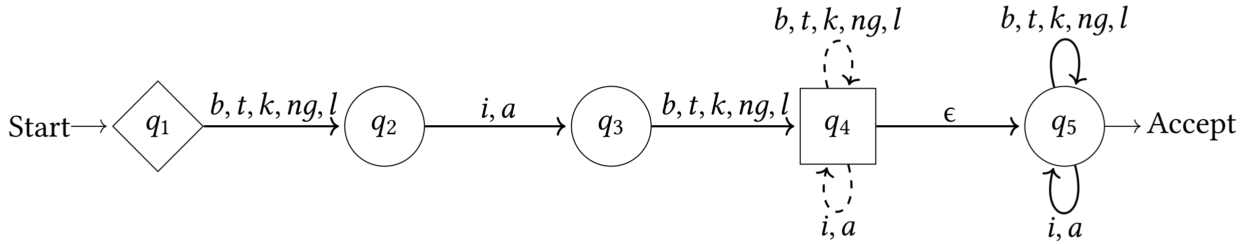


Figure 10: *An FSBM* M_3 *for Agta CVC-reduplicated plurals: G = {q_1} and H = {q_4}*

Example 2. Partial reduplication Assume $\Sigma = \{b, t, k, ng, l, i, a\}$, the FSBM M_3 in Figure 10 serves as a model of two Agta CVC reduplicated plurals in Table 1. Given the initial state q_1 is in G, M_3 has to enter B mode before it takes any transitions. In B mode, M_3 transits to a plain state q_2, consuming a consonant in 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 state, urges M_3 to stop buffering and enter E mode. Using the special transitions between H states (in this case, loops on q_4), M_3 matches the CVC in the buffer with the remaining input. Then, M_3 with a blank buffer can switch to N mode at q_4. M_3 in N mode loses the access to loops on q_4, as they are available only in E mode. It transits to q_5 to process the rest of the input by the normal transitions between q_5. A successful run should end at q_5, the only final state. Figure 11 gives a complete run of M_3 on the string "*taktakki*". As illustrated by Figure 12, M_3 rejects the string with non-matching sub-string "tiktakki".

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

Figure 11: M_2 *in Figure 10 accepts taktakki*

	Used Arc	*State Info*	*Configuration*	
1.	*N/A*	$q_1 \in$ G	(*tiktakki*, q_1, ϵ, N)	Buffering triggered by q_1 and empty buffer
2.	*N/A*		(*tiktakki*, q_1, ϵ, B)	
3.	(q_1, t, q_2)	$q_2 \notin$ G	(*iktakki*, q_2, *t*, B)	
4.	(q_2, i, q_3)		(*ktakki*, q_3, *ti*, B)	
5.	(q_3, k, q_4)	$q_4 \in$ H	(*takki*, q_4, *tik*, B)	Emptying triggered by q_4
6.	*N/A*		(*takki*, q_4, *tik*, E)	
7.	(q_4, t, q_4)		(*akki*, q_4, *ik*, E)	
			Clash	

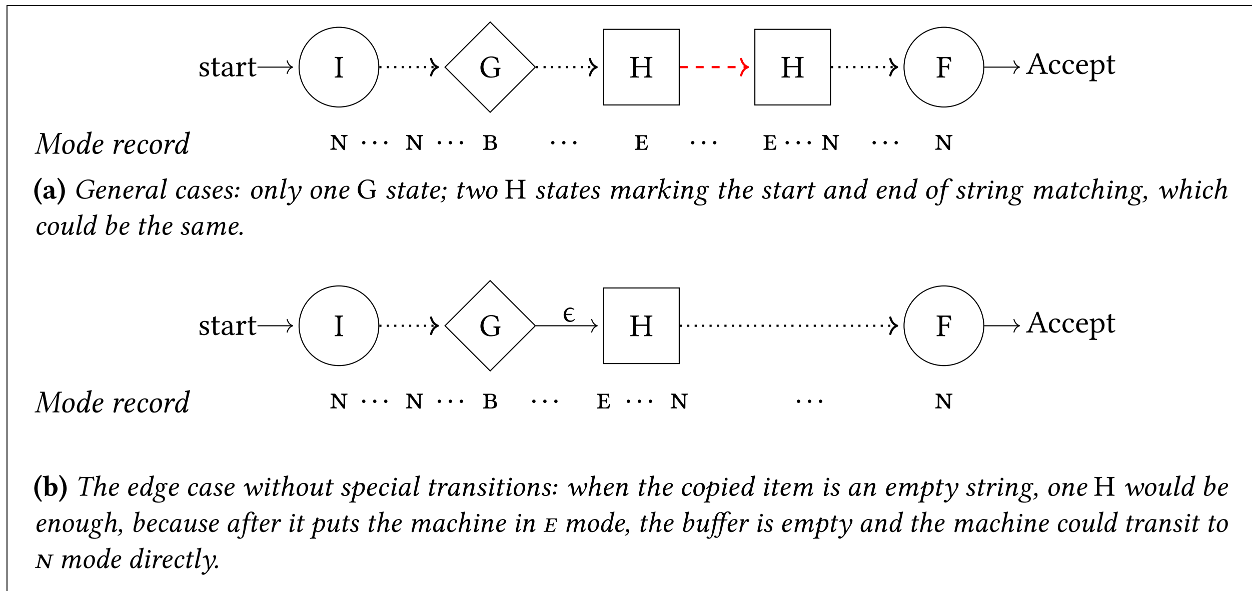
Figure 12: M_2 *in Figure 10 rejects tiktakki*

2.3.3 Implementation of the copying mechanism and complete-path FSBMs

The copying mechanism is realized by four essential components: 1) the unbounded memory buffer, which has queue storage; 2) three added modalities of the machine; 3) added specifications of states urging the machine to buffer symbols into memory, namely states in G; 4) added specifications of states urging 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 2.3.2, the machine is supposed to 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, E, N mode changes. Correspondingly, the resulting languages reflect either no copying (functioning as plain FSAs) or full copying. The notion of "half-copying" is prohibited.

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 enter E mode and transit only between H states in E mode. 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 states later to be put in E mode. 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 13.



(a) *General cases: only one* G *state; two* H *states marking the start and end of string matching, which could be the same.*

(b) *The edge case without special transitions: when the copied item is an empty string, one* H *would be enough, because after it puts the machine in* E *mode, the buffer is empty and the machine could transit to* N *mode directly.*

Figure 13: *The template for the implementation of the copying in FSBMs. Key components:* G *state,* H *states, transitions between* H *states, and strict ordering between* G *and* H. *Solid lines represent a transition in one step. Dotted lines represent a sequence of normal transitions. Black dotted lines replace plain non-G non-H states. H states in between H states are replaced by red dashed lines.*

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 and define the *completeness restriction* on a path as in Definition 15. While maintaining definitions of configurations and configuration changes unchanged, we define those machines whose architecture has all paths allowing the *possibility* of full-cycle mode changes to meet the *completeness restriction.*

Definition 14. *A* ***path*** *from one state p to another state q in a machine is a sequence of states $p, p_1, p_2, p_3, \ldots p_n, q$ such that there is a transition between two adjacent states in such sequence.*

Definition 15. *A path s from an initial state to a final state in a machine is said to be* ***complete*** *if*

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

Schematically, we use P to represent those non-G, non-H plain states ($P = Q - (G \cup H)$). Moreover, we represent initial, final states as I, F respectively. Then, the regular expression denoting the state information in a path s should be of the form: $I(P^*GP^*HH^*P^* + P^*)^*F$. We further define a copying path to be guaranteed to trigger the full copying and rule out the "no copying" case.

Definition 16. *A path is said to be* ***a copying path*** *if it is complete and there is at least one* G *state.*

Definition 17. *A* ***complete-path finite-state buffered machine*** *is an FSBM in which all possible paths are complete.*

The machine M_1 in Figure 6, M_2 Figure 7a and M_3 in Figure 10 are illustrations of complete-path finite-state buffered machines. For the rest of this section, we describe several cases for an incomplete path in M, as further illustrated in Figure 14.

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

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

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

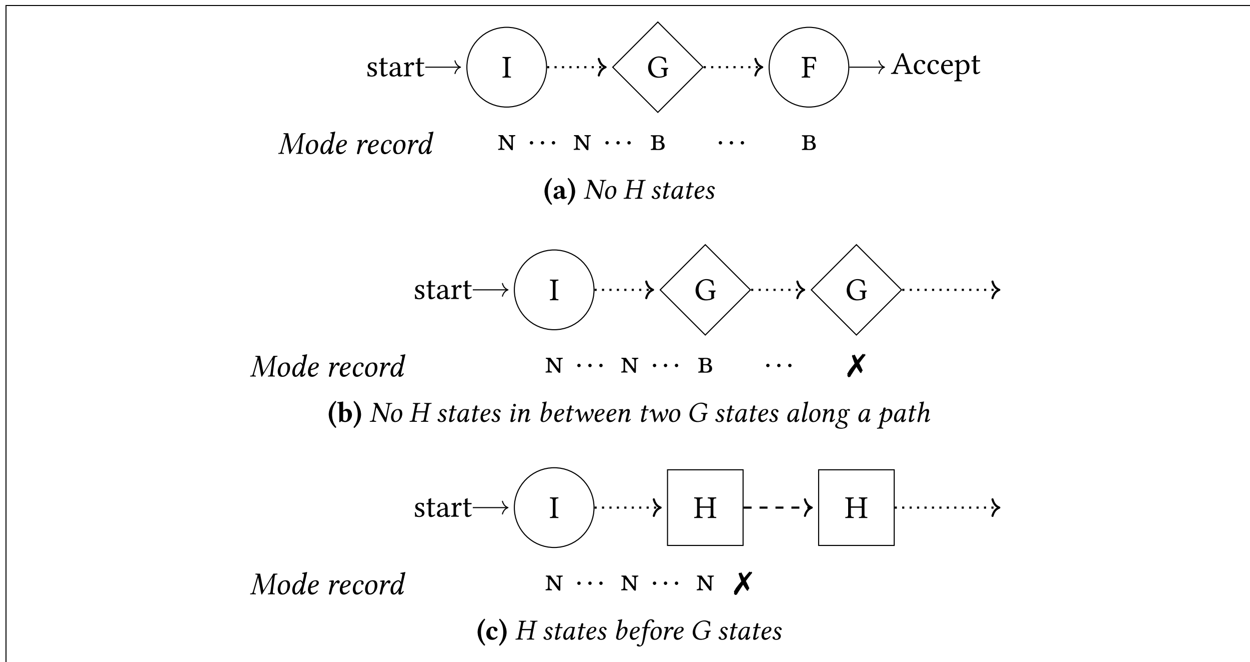


Figure 14: *Possible paths in a machine failing on the completeness requirement. Dotted lines represent a sequence of normal transitions. Dashed lines are special transitions between H states in one step.*

It should be emphasized that a machine M in N mode can pass through one (and only one) H state to another plain state, because the relation between configurations for machines in normal modes only requires at least one state to not be an H state. Then, some incomplete paths could lead to non-empty languages, but those languages are still regular. For instance, the language of the FSBM M_4 in Figure 15 is equivalent to the language recognized by the FSA in Figure 16. M_4 remains to be an incomplete FSBM because it doesn't have any G state preceding the H states q_2 and q_4.

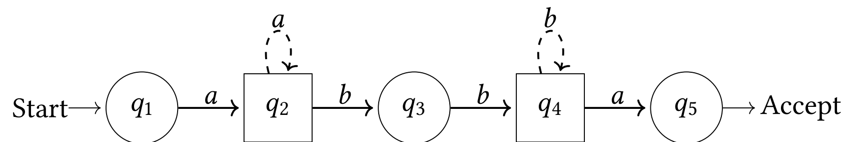


Figure 15: *An incomplete FSBM* M_4 *with* $G = \varnothing$ *and* $H = \{q_2, q_4\}$; $L(M_4) = \{abba\}$

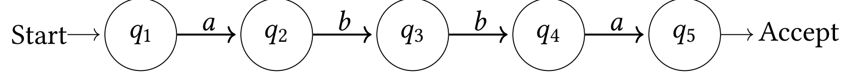


Figure 16: *An FSA (or an FSBM with* G = ∅ *and* H = ∅*) whose language is equivalent as* M_3 *in Figure 15*

2.3.4 The equivalence between general FSBMs and complete-path FSBMs

In this section, we argue that the languages recognized by FSBMs are precisely the languages recognized by complete-path FSBMs. One key observation is the language recognized by the new machine is the union of the languages along all possible paths. Then, the validity of such a statement builds on different incomplete cases of G and H states along a path mentioned in the previous section. Here, we provide a more direct and comprehensive analysis, listed in Table 3 and it will be easy to construct a proof by cases. It can be noticed that only finite-state buffered machine with at least one copying path uses the copying power and extends the regular languages. Otherwise, finite-state buffered machine without any copying path is equivalent to an FSA and the corresponding language is still in the regular set. On the other hand, FSAs can be viewed as FSBMs without a copying path: they can be converted to an FSBM with an empty G set, an empty H set and trivially no special transitions between H states.

Cases		complete	copying	resulting languages
$\lvert G\rvert = 0, \lvert H\rvert = 0$		✓	✗	regular
$\lvert G\rvert \geq 1, \lvert H\rvert = 0$		✗	✗	$L_\varnothing$
$\lvert G\rvert = 0, \lvert H\rvert \geq 1$	any adjacent H sequences	✗	✗	$L_\varnothing$
	no adjacent H	✗	✗	regular
$\lvert G\rvert \geq 1, \lvert H\rvert \geq 1$	well-ordered as sequences of G...HH*	✓	✓	copying-derived
	ill-ordered cases	✗	✗	$L_\varnothing$

Table 3: *Different cases for* G *and* H *states along a path*

2.3.5 Closure properties of complete-path FSBMs

In this section, we show closure properties of the languages recognized by complete-path FSBMs. Noticeably, given complete-path FSBMs are finite-state machines with a copying mechanism, the proof ideas in this section are similar to the standard proofs for FSAs, which can be found in Hopcroft and Ullman (1979) and Sipser (2013).

Intersection with FSAs

Theorem 2. *If* L_1 *is a complete-path FSBM-recognizable language and* L_2 *is a regular language, then* $L_1 \cap L_2$ *is a complete-path FSBM-recognizable language.*

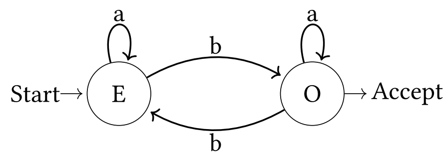
Proof. In other words, if L_1 is a language recognized by a complete-path FSBM $M_1 = \langle Q_1, \Sigma, I_1, F_1, G_1, H_1, \delta_1 \rangle$, and L_2 is a language recognized by an FSA $M_2 = \langle Q_2, \Sigma, I_2, F_2, \delta_2 \rangle$, then $L_1 \cap L_2$ is a language recognizable by another complete-path FSBM. It's easy to construct an intersection machine M where $M = \langle Q, \Sigma, I, F, G, H, \delta \rangle$ such that

- $Q = Q_1 \times Q_2$
- $I = I_1 \times I_2$
- $F = F_1 \times F_2$
- $G = G_1 \times Q_2$
- $H = H_1 \times Q_2$
- $((q_1, q_1'), x, (q_2, q_2')) \in \delta$ iff $(q_1, x, q_2) \in \delta_1$ and $(q_1', x, q_2') \in \delta_2$

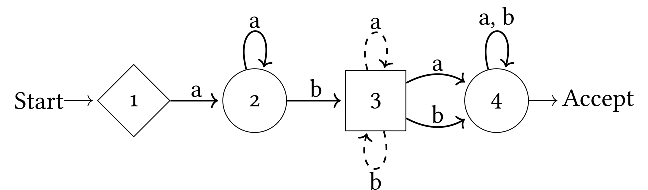
Paths in M would inherit the completeness from M_1 given the current construction. Then, $L(M) = L_1 \cap L_2$, as M simulates $L_1 \cap L_2$ by running M_1 and M_2 simultaneously. M accepts w iff both M_1 and M_2 accept w. A detailed proof showing $L(M) = L_1 \cap L2$ can be found in the Appendix B. □

Example An example demonstrating how the intersection works can be found in Figure 17. The FSA in Figure 17a recognizes the language whose strings have odd numbers of *b*s, such as *b*, *ababb* and *aabbababb*. The FSBM in Figure 17b computes the language after initial aa*b- copying, such as *ababa*, *aabaabb*, *aaabaaabaaaa*. The intersection FSBM is shown in Figure 17c, which recognizes languages with both initial 'aa*b-' reduplication and odd number of *b*s, such as *ababb* and *aabaabaaaab*.

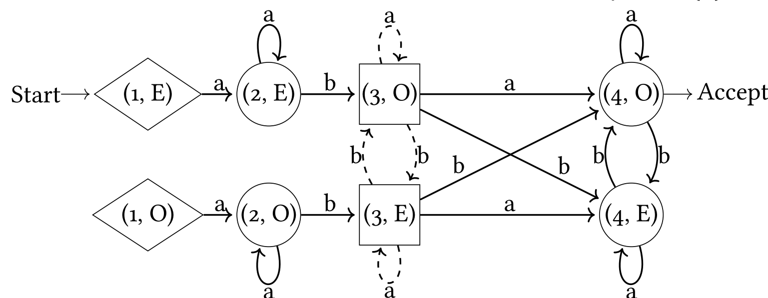
To observe how the intersection FSBM achieves both restrictions at the same time, let us look at its internal mechanisms more closely: the only initial state is (1, E), which is also a G state. Therefore, the machine would be put in B mode. It stores any string w of the form aa^*b in the buffer and reaches (3, O). This H state puts the machine in E mode to further match the input with w in the buffer. The machine stays in the state (3, O) after checking the $aa*$ and transits to state (3, E) after checking the b. (3, E) keeps a record that so far, the machine sees two *b*s. At state (3, E), the machine switches to N mode because of an empty buffer. For the rest of the input, the machine transits between (4, O) and (4, E), whose transitions are governed by the FSA transitions. The machine can only end at (4, O), which marks an odd number of *b*s.



(a) *an FSA enforcing odd number of bs in a string. State E and State O represent even, odd number of bs in the prefix respectively*



(b) *a complete-path FSBM recognizing initial 'aa*b'- identity.* G = {1}, H = {3}



(c) *the intersection FSBM after the construction:* G = {(1, E), (1, O)}, H = {(3, 0), (3, E)}.

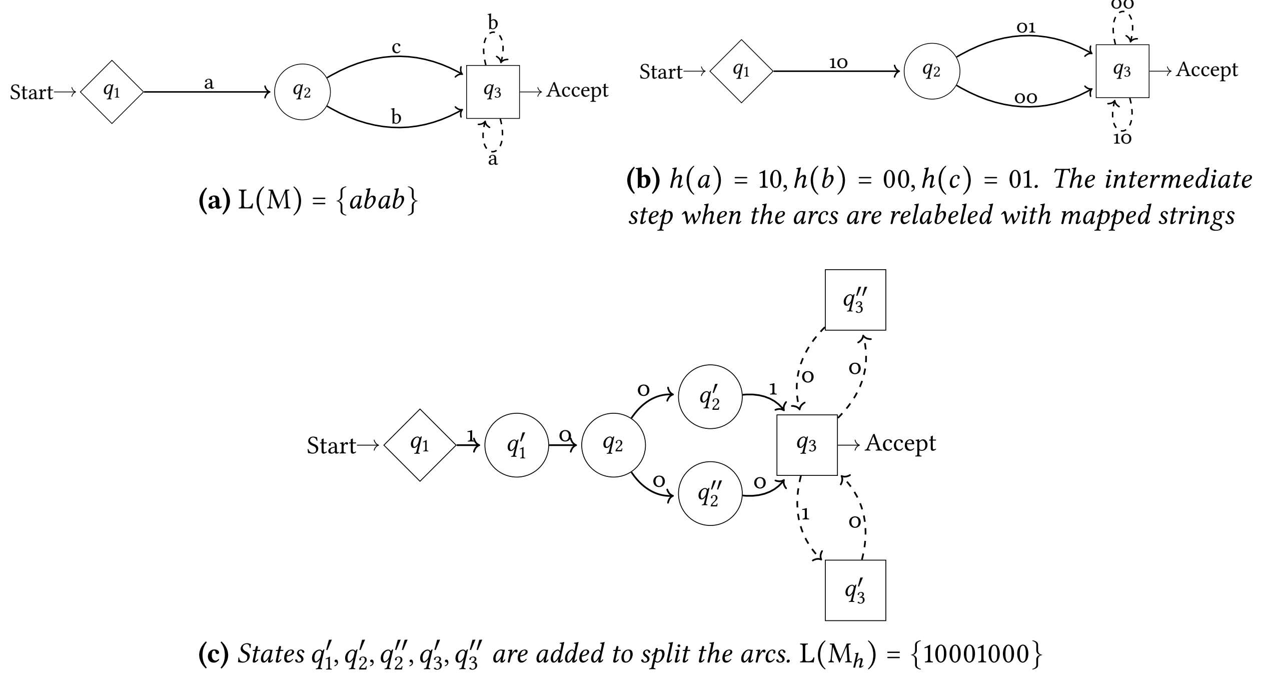
Figure 17: *Example for the intersection construction*

Homomorphism

Theorem 3. *The class of languages recognized by complete-path FSBMs are closed under any homomorphisms.*

Proof. That complete-path FSBM languages is closed under homomorphism can be proved by constructing a new machine M_h based on the base machine M. The construction goes as follows: relabel the odd arcs to mapped strings and add states to split the arcs so that there is only one symbol or ϵ on each arc in M_h. When there are multiple symbols on any normal arcs, the newly added states can only be plain non-G, non-H states. For multiple symbols on the special arcs between two H states, all newly added states are H states. All paths in M_h are complete since the construction does not affect the ordering between G and H states. ◻

Example Figure 18 is a simple instance of the construction of a new machine recognizing the homomorphic image. Given $\Sigma = \{a, b, c\}$ and $h : \Sigma \to \{0, 1\}^*$, the base machine recognizes a finite language $\{abab\}$. Even it contains a 'c' transition from q_2 to q_3; it is not possible to recognize any strings with c because (q_2, c, q_3) is part of the copying, and there is no special arcs with c for future string matching. To construct the new machine of the homomorphism language, we relabel each arc with their mapped strings. The intermediate representation is shown in Figure 18b. Then, after adding states and splitting each arc when necessary, we get the result FSBM as in Figure 18c, which recognizes the singleton language $\{10001000\}$.



(a) $\mathrm{L(M)} = \{abab\}$

(b) $h(a) = 10, h(b) = 00, h(c) = 01$. *The intermediate step when the arcs are relabeled with mapped strings*

(c) *States* $q_1', q_2', q_2'', q_3', q_3''$ *are added to split the arcs.* $\mathrm{L}(\mathrm{M}_h) = \{10001000\}$

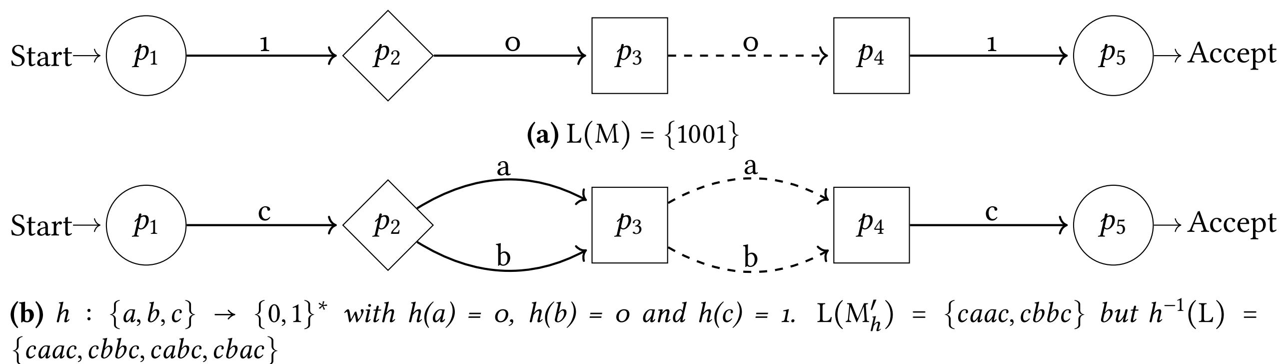
Figure 18: *Constructions used for the homomorphic language in Theorem 3.*

Inverse homomorphisms? We conjecture that the class of languages recognized by complete-path FSBMs is not closed under inverse alphabetic homomorphisms and thus inverse homomorphism. Consider a complete-path FSBM-recognizable language $\mathrm{L} = \{a^i b^j a^i b^j \mid i, j \geq 1\}$ (see Figure 7a). Consider an alphabetic homomorphism $h : \{0, 1, 2\} \rightarrow \{a, b\}^*$ such that $h(0) = a$, $h(1) = a$ and $h(2) = b$. Then, the inverse homomorphic image of L is $h^{-1}(\mathrm{L}) = \{(0+1)^i 2^j (0+1)^i 2^j \mid i, j \geq 1\}$, including 002002, 1202, 11020002. The language $\{w2^j w2^j \mid w \in \{0,1\}^*, j \geq 1\} \subset h^{-1}(\mathrm{L})$. $h^{-1}(\mathrm{L})$ seems to be challenging for complete-path FSBMs. Finite state machines cannot handle the incurred crossing dependencies while the augmented copying mechanism only contributes to recognizing *identical* copies, but not general cases of symbol correspondence. [4]

Even though we are pessimistic about inverse homomorphism closure property, analyzing why it does not hold would hint at some (in-)capabilities of FSBMs. We suspect the pivotal point comes from the one-to-many mapping. At first glance, one might want to try the conventional construction of inverse homomorphism in FSAs in FSBMs and build a new machine M_h', which

[4] However, we admit that a more formal and rigorous mathematical proof proving such language is not FSBM-recognizable should be carried out. That's why the current status of inverse homomorphism closure is a conjecture. To achieve that, a more formal tool, like a developed pumping lemma for FSBM languages is needed.

reads any symbol a in the new alphabet and simulates M on $h(a)$, as shown in Figure 19.



(a) L(M) = {1001}

(b) $h : \{a, b, c\} \rightarrow \{0, 1\}^*$ *with h(a) = 0, h(b) = 0 and h(c) = 1.* $L(M'_h) = \{caac, cbbc\}$ *but* $h^{-1}(L) = \{caac, cbbc, cabc, cbac\}$

.

Figure 19: *Under-generation of the conventional construction of the inverse homomorphic image*

However, it should be noted that such construction eventually yields under-generation: the newly constructed FSBMs still track the identity relation among symbols, which is already obscured under the definition of inverse homomorphism. In the example in Figure 19, there are two regular copying expression that can result in the language {1001}: $1(0)^C1$ (with full copying) and 1 0 0 1 (without copying but just concatenation).[5] Namely, inverse homomorphism would want a machine to be insensitive to such ambiguity and give all possible base languages. However, *under such construction*, the resulting machine is still sensitive to the identity relation. One might think another construction method, which let FSBMs insensitive to the ambiguities of internal structures by including both the copied interpretation and the non-copied interpretation, would get us around with the issue. However, as we see from the potential counter-example, in the inverse homomorphic image of a complete-path FSBM language, languages without copying could be beyond FSBM's capabilities.

Parallel multiple context-free grammars (Seki et al., 1991) adds unbounded copying to multiple context-free grammars. The conjecture we maintain here is in line with the fact that the family of parallel multiple context-free languages is not closed under inverse morphism (Nishida and Seki, 2000, 145, Corollary 12).

[5]Note, RCEs are used here to understand what is happening within the languages. No claims about the relationship between RCEs and FSBMs have been made.

Three regular operations

Theorem 4. *If* L_1, L_2 *are two complete-path 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 M_1 and M_2 such that $L(M_1) = L_1$ and $L(M_2) = L_2$, then …

Union $L_1 \cup L_2$ is a complete-path FSBM-recognizable language. Assume $M_1 = \langle \Sigma, Q_1, I_1, F_1, G_1, H_1, \delta_1 \rangle$ accept L_1 while $M_2 = \langle \Sigma, Q_2, I_2, F_2, G_2, H_2, \delta_2 \rangle$ accepts L_2. 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 plain 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. Note q_0 is a non-G, non-H plain state. In this way, the new machine would guess on either M_1 or M_2 accepts the input. If one accepts w, M will accept w, too.

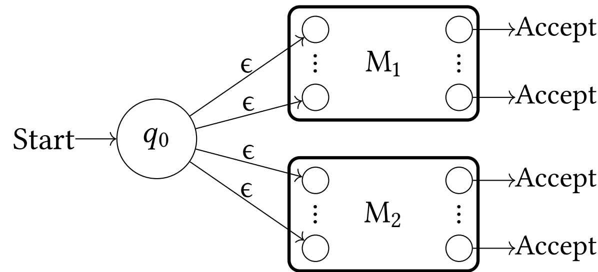


Figure 20: *The construction used in the union of two FSBMs*

Concatenation there's 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 $L(M) = L_1 \circ L_2$

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

The new machine adds a new plain state q_0 and make 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. Besides, M adds ϵ-arcs from any old final states in M_1 to any possible initial states in M_2. The *completeness* obeying feature is inherited. A path in the resulting machine is guaranteed to be complete because it is essentially the concatenation of two complete paths.

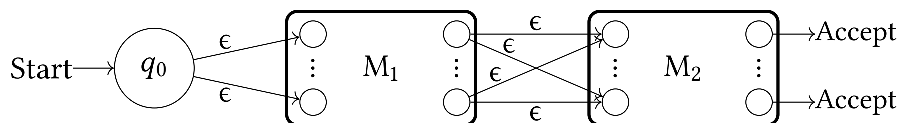


Figure 21: *The construction used in the concatenation of two FSBMs*

It is important to obey the completeness requirements and ensure N mode when it gets the end of M_1. Incomplete paths in two arbitrary machines might create a perfect copying path, thus over-generating under the construction of concatenation mentioned here. For example, as illustrated in Figure 22, imagine one path in M_1 only has G states but no H states, and another path in M_2 starts with consecutive H states. They both recognize the empty set language $L_\varnothing = \varnothing$. Therefore, the concatenation of these two languages should also be $L_\varnothing$. However, under the construction discussed previously, the new machine might create some non-empty copying languages.

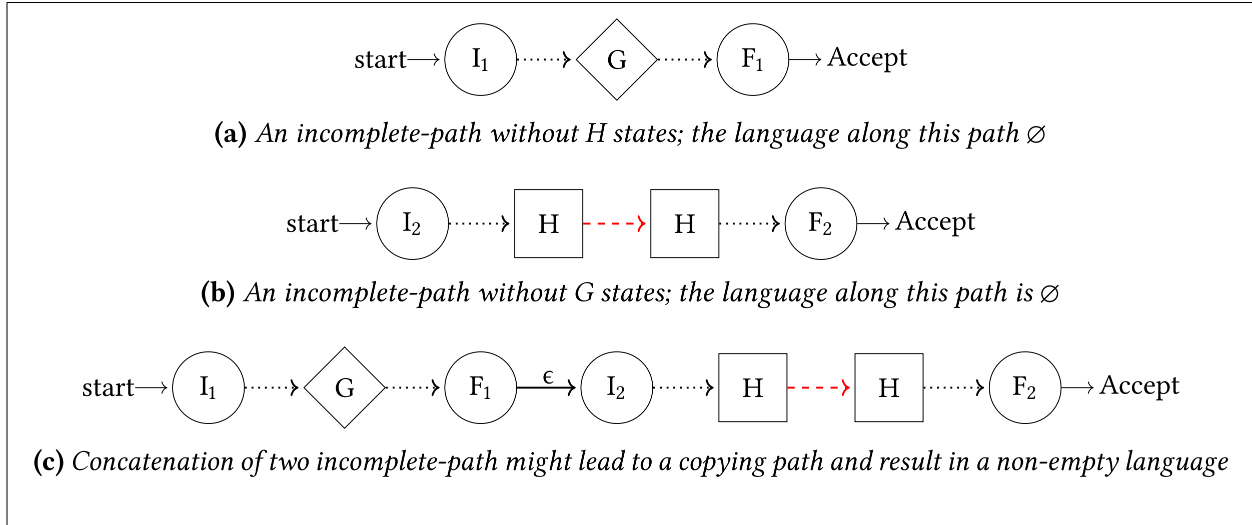


Figure 22: *Problems arise in the concatenation of two incomplete paths. Dotted lines represent a sequence of normal transitions. Red dashed lines represent a sequence of special transitions*

The over-generation issue is tied with the construction method, but not the closure of concatenation. A different concatenation construction would resolve the problem. For example, only concatenate when two paths are complete. Again, since general finite-state buffered machines are equivalently expressive as complete-path finite-state buffered machines, the properties hold for the set of complete-path FSBM-recognizable languages would still hold for the most general cases.

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 $L(M) = (L_1)*$

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

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 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. The completeness obeying feature is inherited. □

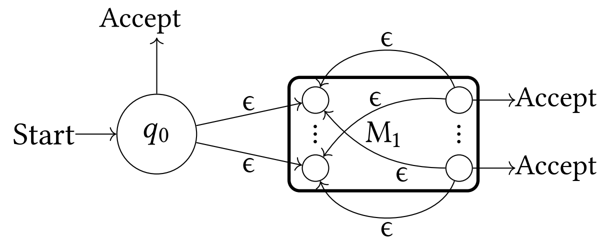


Figure 23: *The construction used in the star operation*

2.4 The equivalence between RCEs and FSBMs

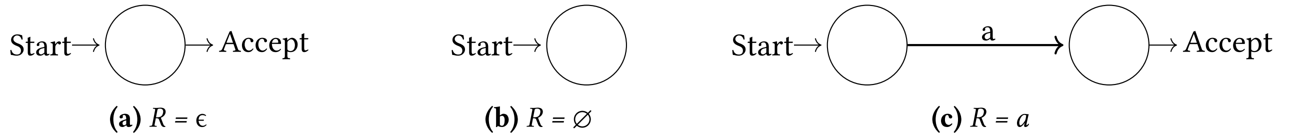
Section 2.2 introduced regular copying expressions, while in Section 2.3, we proposed another computing device: finite-state buffered machines. This section shows they are equivalent in terms of the expressivity: namely, the languages accepted by FSBMs are precisely the languages denoted by RCEs. We 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.

RCE to FSBM

Theorem 5. *Let* R *be a regular copying expression. Then, there exists an FSBM that recognizes* L(R).

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

Base case: zero operators R must be ϵ, $\varnothing$, a for some symbol a in Σ. Then, the FSBMs in Figure 24 meet the requirements.



(a) $R = \epsilon$ **(b)** $R = \varnothing$ **(c)** $R = a$

Figure 24: *FSBMs for the base step in Theorem 5. All have* $G = \varnothing; H = \varnothing$

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
  - $Q = Q' \cup \{q_0, q_f\}$
  - $G = I = \{q_0\}$
  - $H = F = \{q_f\}$

- $\delta = \delta' \cup \{(q_f, x, q_f) \mid x \in \Sigma\} \cup \{(q_0, \epsilon, q) \mid q \in I'\} \cup \{(q, \epsilon, q_f) \mid q \in F'\}$

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. Adding loops of all symbols in Σ between q_f (thus, between H states) creates all possible strings out of Σ and eventually contain all strings in $L(R_1)$. Thus, if w is in $L(R_1)$, ww must be in the language accepted by this complete-path FSBM and nothing beyond. Figure 25 shows such a construction, and Appendix C provides the proof for $L(M) = L(R)$.

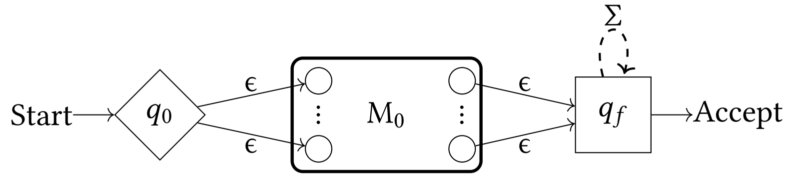


Figure 25: *The construction used in converting the copy expression* R_1^C *to a finite state buffered machine.* $L(M_0) = L(R_1)$.

- 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, $R = R_1 + R_2$, 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 in Theorem 4, we can construct the new FSBM for R.

□

FSBM to RCE

Theorem 6. *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, 33-34).

The previous discussion on the realization of the copying mechanism in complete-path FSBMs concluded with four aspects 1) the specification of G states, 2) the specification of H states, 3) the special transitions between H states, and 4) 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 second stage of copying (i.e., the string matching, or buffer-emptying) only uses special transitions among H states. Therefore, let us further zoom in to the areas with all H states. Intuitively, we want to find the borderlines where the machine first reaches an H state and ends with another (or the same) H state. So we need first to select those border states adjacent to some non-H states. For any pairs of border H states and the transitions in between, it can be treated as a small FSA abstractly since there is no complete notion of copying yet. Following this vein, utilizing the FSA-to-RE conversion produces a regular expression R_1 denoting the languages between any two H states on the border.

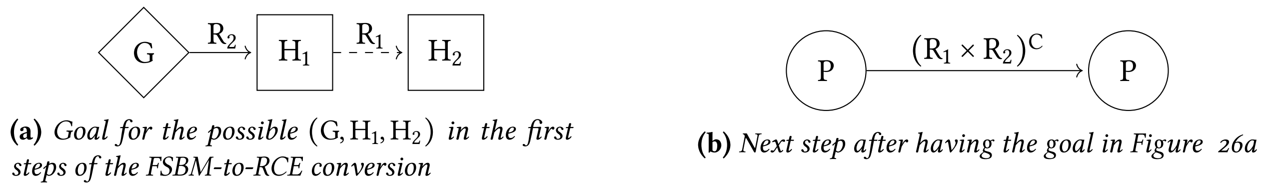
After tallying up all languages that are candidates of string matching, we switch our scope to G states, which marks the beginning of the copying and the beginning of symbol buffering. The core is the same: for any G and borderline H pair, if they do not cross other H states, borrow the FSA-to-RE conversion to get a regular expression R_2 denoting the languages possible to be stored in the buffer temporarily. If they have to cross other H states, this borderline H does not immediately follow that G state and we would put down $\varnothing$ for now.

The most essential step is to combine what we have: given pairs of 1) G and border Hs and 2) pairs of border Hs, the language produced by any G, H_1, H_2 sequence is the copy of the *intersection*

of the languages between H_1, H_2 (R_1) and the languages between G, H_1 (R_2). In other words, the copying for any G, H_1, H_2 sequences would be equivalent to $(R_1 \times R_2)^C$. Although the definition of a regular expression does not contain an intersection operation, since generalized regular expressions with the closure of intersection are equivalently expressive as regular expressions, there must be a regular expression $R_3 = R_1 \times R_2$. Therefore, the RCE is just R_3^C.

Importantly, there are only finitely many (G, H_1, H_2) tuples. Iterating through all possible intermediate border H_1 states and getting a general RCE R by the union, we use two plain states with the RCE R along the arc to denote the languages from G to H_2. Then we plug them back to the starting FSBM. Note that all special transitions are also eliminated. Then, we get an intermediate representation with only plain states. Similar ideas as FSA-to-RE conversion could be applied again to get the final regular copying expression for this FSBM.

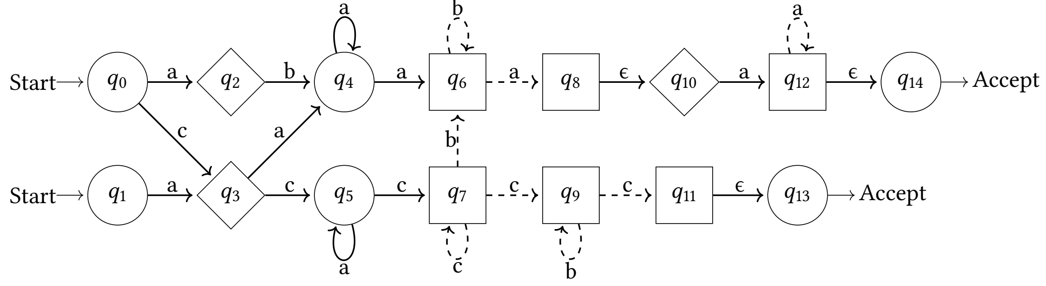
The described conversion of the copying mechanism in a machine to a copy expression is depicted in Figure 26 with an example provided in Figure 27.



(a) *Goal for the possible* (G, H_1, H_2) *in the first steps of the FSBM-to-RCE conversion*

(b) *Next step after having the goal in Figure 26a*

Figure 26: *The conversion of the copying mechanism in an FSBM to RCE. P represents the plain, non-H, non-G states*



(a) *An FSBM to start with.* $G = \{q_2, q_3, q_{10}\}$; $H = \{q_6, q_7, q_8, q_9, q_{11}, q_{12}\}$

“border” H pairs	RE	“border” H pairs	RE	“border” H pairs	RE	“border” H pairs	RE	“border” H pairs	RE
(q_6, q_6)	b^*	(q_7, q_6)	c^*bb^*	(q_{12}, q_6)	$\varnothing$	(q_8, q_6)	$\varnothing$	(q_{11}, q_6)	$\varnothing$
(q_6, q_7)	$\varnothing$	(q_7, q_7)	c^*	(q_{12}, q_7)	$\varnothing$	(q_8, q_7)	$\varnothing$	(q_{11}, q_7)	$\varnothing$
(q_6, q_8)	b^*a	(q_7, q_8)	c^*bb^*a	(q_{12}, q_8)	$\varnothing$	(q_8, q_8)	$\varnothing$	(q_{11}, q_8)	$\varnothing$
(q_6, q_{11})	$\varnothing$	(q_7, q_{11})	c^*cb^*c	(q_{12}, q_{11})	$\varnothing$	(q_8, q_{11})	$\varnothing$	(q_{11}, q_{11})	$\varnothing$
(q_6, q_{12})	$\varnothing$	(q_7, q_{12})	$\varnothing$	(q_{12}, q_{12})	a^*	(q_8, q_{12})	$\varnothing$	(q_{11}, q_{12})	$\varnothing$

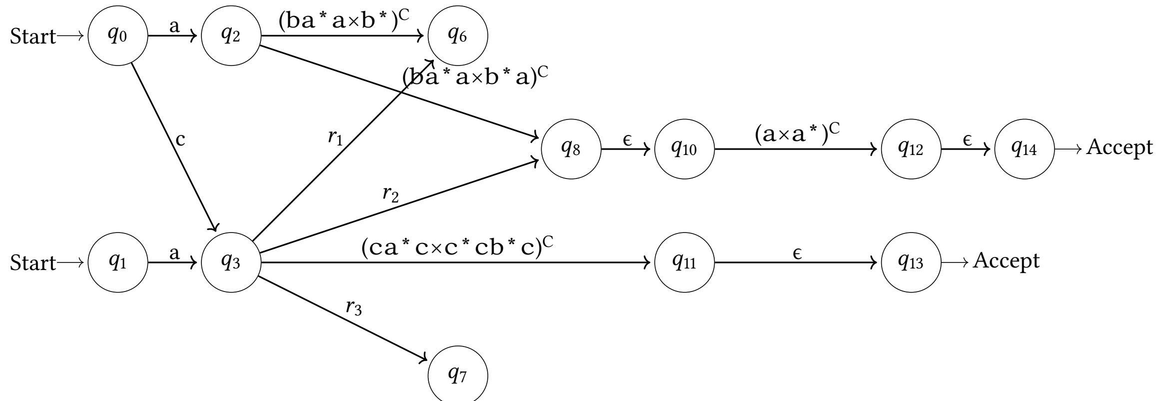
(b) *“border” H pairs and the corresponding REs*

(G, “border” H) pairs	RE	(G, “border” H) pairs	RE	(G, “border” H) pairs	RE
(q_2, q_6)	ba^*a	(q_3, q_6)	aa^*a	(q_{10}, q_6)	$\varnothing$
(q_2, q_7)	$\varnothing$	(q_3, q_7)	ca^*c	(q_{10}, q_7)	$\varnothing$
(q_2, q_8)	$\varnothing$	(q_3, q_8)	$\varnothing$	(q_{10}, q_8)	$\varnothing$
(q_2, q_{11})	$\varnothing$	(q_3, q_{11})	$\varnothing$	(q_{10}, q_{11})	$\varnothing$
(q_2, q_{12})	$\varnothing$	(q_3, q_{12})	$\varnothing$	(q_{10}, q_{12})	a

(c) *(G, “border” H) pairs and the corresponding REs*

(G, H, H)	RCE	(G, H, H)	RCE	(G, H, H)	RCE
(q_3, q_6, q_6)	$(aa^*a{\times}b^*)^C$	(q_2, q_6, q_6)	$(ba^*a{\times}b^*)^C$	(q_{10}, q_{12}, q_{12})	$(a{\times}a^*)^C$
(q_3, q_6, q_8)	$(aa^*a{\times}b^*a)^C$	(q_2, q_6, q_8)	$(ba^*a{\times}b^*a)^C$		
(q_3, q_7, q_6)	$(ca^*c{\times}c^*bb^*)^C$				
(q_3, q_7, q_7)	$(ca^*c{\times}c^*)^C$				
(q_3, q_7, q_8)	$(ca^*c{\times}c^*bb^*a)^C$				
(q_3, q_7, q_{11})	$(ca^*c{\times}c^*cb^*c)^C$				

(d) *(G, H_1, H_2) tuples and the corresponding RCEs*



(e) *The resulting intermediate representation to be fed in the FSA-to-RE conversion;* $r_1 = r_{(q_3,q_6,q_6)} + r_{(q_3,q_7,q_6)}$; $r_2 = r_{(q_3,q_6,q_8)} + r_{(q_3,q_7,q_8)}$; $r_3 = r_{(q_3,q_6,q_7)} + r_{(q_3,q_7,q_7)}$

Figure 27: *The example conversion of the copying mechanism to a copy expression*

2.5 Regular + copying languages

The previous section proves the equivalence between regular copying expressions and FSBMs in their descriptive powers. Given such equivalence, we define the class of regular+copying languages (RCLs) as follows.

Definition 18. *A language is in the class of regular+copying languages if and only if some regular copying expression denotes it, and equivalently if and only if some FSBMs recognizes it.*

Table 4 is a summary table on the closure properties of such language class surveyed so far.

Operations	Closed or not
union	✓
concatenation	✓
Kleene star	✓
homomorphism	✓
intersection with regular languages	✓
inverse homomorphism	? ✗
Copy(L)	✗

Table 4: *Surveyed closure properties of RCLs*

3 The linguistic relevance of the formal methods

Section 2 introduced the formal tools to define regular + copying set of languages and its closure properties. This section shows the linguistic relevance of two of its closure properties: closed under intersection with regular languages and closed under homomorphism.

Closed under intersection with regular languages That FSBM-recognizable languages are closed under intersection with regular languages is the first step in relating FSBMs as a formal model with theoretical phonological theory. Assume a natural language X imposes one requirement, saying X cannot have adjacent consonant clusters, which can be modeled by an FSA M_{*CC}. In addition, this language also requires phonological strings of specific forms to be reduplicated, which can be modeled by an FSBM M_{RED}. One at this moment can construct another FSBM $M_{RED\times *CC}$, which would yield to a set of strings without any consonant clusters and with the total identity of sub-strings in those forms. Not limited to consonant vowel distributions, phonotactics other than identity of sub-strings are also regular (Heinz, 2018), indicating almost all phonological well-formedness knowledge can be modeled by FSAs. When FSBMs intersect with FSAs computing those phonotactic restrictions, the resulting formalism is still an FSBM but not other grammar with higher computation power. Thus, FSBMs should be sufficient to compute natural language phonotactics once including recognizing surface sub-string identity.

This closure property is also closely related to implement Optimality Theory. Classic Optimality Theory (Smolensky and Prince, 1993) itself contains three basic components. Gen takes an input and generates a (infinite) set of possible output candidates. Con provides a set of constraints, which itself consists of two different types of constraints: *markedness* constraints encoding phonological well-formedness knowledge are restrictions solely on surface forms themselves; *faithfulness* constraints enforces a relation between input and output of the grammar, based on the correspondence among input and output symbols. Eval picks out the optimal candidate as the output according to the constraints. It is commonly accepted that finite-state methods can be used to model both *markedness* constraints and *faithfulness* constraints. Moreover, OT-wise, a

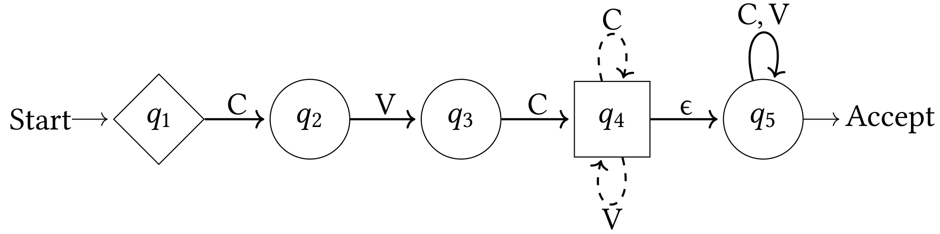
finite state machine can represent all possible surface candidates generated by Gen (Ellison, 1994; Eisner, 1997; Albro, 1998). The evaluation procedure Eval is achieved by an iterative intersection of the machine representing the set of candidates with every constraint.

Then, two possible directions of plugging the new machine to the implementation of Optimality Theory emerge to include reduplication. Firstly, if speakers' knowledge of phonological well-formedness ever contains surface sub-string identity, FSBMs can be used to model such requirement.[6] Secondly, following Albro (2000), FSBMs can be used as the representation of the candidate set enforcing reduplicative identity and iteratively intersect with finite-state machines representing the constraints. To fully achieve this goal, future work should consider developing an efficient algorithm that intersects complete-path FSBMs with *weighted* FSAs. To conclude, plugging FSBMs in the implementation of OT can either take constraints beyond regularity, or take candidate sets beyond regularity, or both. Now, it is inconclusive whether FSBMs are closed under intersection. If FSBMs are not closed under intersection, it seems at least one regularity assumption should be kept and taking both constraints and candidate sets beyond regular seems to be impossible.

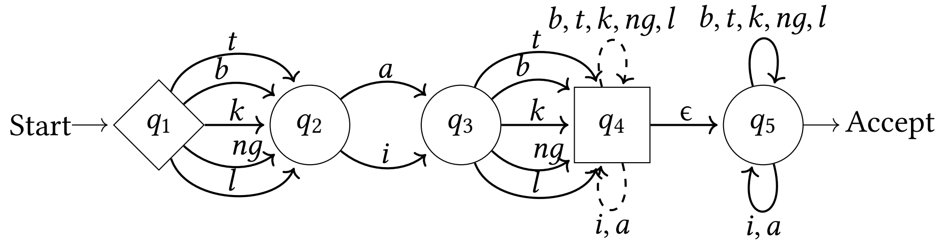
Closed under homomorphism The fact that FSBMs are closed under homomorphism allows theorists to perform analyses at a certain levels of abstraction of certain symbol representations. Consider two alphabets $\Sigma = \{b, t, k, ng, l, i, a\}$ and $\Delta = \{\mathrm{C}, \mathrm{V}\}$ with an alphabetic homomorphism h mapping every consonant (b, t, k, ng, l) to C while every vowel (i, a) to V. As illustrated by M_2 on alphabet Σ (Figure 10) and M_4 on alphabet Δ (Figure 28a), FSBM-definable patterns on Σ would be another FSBM-definable patterns on Δ.

It should be noted that it is highly possible that FSBM languages are not closed under inverse homomorphism. However, the set of regular languages is closed under inverse homomorphism. Given an FSA, one can let each symbol in the domain of the homomorphism to simulate the mapped string and construct a new machine for the inverse homomorphic language. Even if

[6] Again, we are not making any theoretical claims about whether phonology should include such requirement or not, but more from a more modeling perspective.



(a) *An FSBM* M_4 *on the alphabet* $\{C, V\}$ *such that* $L(M_4) = h(L(M_2))$ *with* M_2 *in Figure 10*



(b) *An FSBM* M_5 *on the alphabet* $\{b, t, k, ng, l\}$ *such that* $L(M_4)$ *recovers the pattern of* M_2 *in Figure 10*

Figure 28: *The linguistic relevance of the closure under homomorphism*

applying this construction to FSBMs does not produce the desired language, such a construction could be practical and useful as it keeps track of the identity relation among segments. For example, given the FSBM in 28a, the resulting machine under the described construction is shown in 28b. The previous machine accepts the surface forms with the prefix $C_1V_1C_2C_1V_1C_2-$. Then, the resulting machine keeps the identity relation in the previous machine and never recognizes the concatenation of two arbitrary CVCs, or $C_1V_1C_2C_3V_2C_4-$. In other words, the resulting machine only recognizes prefixes such as *babbab–* but never *babkak–*, nor *kablil–* which is in the inverse homomorphic image.

4 Discussion

The first two formal questions raised in Section 1 were answered in Section 2. This section attempts to answer the third question: how much can finite-state buffered machines and regular copying expressions explain or model the typology of reduplication? Or how does regular + copying languages align with natural language patterns? Under current definitions, regular + copying languages indeed show some limitations on the typology of reduplication. It only contains local reduplication with *two* adjacent, completely identical copies. It cannot handle non-local reduplication nor multiple reduplication. However, with some modifications, finite-state buffered machines or regular copying expressions as models for the whole typology should be possible. This section discusses some possible modifications without rigorously exploring what outcomes those modifications bring to the closure properties. We leave this area for future research.

4.1 Typology of reduplication

4.1.1 Non-local Reduplication

Non-local reduplication is when the surface phonological strings have non-adjacent copies, which incurs non-local correspondence among symbols. A more comprehensive typology and linguistic analysis on non-local reduplication can be found in Riggle (2004). Examples from Chukchee are shown in Table 5. As Bogoras (1969, 688, 690) described, the absolute form of a noun occupies the subject of an intransitive verb and the object of transitive verb and “dissyllabic words repeat the first syllable at the end of the word”.

Non-local reduplication: Chukchee absolutive singular (Bogoras, 1969, 60)		
Gloss	*Stem*	*absolute*
‘voice’	quli	quliqul
‘tears	mêrê	mêrêmêr
nute	‘land’	nute-nut

Table 5: *Chukchee absolutive singular: copies the first CVC sequence to the end of the word*

Currently, FSBMs are unable to capture non-local reduplication. The problem comes from the

requirement in which a B mode has to be directly followed by an E mode, and a filled buffer is not allowed in N mode. Then, the modification should be straightforward: FSBMs need to allow the buffer filled in N mode or another newly-defined memory holding mode, and match strings when needed. These revisions can be achieved by letting G pick out another area of a machine, as H does in the current FSBMs. Specifically, the transitions are unique among two G states in that they can only be used in B mode. Those transitions 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.

4.1.2 Multiple Reduplication

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 Thompson, a Salish language, are listed in Table 6.

multiple reduplication: Thompson (Broselow, 1983, 329)	
Gloss	Strings
calio	sil
DIM-calico	sí-sil
DIST-calico	sil-síl
DIST-DIM-calico	sil- sí-sil

Table 6: *Multiple reduplication in Thompson*

While the computational nature of multiple reduplication in natural language phonology and morphology remains an open question, the machines can easily be modified to include multiple copies of the same base form ($\{w^n \mid w \in \Sigma^*, n \in \mathbb{N}\}$). Given n, FSBMs can be granted with the freedom of not consuming buffered symbols in string matching E mode until the last nth sub-string is emitted. On the other hand, FSBMs cannot be easily modified to recognize the language containing copies of the copies ($\{w^{2^n} \mid w \in \Sigma^*, n \in \mathbb{N}\}$). To capture both cases in multiple

reduplication, modifications in RCEs are relatively easy. For multiple copies of the same base, $L(R^c) = \{w^n \mid w \in \Sigma^*, n \in \mathbb{N}\}$; as for copies of copy, R^c should no longer apply to only regular expressions, but can apply to a regular copying expression.

The copy language L_{ww} is midly context-sensitive. It remains inconclusive whether human languages contain w^{2^n}, a non-semilinear language in context-sensitive languages but not mildly context-sensitive. Joshi (1985) argues for a mildly context-sensitive upper bound for natural language syntax. For copying in the syntactic domain, Stabler (2004) differentiates a *generating grammar* (MCS grammars) from a *copying grammar* (beyond MCS). In generating grammars, both copies are generated step by step, while copying grammars copy some part of the structure in one step. Following Joshi (1985), Stabler (2004) examines different crossing dependencies in natural languages and argues for the generating account. On the other hand, Clark and Yoshinaka (2014) developed learning algorithms for a specific copying grammar, parallel multiple context-free grammars (pMCFGs), and claims that copying in "a full explanation of acquisition" may be equivalent as pMCFGs, not MCFGs (Clark and Yoshinaka, 2014, 13). One of the most suggestive pieces of evidence in natural languages is from Kobele (2006). It discusses the copying account of the serial verb construction in Yoruba relative clauses, which may itself include copied structures, thus challenging the MCS hypothesis. It would be advantageous for future research to undertake empirical and theoretical investigations regarding the nature of multiple reduplication in natural languages and the MCS hypothesis.

4.1.3 Reduplication with non-identical copies

In natural languages, further phonological processes complicate the issue by resulting in non-identical copies. As illustrated in Table 7, whole stems are copied in Javanese habitual forms. However, the first half of the reduplicated strings use [a] in place of the vowel in the last syllable.

In terms of recognizing non-identical copies, we can allow the machine to either store or empty not exactly the same input symbols, but mapped symbols according to some function f. [7]

[7] Thanks to a reviewer of the 18th SIGMORPHON workshop for suggesting this.

Non-identical copies: Javanese Habitual Repetitive (Yip, 1995, 249)		
Gloss	*root*	*Habitual Repetitive*
'remember'	eliŋ	elaŋ-eliŋ
'buy'	tuku	tuka-tuku
'bad'	eleʔ	elaʔ-eleʔ

Table 7: *Non-identical copies in Javanese*

Under this modification, the new automata would recognize $\{a^n b^n \mid n \in \mathbb{N}\}$ with $f(a) = b$ but still exclude string reversals. However, after this modification, the set of languages would also include $\{a^i b^j c^i d^j \mid i, j \geq 1\}$ with $f(a) = c, f(b) = d$.

5 Conclusion

This thesis provides new formal methods to compute unrestricted total reduplication on any regular languages, including the simplest copying language L_{ww} where w can be any arbitrary string of an alphabet. Two angles taken are 1) including copying as an expression operator and defining regular copying expressions; 2) proposing a new computational device: finite state buffered machines, which adds copying to regular languages. Eventually, regular copying expressions and finite-state buffered machines are proved to be equivalent in terms of the class of languages they describe. As a result, they introduce a new class of languages incomparable to context-free languages, named regular+copying languages.

This class of languages extends regular languages with *unbounded* copying but exclude non-reduplicative non-regular patterns: we hypothesize context-free string reversals are excluded since the buffer is queue-like. Meanwhile, the mildly context-sensitive Swiss-German cross-serial dependencies, abstracted as $\{a^i b^j c^i d^j \mid i, j \geq 1\}$, is also excluded, since the buffer works on the same alphabet as the input tape and only matches *identical* sub-strings.

In this thesis, we also surveyed the closure properties. The regular + copying 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}. We conjecture this class is not closed under inverse homomorphism, given it cannot recover the possibility of non-identity among corresponded segments when the mapping is many-to-one (and the inverse homomorphic image is one-to-many). Future works can look more into providing a pumping lemma in FSBMs to better understand what languages do not belong to regular + copying languages.

Automata-wise, following the breakthrough of sub-classes of 2-way FSTs in Dolatian and Heinz (2018a,b, 2019, 2020), which successfully capture unbounded copying as *function*s while exclude mirror image mappings, finite-state buffered machines successfully capture the total-reduplicated stringsets while exclude string reversals. One potential direction of future research is to connect FSBM with the studied 2-way FST account. We briefly mention two possibilities following this

line. Firstly, a comparison between the characterized languages in this thesis and the image of functions in Dolatian and Heinz (2020) should be carried out to build the bridges. Secondly, one can add another tape for output strings and extend FSBMs as acceptors to finite-state buffered transducers (FSBTs). The morphological analysis ($ww \rightarrow w$) problem is claimed to be difficult for 2-way FSTs, since deterministic 2-way FSTs are not invertible. Our intuition is FSBTs would help solve this issue: after reading the first w in input and buffering the string in memory, the machine can output ϵ for each matched symbol when transiting in between H states. Again, a more detailed and rigorous analysis should be conducted.

Another promising area of research is to extend Primitive Optimality Theory (Eisner, 1997; Albro, 1998) as Albro did. Albro (2000) maintains weighted finite state machine as the constraints while represents sets of candidates using multiple context-free Grammars to enforce base-reduplicant correspondence (McCarthy and Prince, 1995). Then, the goal here is to computationally implement reduplication in finite-state buffered machines without the full power of mildly context-sensitive languages, compared to the employed multiple context-free languages in Albro (2000, 2005). To fully achieve this goal, an efficient algorithm that intersects complete-path FSBMs with *weighted* FSAs is necessary.

Last but not least, as discussed in Section 4, the current class of languages excludes non-adjacent copies, multiple reduplication and reduplication with non-identical copies, which are attested in natural languages. Investigations on how to modify corresponding models and what changes those modifications bring should be the next step to complete the typology.

References

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

Albro, D. M. (2000). Taking primitive Optimality Theory beyond the finite state. In *Proceedings of the Fifth Workshop of the ACL Special Interest Group in Computational Phonology*, pages 57–67, Centre Universitaire, Luxembourg. International Committee on Computational Linguistics.

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

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

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

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

Beesley, K. R. and Karttunen, L. (2003). *Finite State Morphology.* CSLI Studies in Computational Linguistics. CSLI Publications.

Bogoras, W. (1969). *Chukchee*, volume 40 of *Bureau of American ethnology bulletin:vol 40. Handbook of American Indian languages, Part 2.* Government Printing Office, Washington.

Broselow, E. (1983). Salish double reduplications: Subjacency in morphology. *Natural Language Linguistic Theory*, 1:317–346.

Chandlee, J. (2014). *Strictly local phonological processes.* PhD thesis, University of Delaware.

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

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

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

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2(2):137–167.

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

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

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

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

Dolatian, H. and Heinz, J. (2018a). Learning reduplication with 2-way finite-state transducers. In Unold, O., Dyrka, W., and Wieczorek, W., editors, *Proceedings of the 14th International Conference on Grammatical Inference*, volume 93 of *Proceedings of Machine Learning Research*, pages 67–80. PMLR.

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

Dolatian, H. and Heinz, J. (2019). Redtyp: A database of reduplication with computational models. In *Proceedings of the Society for Computation in Linguistics*, volume 2. Article 3.

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

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

Ellison, T. M. (1994). Phonological derivation in optimality theory. In *Proceedings of the 15th Conference on Computational Linguistics - Volume 2*, COLING '94, page 1007–1013, USA. Association for Computational Linguistics.

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

Graf, T. (2013). *Local and Transderivational Constraints in Syntax and Semantics*. PhD thesis, UCLA.

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

Healey, P. M. (1960). *An Agta Grammar*. Bureau of Printing, Manila.

Heinz, J. (2007). *The Inductive Learning of Phonotactic Patterns*. PhD thesis, University of California, Los Angeles.

Heinz, J. (2018). The computational nature of phonological generalizations. In Hyman, L. and Plank, F., editors, *Phonological Typology*, Phonetics and Phonology, chapter 5, pages 126–195. De Gruyter Mouton.

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

Hopcroft, J. E. and Ullman, J. D. (1979). Introduction to automata theory, languages, and computation. *Addison-Welsey, NY.*

Hulden, M. (2009). *Finite-state Machine Construction Methods and Algorithms for Phonology and Morphology*. PhD thesis, University of Arizona, Tucson, USA.

Inkelas, S. (2008). The dual theory of reduplication. 46(2):351–401.

Inkelas, S. and Zoll, C. (2005). *Reduplication: Doubling in morphology*, volume 106. Cambridge University Press.

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

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

Johnson, C. D. (1972). Formal aspects of phonological description.

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

Joshi, A. K., Shanker, K. V., and Weir, D. (1990). The convergence of mildly context-sensitive grammar formalisms. *Technical Reports (CIS).*

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

Kobele, G. M. (2006). *Generating Copies: An investigation into structural identity in language and grammar.* PhD thesis, University of California, Los Angeles.

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

Marcus, G. F., Fernandes, K. J., and Johnson, S. P. (2007). Infant rule learning facilitated by speech. *Psychological Science*, 18(5):387–391. PMID: 17576276.

Marcus, G. F., Vijayan, S., Rao, S. B., and Vishton, P. M. (1999). Rule learning by seven-month-old infants. *Science*, 283(5398):77–80.

McCarthy, J. J. and Prince, A. S. (1995). Faithfulness and reduplicative identity.

McCollum, A. G., Baković, E., Mai, A., and Meinhardt, E. (2020). Unbounded circumambient patterns in segmental phonology. *Phonology*, 37:215 – 255.

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

Moreton, E., Prickett, B., Pertsova, K., Fennell, J., Pater, J., , and Sanders, L. (2021). Learning reduplication, but not syllable reversal. In Bennett, R., Bibbs, R., Brinkerhoff, M. L., Max J. Kaplan, S. R., Rysling, A., Handel, N. V., and Cavallaro, M. W., editors, *Supplemental Proceedings of the 2020 Annual Meeting on Phonology*.

Nishida, T. and Seki, S. (2000). Grouped partial etol systems and parallel multiple context-free grammars. *Theoretical Computer Science*, 246(1):131–150.

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

Riggle, J. (2004). Nonlocal reduplication. In *Proceedings of the 34th Meeting of the North-East Linguistics Society (NELS 34)*, page 485–496, USA. GLSA, University of Massachusetts.

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

Rubino, C. (2013). Reduplication. In Dryer, M. S. and Haspelmath, M., editors, *The World Atlas of Language Structures Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Seki, H., Matsumura, T., Fujii, M., and Kasami, T. (1991). On multiple context-free grammars. *Theoretical Computer Science*, 88(2):191–229.

Shepherdson, J. C. (1959). The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3(2):198–200.

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

Simon, I. (1975). Piecewise testable events. In Brakhage, H., editor, *Automata Theory and Formal Languages*, pages 214–222, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

Smolensky, P. and Prince, A. (1993). Optimality theory: Constraint interaction in generative grammar. *Optimality Theory in phonology*, 3.

Stabler, E. (1997). Derivational minimalism. In Retoré, C., editor, *Logical Aspects of Computational Linguistics*, pages 68–95, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

Steedman, M. (1996). *Surface Structure and Interpretation*. MIT Press, Cambridge, MA, USA.

Waksler, R. (1999). Cross-linguistic evidence for morphological representation in the mental lexicon. *Brain and Language*, 68(1):68–74.

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

Yip, M. (1995). Repetition and its avoidance: The case of javanese. In *Proceedings of the South Western Optimality Theory workshop 1995.Arizona Phonology Conference Volume 5,*, pages 238—-262, Tucson, AZ. University of Arizona.

Zuraw, K. (1996). Floating phonotactics: Infixation and reduplication in tagalog loanwords. Master's thesis, UCLA.

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

A Mathematical preliminaries

Sets and set operations A *set* is a collection of distinct objects as its*elements*. That an object a is an element of a set A is denoted by $a \in \mathrm{A}$. The symbol $\notin$ denotes non-membership. The number of elements in a set A is called the cardinality of A, written $|\mathrm{A}|$. A set A is *finite* if and only if (iff) there exists some natural number k such that A has exactly k many elements, as $|\mathrm{A}| = k$. One finite set of special interests is the set with no elements, or the empty set $\varnothing$. It's easy to see $|\varnothing| = 0$. Conversely, if A contains infinitely many elements, A would be an *infinite* set and no such natural number k could describe the cardinality of A.

Different ways to describe a set can ultimately lead to the same set. For example, $\{2, 4, 6, 8\}$ and $\{x \,|\, x \text{ is even and } 0 < x < 10\}$ are different specifications but include exactly the same elements. Two sets A and B are equal, or A = B, iff they contain exactly the same elements. So $\{2, 4, 6, 8\}$ $= \{x \,|\, x \text{ is even and } 0 < x < 10\}$. Phrasing the equality of two sets in another way, A = B if two conditions holds: 1): for every x, if $x \in \mathrm{A}$, then $x \in \mathrm{B}$; 2): for every x, if $x \in \mathrm{B}$, then $x \in \mathrm{A}$. If every element of a set A is an element of a set B, then A is a *subset* of B, denoted as $\mathrm{A} \subseteq \mathrm{B}$. Thus, we see proving A = B is equivalent as proving $1) : \mathrm{A} \subseteq \mathrm{B}$ and $2) : \mathrm{B} \subseteq \mathrm{A}$. In nature, many proofs in this thesis are proving two sets are equal and relies on proving the subset relations in both directions. Moreover, based on the definition of the subset relation, a set can be a subset of itself. A is a subset of B and not equal to B, A is said to be a *proper subset* of B, written as $\mathrm{A} \subset \mathrm{B}$. Moreover, the usual operations of union, intersection, difference, Cartesian product and powerset are defined below.

$$\mathrm{A} \cup \mathrm{B} =_{def} \{x \,|\, x \in \mathrm{A} \text{ or } x \in \mathrm{B}\} \qquad \text{union}$$

$$\mathrm{A} \cap \mathrm{B} =_{def} \{x \,|\, x \in \mathrm{A} \text{ and } x \in \mathrm{B}\} \qquad \text{intersection}$$

$$\mathrm{A} - \mathrm{B} =_{def} \{x \,|\, x \in \mathrm{A} \text{ and } x \notin \mathrm{B}\} \qquad \text{difference}$$

$$\mathrm{A} \times \mathrm{B} =_{def} \{\langle a, b\rangle \,|\, a \in \mathrm{A} \text{ and } b \in \mathrm{B}\} \qquad \text{Cartesian product}$$

$$\wp(\mathrm{A}) =_{def} \{\mathrm{X} \,|\, \mathrm{X} \subseteq \mathrm{A}\} \qquad \text{powerset}$$

Relations and functions A *binary relation* on sets A and B is a subset of $A \times B$. A function f from its domain A to its co-domain B written as $f : A \rightarrow B$ is a relation $f \subseteq A \times B$ such that if $\langle a, b \rangle \in f$, then there is no $b' \in B$ distinct from b such that $\langle a, b' \rangle \in f$. A *total* function gets every element in its domain A mapped to something in B.

Alphabets, strings, languages An *alphabet* is a non-empty finite set of symbols, denoted by Σ. A *string* over an alphabet Σ is a finite sequence of symbols from Σ. The *length* of a string w ($|w|$) is the number of symbols in w. The empty string ϵ contains zero symbols and thus $|\epsilon| = 0$.

A *language* or a *stringset* is a set of strings. As usual, Σ^* denotes the language which includes all possible strings over the alphabet Σ. Σ^n is the language with all possible strings of length n. $\Sigma^{\leq n}$ is the language that include all possible strings of length less than or equal to n. For example, if $\Sigma = \{0, 1\}$, $\Sigma^2 = \{00, 11, 01, 10\}$, $\Sigma^{\leq 2} = \{\epsilon, 0, 1, 00, 11, 01, 10\}$ and $\Sigma^* = \{\epsilon, 0, 1, 00, 11, 01, 10, 000 \ldots\}$.

B A proof of Theorem 2: closure under the intersection with regular languages

If L_1 is a language recognized by a complete-path FSBM $M_1 = \langle Q_1, \Sigma, I_1, F_1, \delta_1, G_1, H_1 \rangle$, and L_2 is a language recognized by an FSA $M_2 = \langle Q_2, \Sigma, I_2, F_2, \delta_2 \rangle$, then $L_1 \cap L_2$ is a language recognizable by another FSBM.

Proof. by cross-product construction: construct such M to recognize $L_1 \cap L_2$ where $M = \langle Q, \Sigma, I, F, \delta, G, H \rangle$ with:

- $Q = Q_1 \times Q_2$
- $I = I_1 \times I_2$
- $F = F_1 \times F_2$
- $G = G_1 \times Q_2$
- $H = H_1 \times Q_2$
- $((q_1, q_1'), x, (q_2, q_2')) \in \delta$ iff $(q_1, x, q_2) \in \delta_1$ and $(q_1', x, q_2') \in \delta_2$

$\because G_1 \cap H_1 = \varnothing$
$\therefore (G_1 \times Q_2) \cap (H_1 \times Q_2) = \varnothing$
$\therefore G \cap H = \varnothing$
Then, we need to show $L(M) = L_1 \cap L_2$. To show this, we need to show 1): $L(M) \subseteq L_1 \cap L_2$; 2): $L_1 \cap L_2 \subseteq L(M)$.

1. $\boxed{L(M) \subseteq L_1 \cap L_2}$

   Assume $w = x_1 x_2 x_3 \ldots x_n \in L(M)$ for $n \geq 0$, N.T.S $w \in L_1 \cap L_2$. That's to show $w \in L_1$ and $w \in L_2$.

   Given $w \in L(M)$, there exists a sequence of configurations $D_0, D_1, D_2 \ldots D_m$ for some $m \geq 0$ such that

- D_0 is a starting configuration. That's $D_0 = (w, (p_0, q_0), \epsilon, \text{N})$ with $(p_0, q_0) \in I$.

  $\therefore p_0 \in I_1$, $q_0 \in I_2$.

- D_m is a accepting configuration. That's $D_m = (\epsilon, (p_m, q_m), \epsilon, \text{N})$ with $(p_m, q_m) \in F$.

  $\therefore p_m \in F_1$, $q_m \in F_2$.

- $\forall i \in \{0, 1, \ldots m\}, D_i \vdash_M D_{i+1}$

To show $w \in L_1$, N.T.S $\exists k \in \mathbb{N}$ such that there exists a sequence of configurations $A_0, A_1 \ldots A_k$ of M_1 such that $A_0 \vdash^*_{M_1} A_k$ in order to process w.

Similarly, to show $w \in L_2$, N.T.S $\exists l \in \mathbb{N}$ such that there exists a sequence of configurations $B_0, B_1 \ldots B_l$ such that $B_0 \vdash^*_{M_2} B_l$ in order to process w.

$\forall 0 \leq i \leq m, D_i = (u, (p_i, q_i), v, x)$ for some $u \in \Sigma^*, v \in \Sigma^*, x \in \{\text{N}, \text{B}, \text{E}\}$,

let $A_i = (u, p_i, v, x)$ and $B_i = (u, q_i)$.

W.T.S $A_0, A_1, A_2 \ldots A_m$ and $B_0, B_1, B_2 \ldots B_m$ are such sequences.

Let's focus on $A_0, A_1, A_2 \ldots A_m$ first. Under such construction, we know

- $A_0 = (w, p_0, \epsilon, \text{N})$ with $p_0 \in I_1$

  $\therefore A_0$ is a starting configuration for w in M_1

- $A_m = (\epsilon, p_m, \epsilon, \text{N})$ with $p_m \in F_1$

  $\therefore A_m$ is an accepting configuration for w in M_1

Then, we are left to show that $A_0 \vdash^*_{M_1} A_m$. That's $\forall 0 \leq i < m$, $A_i \vdash_{M_1} A_{i+1}$. There are six possible cases for $D_i \vdash_M D_{i+1}$.

For some $x \in \Sigma, u, v \in \Sigma^*$,

(a) $(xu, (p_i, q_i), \epsilon, \text{N}) \vdash_M (u, (p_{i+1}, q_{i+1}), \epsilon, \text{N})$.

$\therefore$ The following statements hold.

i. $((p_i, q_i), x, (p_{i+1}, q_{i+1})) \in \delta$

ii. at least one state of $(p_i, q_i), (p_{i+1}, q_{i+1}) \notin \mathrm{H}$

iii. $(p_i, q_i) \notin \mathrm{G}$

$\therefore (p_i, x, p_{i+1}) \in \delta_1$ and at least one of p_i, $p_{i+1} \notin \mathrm{H}_1$ and $p_i \notin \mathrm{G}_1$

$\therefore (xu, p_i, \epsilon, \textsc{n}) \vdash_{\mathrm{M}_1} (u, p_{i+1}, \epsilon, \textsc{n})$

$\therefore \mathrm{A}_i \vdash_{\mathrm{M}_1} \mathrm{A}_{i+1}$

(b) $(xu, (p_i, q_i), v, \textsc{b}) \vdash_{\mathrm{M}} (u, (p_{i+1}, q_{i+1}), vx, \textsc{b})$.

$\therefore$ The following statements hold.

i. $((p_i, q_i), x, (p_{i+1}, q_{i+1})) \in \delta$

ii. at least one state of $(p_i, q_i), (p_{i+1}, q_{i+1}) \notin \mathrm{H}$

iii. $(p_{i+1}, q_{i+1}) \notin \mathrm{G}$

$\therefore (p_i, x, p_{i+1}) \in \delta_1$ and $p_{i+1} \notin \mathrm{G}_1$ and at least one of p_i, $p_{i+1} \notin \mathrm{H}_1$

$\therefore (xu, p_i, v, \textsc{b}) \vdash_{\mathrm{M}_1} (u, p_{i+1}, vx, \textsc{b})$

$\therefore \mathrm{A}_i \vdash_{\mathrm{M}_1} \mathrm{A}_{i+1}$

(c) $(xu, (p_i, q_i), xv, \textsc{e}) \vdash_{\mathrm{M}} (u, (p_{i+1}, q_{i+1}), v, \textsc{e})$.

$\therefore ((p_i, q_i), x, (p_{i+1}, q_{i+1})) \in \delta$, $(p_i, q_i) \in \mathrm{H}$ and $(p_{i+1}, q_{i+1}) \in \mathrm{H}$

$\therefore (p_i, x, p_{i+1}) \in \delta_1$, $p_i \in \mathrm{H}_1$ and $p_{i+1} \in \mathrm{H}_1$.

$\therefore (xu, p_i, xv, \textsc{e}) \vdash_{\mathrm{M}_1} (u, p_{i+1}, v, \textsc{e})$.

$\therefore \mathrm{A}_i \vdash_{\mathrm{M}_1} \mathrm{A}_{i+1}$

(d) $(u, (p_i, q_i), \epsilon, \textsc{n}) \vdash_{\mathrm{M}} (u, (p_{i+1}, q_{i+1}), \epsilon, \textsc{b})$.

$\therefore (p_i, q_i) = (p_{i+1}, q_{i+1}) \in \mathrm{G}$

$\therefore p_i = p_{i+1} \in \mathrm{G}_1$

$\therefore (u, p_i, \epsilon, \textsc{n}) \vdash_{\mathrm{M}_1} (u, p_{i+1}, \epsilon, \textsc{b})$

$\therefore \mathrm{A}_i \vdash_{\mathrm{M}_1} \mathrm{A}_{i+1}$

(e) $(u, (p_i, q_i), v, \textsc{b}) \vdash_{\mathrm{M}} (u, (p_{i+1}, q_{i+1}), v, \textsc{e})$.

$\therefore (p_i, q_i) = (p_{i+1}, q_{i+1}) \in \mathrm{H}$

$\therefore p_i = p_{i+1} \in \mathrm{H}_1$

$\therefore (u, p_i, v, \text{B}) \vdash_{M_1} (u, p_{i+1}, v, \text{E})$

$\therefore A_i \vdash_{M_1} A_{i+1}$

(f) $(u, (p_i, q_i), \epsilon, \text{E}) \vdash_M (u, (p_{i+1}, q_{i+1}), \epsilon, \text{N})$.

$\therefore (p_i, q_i) = (p_{i+1}, q_{i+1}) \in H$

$\therefore p_i = p_{i+1} \in H_1$

$\therefore (u, p_i, \epsilon, \text{E}) \vdash_{M_1} (u, p_{i+1}, \epsilon, \text{N})$

$\therefore A_i \vdash_{M_1} A_{i+1}$

Therefore, we see every transition among the constructed $A_0, A_1, A_2....A_m$ sequence is valid in M_1. We can conclude that $w \in L_1$.

Similarly, for $B_0, B_1, B_2....B_m$, we know

- $B_0 = (w, q_0)$ with $q_0 \in I_2$

  $\therefore B_0$ is a starting configuration for w in M_2

- $B_m = (\epsilon, q_m)$ with $q_m \in F_2$

  $\therefore B_m$ is an accepting configuration for w in M_2

Then, we are left to show that $B_0 \vdash^*_{M_2} B_m$. That's to show, in the sequence $B_0, B_1, B_2 \ldots B_m$, a configuration B_i goes into the next configuration B_{i+1} in one step or zero steps. In other words, $\forall 0 \leq i < m$, $B_i \vdash_{M_2} B_{i+1}$ or $B_i = B_{i+1}$.

For some $x \in \Sigma, u, v \in \Sigma^*$,

(a) There are three possible cases for $D_i \vdash_M D_{i+1}$ that makes $B_i \vdash_{M_2} B_{i+1}$

- $(xu, (p_i, q_i), \epsilon, \text{N}) \vdash_M (u, (p_{i+1}, q_{i+1}), \epsilon, \text{N})$.

  $\therefore ((p_i, q_i), x, (p_{i+1}, q_{i+1})) \in \delta$

  $\therefore (q_i, x, q_{i+1}) \in \delta_2$ with $B_i = (xu, q_i)$, $B_{i+1} = (u, q_{i+1})$

- $(xu, (p_i, q_i), v, \textsc{b}) \vdash_M (u, (p_{i+1}, q_{i+1}), vx, \textsc{b})$.

  $\therefore ((p_i, q_i), x, (p_{i+1}, q_{i+1})) \in \delta$

  $\therefore (q_i, x, q_{i+1}) \in \delta_2$ with $B_i = (xu, q_i)$, $B_{i+1} = (u, q_{i+1})$

- $(xu, (p_i, q_i), xv, \textsc{e}) \vdash_M (u, (p_{i+1}, q_{i+1}), v, \textsc{e})$.

  $\therefore ((p_i, q_i), x, (p_{i+1}, q_{i+1})) \in \delta$

  $\therefore (q_i, x, q_{i+1}) \in \delta_2$ with $B_i = (xu, q_i)$, $B_{i+1} = (u, q_{i+1})$

$\therefore (xu, q_i) \vdash_{M_2} (u, q_{i+1})$

$\therefore B_i \vdash_{M_2} B_{i+1}$

(b) There are three possible cases for $D_i \vdash_M D_{i+1}$ that makes $B_i = B_{i+1}$

- $(u, (p_i, q_i), \epsilon, \textsc{n}) \vdash_M (u, (p_{i+1}, q_{i+1}), \epsilon, \textsc{b})$.

  $\therefore (p_i, q_i) = (p_{i+1}, q_{i+1})$ with $B_i = (u, q_i)$, $B_{i+1} = (u, q_{i+1})$

- $(u, (p_i, q_i), v, \textsc{b}) \vdash_M (u, (p_{i+1}, q_{i+1}), v, \textsc{e})$.

  $\therefore (p_i, q_i) = (p_{i+1}, q_{i+1})$ with $B_i = (u, q_i)$, $B_{i+1} = (u, q_{i+1})$

- $(u, (p_i, q_i), \epsilon, \textsc{e}) \vdash_M (u, (p_{i+1}, q_{i+1}), \epsilon, \textsc{n})$.

  $\therefore (p_i, q_i) = (p_{i+1}, q_{i+1})$ with $B_i = (u, q_i)$, $B_{i+1} = (u, q_{i+1})$

$\therefore (u, q_i) = (u, q_{i+1})$

$\therefore B_i = B_{i+1}$

Therefore, we see every transition among the constructed $B_0, B_1, B_2....B_m$ sequence is valid in M_2. We can conclude that $w \in L_2$.

2. $\boxed{L_1 \cap L_2 \subseteq L(M)}$

Assume $w = x_1 x_2 x_3 ... x_{\mathbf{n}} \in L_1, L_2$ for $n \geq 0$, N.T.S $w \in L(M)$.

$\because w \in L_1$

$\therefore$ there exists a sequence of configurations $A_0, A_1, A_2 .... A_m$ with

- $A_0 = (w, p_0, \epsilon, \text{N})$ with $p_0 \in I_1$
- $A_m = (\epsilon, p_m, \epsilon, \text{N})$ with $p_m \in F_1$
- $\forall 0 \leq i < m$, $A_i \vdash_{M_1} A_{i+1}$

$\because w \in L_2$

$\therefore$ there exists a sequence of configurations $B_0, B_1, B_2 .... B_{\mathbf{n}}$. In other words, there would be $(n+1)$ configurations in this sequence for this length n string.

- $B_0 = (w, q_0)$ with $q_0 \in I_2$
- $B_n = (\epsilon, q_n)$ with $q_n \in F_2$
- $\forall 0 \leq i < n$, $B_i \vdash_{M_2} B_{i+1}$

Then, to show $w \in L(M)$, N.T.S that for some $k \geq 0$, there is a sequence of configurations $D_0, D_1 .... D_k$ in M such that $D_0 \vdash_M^* D_k$ in order to process w.

- Let $D_0 = (w, (p_0, q_0), \epsilon, \text{N})$.

  $\therefore (p_0, q_0) \in I$

  $\therefore D_0$ is a starting configuration for w in M

- $\forall 0 \leq i < m$, for some $x \in \Sigma$, $u, v \in \Sigma^*$

  (a) If $A_i = (xu, p_i, \epsilon, \text{N})$, $A_{i+1} = (u, p_{i+1}, \epsilon, \text{N})$,

  then $\exists\ 0 \leq j < n$, $B_j = (xu, q_j)$, $B_{j+1} = (u, q_{j+1})$,

  then let $D_i = (xu, (p_i, q_j), \epsilon, \text{N})$ and $D_{i+1} = (u, (p_{i+1}, q_{j+1}), \epsilon, \text{N})$.

  $\because A_i \vdash_{M_1} A_{i+1}$

  $\therefore (p_i, x, p_{i+1}) \in \delta_1$, and at least one state of $p_i, p_{i+1} \notin H_1$, and $p_i \notin G_1$

$\because B_j \vdash_{M_2} B_{j+1}$

$\therefore (q_j, x, q_{j+1}) \in \delta_2$

$\therefore ((p_i, q_j), x, (p_{i+1}, q_{j+1})) \in \delta$, and at least one state of $(p_i, q_j), (p_{i+1}, q_{j+1}) \notin H$, and $(p_i, q_j) \notin G$

$\therefore D_i \vdash_M D_{i+1}$

(b) If $A_i = (xu, p_i, v, \text{B})$, $A_{i+1} = (u, p_{i+1}, vx, \text{B})$,

then $\exists\ 0 \leq j < n$, $B_j = (xu, q_j)$, $B_{j+1} = (u, q_{j+1})$,

then let $D_i = (xu, (p_i, q_j), v, \text{B})$ and $D_{i+1} = (u, (p_{i+1}, q_{j+1}), vx, \text{B})$.

$\because A_i \vdash_{M_1} A_{i+1}$

$\therefore (p_i, x, p_{i+1}) \in \delta_1$, and at least one state of $p_i, p_{i+1} \notin H_1$, and $p_{i+1} \notin G_1$

$\because B_j \vdash_{M_2} B_{j+1}$

$\therefore (q_j, x, q_{j+1}) \in \delta_2$

$\therefore ((p_i, q_j), x, (p_{i+1}, q_{j+1})) \in \delta$, and at least one state of $(p_i, q_j), (p_{i+1}, q_{j+1}) \notin H$, and $(p_{i+1}, q_{j+1}) \notin G$

$\therefore D_i \vdash_M D_{i+1}$

(c) If $A_i = (xu, p_i, xv, \text{E})$, $A_{i+1} = (u, p_{i+1}, v, \text{E})$,

then $\exists\ 0 \leq j < n$, $B_j = (xu, q_j)$, $B_{j+1} = (u, q_{j+1})$,

then let $D_i = (xu, (p_i, q_j), xv, \text{E})$ and $D_{i+1} = (u, (p_{i+1}, q_{j+1}), v, \text{E})$.

$\because A_i \vdash_{M_1} A_{i+1}$

$\therefore (p_i, x, p_{i+1}) \in \delta_1$, and $p_i \in H_1, p_{i+1} \in H_1$

$\because B_j \vdash_{M_2} B_{j+1}$

$\therefore (q_j, x, q_{j+1}) \in \delta_2$

$\therefore ((p_i, q_j), x, (p_{i+1}, q_{j+1})) \in \delta$ and $(p_i, q_j) \in H, (p_{i+1}, q_{j+1}) \in H$

$\therefore D_i \vdash_M D_{i+1}$

(d) If $A_i = (xu, p_i, \epsilon, \text{N})$, $A_{i+1} = (xu, p_{i+1}, \epsilon, \text{B})$,

then $\exists\ 0 \le j < n$, $B_j = (xu, q_j)$, $B_{j+1} = (u, q_{j+1})$,

then let $D_i = (xu, (p_i, q_j), \epsilon, \text{N})$ and $D_{i+1} = (xu, (p_{i+1}, q_j), \epsilon, \text{B})$.

$\because A_i \vdash_{M_1} A_{i+1}$

$\therefore p_i = p_{i+1} \in G_1$

$\therefore (p_i, q_j) = (p_{i+1}, q_j) \in G$

$\therefore D_i \vdash_M D_{i+1}$

(e) If $A_i = (xu, p_i, v, \text{B})$, $A_{i+1} = (xu, p_{i+1}, v, \text{E})$,

then $\exists\ 0 \le j < n$, $B_j = (xu, q_j)$, $B_{j+1} = (u, q_{j+1})$,

then let $D_i = (xu, (p_i, q_j), v, \text{B})$ and $D_{i+1} = (xu, (p_{i+1}, q_j), v, \text{E})$.

$\because A_i \vdash_{M_1} A_{i+1}$

$\therefore p_i = p_{i+1} \in G_1$

$\therefore (p_i, q_j) = (p_{i+1}, q_j) \in G$

$\therefore D_i \vdash_M D_{i+1}$

(f) If $A_i = (xu, p_i, \epsilon, \text{E})$, $A_{i+1} = (xu, p_{i+1}, \epsilon, \text{N})$,

then $\exists\ 0 \le j < n$, $B_j = (xu, q_j)$, $B_{j+1} = (u, q_{j+1})$,

then let $D_i = (xu, (p_i, q_j), \epsilon, \text{E})$ and $D_{i+1} = (u, (p_{i+1}, q_j), \epsilon, \text{N})$.

$\because A_i \vdash_{M_1} A_{i+1}$

$\therefore p_i = p_{i+1}$, and $p_i \in H_1$

$\therefore (p_i, q_j) = (p_{i+1}, q_j)$ and $(p_i, q_j) \in H$

$\therefore D_i \vdash_M D_{i+1}$

- Let $D_m = (\epsilon, (p_m, q_n), \epsilon, \text{N})$

  $\therefore (p_m, q_n) \in F$

  $\therefore D_m$ is an accepting configuration in M

The above construction uses those configurations for w in the FSBM M_1 as main ingredients. It enforces $k = m$. That's we have (m+1) configurations to process w in the intersection FSBM M. Moreover, it picks out the corresponding configurations in the FSA M_2 by matching input strings, which yields to unique configurations or pairs of configurations in the B sequences.

Under such construction, we can start at D_0 and follow the sequence of configurations $D_0, D_1, \ldots D_m$ to process w in M.

□

C A proof of Theorem 5: the construction for the copying expression operator

For any regular copying expression $R = R_1^C$ with R_1 as a regular expression. Then, there's an FSBM that accepts only $L(R)$.

Proof. Assume there's an FSA $M_0 = \langle Q', \Sigma, I', F', \delta' \rangle$ that recognizes R_1. Let $M = \langle Q, \Sigma, I, F, \delta, G, H \rangle$ with

- $Q = Q' \cup \{q_0, q_f\}$
- $G = I = \{q_0\}$
- $H = F = \{q_f\}$
- $\delta = \delta' \cup \{(q_f, x, q_f) \mid x \in \Sigma\} \cup \{(q_0, \epsilon, q) \mid q \in I'\} \cup \{(q, \epsilon, q_f) \mid q \in F'\}$

To show L(M) = L(R), we need to show 1): $L(M) \subseteq L(R)$; 2): $L(R) \subseteq L(M)$.

- $\boxed{L(R) \subseteq L(M)}$

  For any $s \in L(R)$, $\exists w = x_1 x_2 x_3 \dots x_n \in L(R_1)$ such that $s = ww$

  $\because w \in L(R_1) = L(M_0)$

  $\therefore$ there exists a sequence of configurations $A_0, A_1, A_2 \dots A_n$ to process w in M_0.

  Then, $A_0 = (w, p_0)$ for some $p_0 \in I$

  $A_n = (\epsilon, p_f)$ for some $p_f \in F$

  $\forall 1 \leq i < n$, $A_i = (x_i x_{i+1} \dots x_n, p_i) \vdash_{M_0} A_{i+1} = (x_{i+1} \dots x_n, p_{i+1})$ with $(p_i, x_i, p_{i+1}) \in \delta'$

  Let $D_0 = (ww, q_0, \epsilon, \text{N})$. It's easy to see D_0 is a starting configuration of s in M.

  $\because q_0 \in G$

  $\therefore D_1 = (ww, q_0, \epsilon, \text{B})$

$\because (q_0, \epsilon, p_0) \in \delta$

$\therefore D_2 = (ww, p_0, \epsilon, \text{B})$

By going through $p_0, p_1, p_2 \ldots p_n$ in $A_0, A_1, A_2 \ldots A_n$, we can reach $D_{2+n+1} = (w, p_n, w, \text{B})$

$\because (p_n, \epsilon, q_f) \in \delta$

$\therefore D_{2+n+2} = (w, q_f, w, \text{B})$

$\because q_f \in H$

$\therefore D_{2+n+2} = (w, q_f, w, \text{E})$

$\because \forall x \in \Sigma, (p_f, x, p_f) \in \delta$

$\therefore \forall y$ appears in w, $(p_f, y, p_f) \in \delta$

$\therefore$ By following the loops on p_f, we can reach $D_{2+n+2+n} = (\epsilon, q_f, \epsilon, \text{E})$

$\because q_f \in H$

$\therefore$ we can reach $D_{2+n+2+n+1} = (\epsilon, q_f, \epsilon, \text{N})$, which is an accepting configuration

$\therefore s = ww \in L(M)$

- $\boxed{L(M) \subseteq L(R)}$

  For any $s \in L(M)$, N.T.S $s \in L(R)$.

  $\because s \in L(M)$

  $\therefore$ there exists a sequence of configurations $D_0, D_1, D_2 \ldots D_m$ such that 1): $D_0 = (s, q_0, \epsilon, \text{N})$; 2): $D_m = (\epsilon, q_f, \epsilon, \text{N})$; 3): $\forall 1 \leq j < m, D_j \vdash_M D_{j+1}$

  $\because q_0 \in G$

  $\therefore D_1 = (s, q_0, \epsilon, \text{B})$

  $\because$ the only transitions from q_0 are (q_0, ϵ, p_0) with some $p_0 \in I'$

  $\therefore D_2 = (s, p_0, \epsilon, \text{B})$

  $\because D_m = (\epsilon, q_f, \epsilon, \text{N})$ and only q_f is the H state

  $\therefore D_{m-1} = (\epsilon, q_f, \epsilon, \text{E})$

  $\therefore \exists 2 \leq k < m - 1$ such that $\exists w \in \Sigma^*$ with $D_{k-1} = (w, q_f, w, \text{B})$ and $D_k = (w, q_f, w, \text{E})$

  $\because$ the only transitions to q_f are (p_f, ϵ, q_f) with some $p_f \in F'$

  $\therefore D_{k-2} = (w, p_f, w, \text{B})$

From D_2 to D_{k-2}, the machine M can go from p_0 to p_f for w and the only transitions from p_0 to p_f is by taking the previous transitions in δ', which means $w \in L(M_0) = L(R_1)$ and $s = ww$.

$\therefore s \in L(R)$

$\square$