Computing and classifying reduplication with 2-way finite-state transducers

Hossep Dolatian and Jeffrey Heinz
Department of Linguistics
Institute of Advanced Computational Science
Stony Brook University

ABSTRACT

This article describes a novel approach to the computational modeling of reduplication. Reduplication is often treated as a stumbling block within finite-state treatments of morphology because they cannot adequately capture the productivity of unbounded copying and because they cannot describe bounded copying without a large increase in the number of states. We provide a comprehensive typology of reduplicative processes and show that an understudied type of finite-state machines, two-way deterministic finite-state transducers (2-way D-FSTs), captures virtually all of them. Furthermore, the 2-way FSTs have few states, are in practice easy to design and debug, and are linguistically motivated in terms of the transducer's origin semantics or segment alignment. Most of these processes, and their corresponding 2-way D-FSTs are available in an online database of reduplication (RedTyp). We classify these 2-way D-FSTs according to the concatenation of known subclasses of regular relations and show that the majority fall into the Concatenated Output Strictly Local (C-OSL) class. Other cases require higher subclasses but are still definable by 2-way D-FSTs.

Keywords: reduplication, two-way finite state transducer, finite state morphology

1 INTRODUCTION

Reduplication is a cross-linguistically common word-formation process involving copying. Given a word, reduplication can copy either a

bounded (1a) or unbounded (1b) number of segments. The symbol ∼ marks the boundary between the two copies.

(1) a. **Partial Reduplication** *Agta* (Moravcsik 1978, 311)
takki→tak∼takki 'leg'→'legs'

b. **Total Reduplication** *Indonesian* (Cohn 1989, 308)
buku→buku∼buku 'book'→'books'

Reduplication is used in the majority of the world's languages, and total reduplication is more common than partial reduplication. The World Atlas of Language Structures (WALS) database documents that 278 out of 368 (75%) languages have total and partial reduplication (Rubino 2013). 35 additional languages (10%) use only total reduplication, not partial reduplication. The 55 (15%) remaining languages do not have productive reduplication, but this figure is debatable.[1] Therefore, developing analyzable and efficient computational models of reduplication is important.

Although reduplication is well-studied, it is a computationally challenging process (Sproat 1992). In computational linguistics, most morphological and phonological processes are analyzed with the finite-state calculus in terms of rational languages and transductions (Kaplan and Kay 1994; Beesley and Karttunen 2003). However, reduplicative processes cannot be easily modeled with those same finite-state systems. For total reduplication, this is because those finite-state systems cannot express unbounded copying in the first place (Culy 1985). As for partial reduplication, those finite-state systems are often described as unwieldy because of the state explosion that partial reduplication causes (Roark and Sproat 2007, 54). §2 of this article explains why reduplication is computationally challenging while reviewing previous computational approaches to this linguistic phenomenon.

In this context, the primary contribution of this article is to show that a specific understudied type of finite-state technology can account for virtually all reduplicative processes. This type of transducer is known as a 2-way Finite-State Transducer or 2-way FST (Savitch

[1] Most of the exceptional languages are Indo-European, but some argue that these languages still use total reduplication (Ghomeshi et al. 2004; Stolz et al. 2011).

1982; Engelfriet and Hoogeboom 2001; Filiot and Reynier 2016). In theoretical computer science, 2-way FSTs are known to be able to model unbounded copying (Engelfriet and Hoogeboom 2001). To our knowledge, we are the first to apply 2-way FSTs to computational linguistics.[2]

The FSTs used in most of computational linguistics are more accurately called *1-way* FSTs. They can only read the input once in one direction. 2-way FSTs are more expressive because the read head can move back and forth on the input tape. On the other hand, the write head can only move forward on the output tape. For this reason, they are less expressive than Turing machines. It is this back-and-forth movement of the read head that allows 2-way FSTs to adequately model reduplication without the difficulties faced by 1-way FSTs. This article introduces deterministic 2-way finite-state transducers (2-way D-FSTs) in section §3, along with their formal definition (§3.1), illustrative examples of reduplication (§3.2, and their computational properties (§3.3).

The fact that the 2-way FSTs used in this article are deterministic is significant. It is well known that deterministic 1-way FSTs are less expressive than non-deterministic 1-way FSTs (Elgot and Mezei 1965; Schützenberger 1975; Choffrut 1977; Mohri 1997; Heinz and Lai 2013). Similarly, 2-way D-FSTs are less expressive than non-deterministic 2-way FSTs (Culik and Karhumäki 1986). Consequently, the empirical result that reduplication can be modeled with deterministic 2-way FSTs is in line with work which shows that various phonological and morphological processes can be described with deterministic finite-state technology (Chandlee et al. 2012; Gainor et al. 2012; Chandlee and Heinz 2012; Heinz and Lai 2013; Chandlee 2014; Luo 2017; Payne 2014, 2017).

Next this article provides a comprehensive cross-linguistic survey of reduplicative processes based on earlier typological studies (Moravcsik 1978; Rubino 2005; Inkelas and Downing 2015a), defined as an input-to-output function (McCarthy and Prince 1995). This survey is documented in a database we constructed, which we call The

[2] 2-way finite-state *acceptors* (2-way FSAs) have been used to model non-concatenative Semitic morphology (Narayanan and Hashem 1993) and parsing dependency grammars (Nelimarkka et al. 1984).

RedTyp database.[3] It contains entries for 138 reduplicative processes from 91 languages and a 2-way D-FST for each entry. Aspects of this survey are presented in ß3-4, and a detailed presentation is given in §6.

In §4, we compare 2-way D-FSTs to 1-way FSTs in terms of empirical coverage (§4.1), practical utility (§4.2), and linguistic motivation (§4.3). We argue that 2-way D-FSTs are linguistically motivated in that they capture the correspondence relations underlying the base and reduplicant in a linguistically natural way. Origin semantics (Bojańczyk 2014) acts as a diagnostic for the strong generative capacity of reduplicative functions. (We do not claim that origin semantics matches linguistic intuitions exactly in every case, but rather that it approximately does so in many instructive cases.)

The final contribution of this article is to classify reduplicative processes according to subclasses of 2-way D-FSTs in §6. The first result we already mentioned: the full typology of reduplicative processes can be modeled with *deterministic* 2-way D-FSTs. These subclasses are defined in terms of concatenations of subclasses of 1-way FSTs. Our next result is that approximately three-quarters of the typology is the concatenation of Output Strictly Local (OSL) functions (Chandlee et al. 2015). The remainder of the typology are the concatenation of sequential functions with some arguably requiring sweeping transducers or unrestricted 2-way D-FSTs. Section

We review these contributions and conclude in §7.

2 BACKGROUND ON COMPUTATION OF REDUPLICATION

Within computational linguistics, reduplication has been a challenging process to model (Culy 1985; Sproat 1992; Roark and Sproat 2007; Hulden 2009a; Chandlee 2014, 2017). Finite-state technology, as currently practiced, cannot adequately and elegantly describe many cases of productive reduplication, especially unbounded total reduplication. There are three kinds of issues: empirical coverage, practical utility, and in matching the intensional description of reduplication. We dis-

[3] An anonymous copy of RedTyp can be found online at our anonymous GitHub page `https://github.com/anonling/RedTyp`.

cuss these challenges in §2.1. In response to these problems, some have proposed finite-state approximations for reduplication (§2.2) or developing more expressive systems just for total reduplication (§2.3). The latter approach however implies that total reduplication is ontologically different than partial reduplication, and thus should be computed differently. In §2.4, we discuss this implication and show that the evidence for it is inconclusive. We summarize in §2.5.

2.1 *Why reduplication is challenging*

Reduplication is challenging because segmental copying entails multiple crossing long-distance dependencies between the two copies. When the number of copied segments (and thus the number of crossing long-distance dependencies) is bound to some maximum number n, the outcome is partial reduplication. When there is no bound, the outcome is total reduplication.

Partial reduplication can be modeled with 1-way FSTs (Roark and Sproat 2007; Chandlee and Heinz 2012). However, as we explain in more detail in section 4.3, these machines are understood as memorizing all finitely-many possible forms of the partial reduplicant. Consequently, the transducers suffer from an explosion of states and become unwieldy. For example, a partial reduplicative process in a language with a medium-sized phonemic inventory of 22 consonants and 5 vowels with and a CVC template would require at least $22 + 22 \times 5 + 22 \times 5 \times 22 = 2552$ states. 1-way FSTs likewise arguably do no match the intensional description of reduplication as a *copying* process because the FSTs simply memorize all possible reduplicants in the language (Roark and Sproat 2007, 54). This is discussed in detail in §4.3.

On the other hand, total reduplication cannot be modeled at all with 1-way FSTs (Culy 1985). This inability is due to the fact that the output language of total reduplication is not a regular language. Rather, the copying process of total reduplication can create output languages that are identical to the non-context-free $L_{ww} = \{ww | w \in \Sigma^*\}$ (Hopcroft and Ullman 1969). Thus the copying (sometimes called squaring) function $w \mapsto ww$ is beyond the expressivity of 1-way FSTs. In fact, virtually all attested morphological processes can be described with 1-way finite-state acceptors and transducers, *except* for total reduplication (Langendoen 1981; Gazdar and Pullum 1985; Roark and

Sproat 2007). In response to this problem, computational morphologists have often resorted to using either finite-state approximations (§2.2) or non-finite-state tools (§2.3).

2.2 *Finite-state approximations*

The literature on finite-state morphology contains many finite-state approximations to reduplication (Walther 2000; Beesley and Karttunen 2003; Cohen-Sygal and Wintner 2006; Hulden and Bischoff 2009). Roark and Sproat (2007, 57) and Cohen-Sygal and Wintner (2006, 52) provide reviews. In general, finite-state approximations are designed to lessen the burden for the developer in designing reduplication rules. They introduce new operations or tools over 1-way FSTs but they do *not* increase their expressivity. In other words, they are designed to improve on practical utility but they don't improve on empirical coverage or intensional description.

Here we briefly review the two main sets of approaches with details following. One set of approaches checks for identity between the two copies (Cohen-Sygal and Wintner 2006; Hulden and Bischoff 2009). Another set of approaches essentially 'postpone' reduplication to a run-time task (Walther 2000; Beesley and Karttunen 2000, 2003). Both try to reduce state complexity either by making a trade-off with time complexity, by implementing reduplication with a unique 1-way transducer for each morpheme in the finite lexicon, or both.

Cohen-Sygal and Wintner (2006) augment 1-way FSAs with finitely many registers (FSRA). These registers keep track of a *bounded* number of segments previously seen in the input. In order to model the total reduplication of a *given* word like *buku*→*buku*∼*buku* (1b), the FSRA has at least as many registers as segments in the base *buku*: four. The registers check that the string *buku*∼*buku* can be broken down into identical copies. Similarly, Hulden and Bischoff (2009) design the EQ function within the `foma` system (Hulden 2009b) which checks if some string is divided into two identical copies.

As for run-time procedures, these systems are designed on an input by input basis. Given some input word, they create a reduplication FST for it *on the fly*. Given an input *buku*, the compile-replace operation (Beesley and Karttunen 2000, 2003) creates an intermediate representation *buku2* via a 1-way FST. This intermediate representation is then interpreted as a regular expression in run-time, i.e. it is *compiled*. By

compiling this regular expression, the word *bukubuku* is outputted.

Within the framework of One-Level Phonology (Bird and Ellison 1994), Walther (2000) models reduplication by representing a potentially-reduplicated morpheme like *buku* as an FSA with augmentations on the types of transition arcs: *content*, *repeat*, and *skip* arcs. These transition arcs turn a linear string *buku* into a multi-linear structure where the reading head can 'move' around the string. This enriched representation is then intersected with a reduplication FSA that is designed to 'move' around this enriched representation and generate *buku*∼*buku*.[4] Ideally, these operations should be applied in run-time. Otherwise, if these operations are applied to the entire lexicon and stored as a single FST, then they suffer from the state explosion that they were designed to avoid.

As for total reduplication, all four of the above modifications are *approximations*. They are approximations because they impose various restrictions which contradict the linguistic generalization that total reduplication is independent of string length. Most notably, to model total reduplication, all four approximations permit only a closed finite set of input strings to undergo total reduplication. This restriction fundamentally alters total reduplication as a process which in principle applies to infinitely many words to one which applies to only finitely many. Such approximations thus fall short of capturing how total reduplication is used as a productive process in natural language.

As for partial reduplication, all of the above four approaches have the same expressivity and are able to capture the linguistic generalization that partial reduplication is independent of string length. However, although they are designed to avoid state explosion by one means or another, they still can be said to memorize the partial reduplicant as opposed to copying it (see section 4.3). In this way they do not intensionally capture the linguistic generalization of copying.

2.3 *Extending formal power*

Because of the difficulty in modeling reduplication with finite-state machinery, various augmentations and expansions of context-free

[4] Walther (2000) does not give a formal analysis. But, we think that these augmented transition arcs are similar to 2-way FSAs and are an independently developed implementation for Precedence-Based Phonology (Raimy 2000).

grammars have been proposed to handle L_{ww} and reduplication. An early augmentation is Reduplication Context-Free Grammars (Manaster-Ramer 1986; Savitch 1989) designed to handle context-free languages and reduplication by using queues instead of stacks. A more recent augmentation is Multiple-Context Free Grammars (MCFGs) which can model L_{ww} (Seki et al. 1991, 1993). MCFGs have been used to model reduplication (Albro 2000, 2005). As an extension of MCFGs, Parallel MCFGs have been used to model reduplication and syntactic copying (Kobele 2006; Clark and Yoshinaka 2012, 2014; Clark 2017).[5] Crysmann (2017) explores the use of HPSG to model total reduplication.

These technologies have had considerably less attention within mainstream computational morphology than finite-state approximations. One shortcoming of these approaches is that they model formal languages, not transformations. They accept well-formed reduplicated words *ww*, but they do not generate a reduplicated word *ww* given some input *w*. Thus, they do not model the squaring function $w \mapsto ww$.

2.4 *Computational distinctions between total and partial reduplication*

The previous sections showed that more expressive mechanisms are needed to model reduplication. Conceptually, the use of more powerful computational formalisms implies that reduplication is 'different' from the rest of morpho-phonology which can be modeled using 1-way FSTs (Roark and Sproat 2007, 60). This is especially the case for total reduplication which cannot be exactly modeled with 1-way FSTs, whereas partial reduplication can. This difference has caused debates over whether both types of reduplication *should* be computed with the same formalism or not. However, this debate is inconclusive.

On one hand, Chandlee (2017) suggests that the inadequacy of 1-away FSTs for total reduplication is evidence for total reduplication being ontologically different from partial reduplication. There is some empirical support for this argument. Prosodically, total reduplication resembles more 'syntactic' processes like compounding more often than partial reduplication (Downing 2006). The two copies in

[5] Kobele (2006) shows that syntactic copying can generate languages of the form a^{2^n}, i.e., exponential copying. This isn't attested in morphological copying. Note the string a^{2^n} is generated as the yield language of a tree transduction over a derivation tree.

total reduplication can be stressed separately or have separate tonal contours (Downing 2003).

On the other hand, partial and total reduplication are closely related processes. Typologically, if a language has partial reduplication, then it almost always has total reduplication too (Rubino 2013). Diachronically, both types of reduplication are related to each other, but not always (Hurch and Mattes 2009). And in linguistic theory, both are modeled with the same tools (Steriade 1988; Raimy 2000; Inkelas and Zoll 2005; McCarthy et al. 2012).

Psycholinguistic work would shed more light on the issue of total reduplication vis a vis partial reduplication. Sadly, there is little to no work on the psycholinguistic processing of reduplication. Existing work we are aware of focuses on partial reduplication, not total reduplication (Ohala et al. 1986; Waksler 1999).

Learnability is another factor which could tease these processes apart. It is an open question whether both partial and total reduplication can be learned in the same way with the same mechanism. In terms of stringsets or languages, reduplicated words *ww* can be learned with MCFGs (Clark and Yoshinaka 2012, 2014, 2016). There is a substantial body of work in cognitive science and connectionism on how to learn reduplicated words (Marcus et al. 1999; Berent et al. 2014, 2016, 2017; Andan et al. 2018; Alhama 2017; Alhama and Zuidema 2019). Here, the task is learning words which have have repeated substrings (ABB or ABA). This work thus focuses on learning partial reduplication, not total reduplication.

In contrast, there is little to no work on learning reduplication as a function ($w \rightarrow ww$), whether in machine learning or grammatical inference. To our knowledge, the only algorithm designed specifically for learning reduplication is Nevins (2004) in the principles-and-parameters tradition. There is some recent work on using neural networks to learn copying (Gu et al. 2016; Prickett et al. 2018; Wilson 2019). We speculate that one reason for the lack of learning results is due to the challenges outlined above for finding natural computational models for reduplication.

2.5 *Summary and consequences*

All in all, current finite-state treatments of reduplication have issues regarding their empirical coverage (total reduplication's productiv-

ity), practical utility (state space explosion), and intensional descriptions (copying vs. remembering). The present study uses a computational formalism which does not suffer from these three problems: two-way finite-state transducers (2-way FSTs).

3 TWO-WAY FINITE-STATE TRANSDUCERS: DEFINITION AND APPLICATION TO REDUPLICATION

1-way FSTs read the input *once* from left to right. Most applications use non-deterministic 1-way FSTs (Roark and Sproat 2007), though deterministic 1-way FSTs are largely sufficient (Chandlee 2017). 2-way FSTs can move back and forth on the input (Rabin and Scott 1959; Hopcroft and Ullman 1969). This ability makes them more expressive than 1-way FSTs (Savitch 1982; Engelfriet and Hoogeboom 2001).

It is useful to imagine a 2-way FST as a machine operating on an input tape and writing to an output tape. The symbols on the input tape are drawn from an alphabet Σ and the symbols written to the output tape are drawn from an alphabet Γ. For an input string $w = \sigma_1 \dots \sigma_n$, the initial configuration is that the FST is in some internal state q_0, its read head on σ_1, and its write head at the beginning of an empty output tape. After the FST reads the symbol under the read head, three things occur:

- The internal state of the FST may change.
- The FST writes some string, possibly empty, to the output tape.
- The read head moves in one of three ways: moves to the left (-1), moves to the right ($+1$), or stays (0).

This process repeats until the read head "falls off" one of the edges of the input tape. If for some input string w, the FST falls off the right edge of the input tape when the FST is in an accepting state after writing u on the output tape, we say the FST transduces, transforms, or maps, w to u. If for some input string w, the FST falls off the left edge, falls off the right edge while in a non-accepting state, or never falls off an edge, then the FST is undefined at w. The writing head cannot move back along the output tape. It can only advance as strings are written.

We formalize the definition and behavior of 2-way FSTs in §3.1. They are illustrated for reduplication in §3.2. We then describe their

generative capacity and computational complexity (§3.3).

3.1 *Preliminaries and formal definition*

Given a finite alphabet Σ, the set of all possible strings of finite-length built from Σ is Σ^*. The empty string is represented by λ. The length of a string w is $|w|$, so $|\lambda| = 0$. For the given strings w_1, w_2, their concatenation is $w_1 w_2$. Below is a formalization of deterministic 2-way FSTs based on Filiot and Reynier (2016) and Shallit (2008). We adopt the convention that inputs to a 2-way D-FST are flanked with the start ($\rtimes$) and end boundaries ($\ltimes$). This larger alphabet is denoted by $\Sigma_{\ltimes}$.

(2) **Definition:** A 2-way D-FST is a six-tuple $(Q, \Sigma_{\ltimes}, \Gamma, q_0, F, \delta)$ where:
- Q is a finite set of states,
- $\Sigma_{\ltimes} = \Sigma \cup \{\rtimes, \ltimes\}$ is the input alphabet,
- Γ is the output alphabet,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of final states,
- $\delta : Q \times \Sigma \to Q \times \Gamma^* \times D$ is the transition function where the direction $D = \{-1, 0, +1\}$.

A configuration of a 2-way D-FST T is an element of $\Sigma_{\ltimes}^* Q \Sigma_{\ltimes}^* \times \Gamma^*$. The meaning of the configuration (wqx, u) is that the input to T is wx and the machine is currently in state q with the read head on the first symbol of x (or has fallen off the right edge of the input tape if $x = \lambda$) and that u is currently written on the output tape.

If the current configuration is $(wqax, u)$ and $\delta(q, a) = (r, v, 0)$ then the next configuration is $(wrax, uv)$, in which case we write $(wqax, u) \to (wrax, uv)$. If the current configuration is $(wqax, u)$ and $\delta(q, a) = (r, v, +1)$ then the next configuration is $(warx, uv)$. In this case, we write $(wqax, u) \to (warx, uv)$. If the current configuration is $(waqx, u)$ and $\delta(q, a) = (r, v, -1)$ then the next configuration is $(wrax, uv)$. We write $(waqx, u) \to (wrax, uv)$. Observe that since δ is a function, there is at most one next configuration.

The transitive closure of $\to$ is denoted with $\to^+$. Thus, if $c \to^+ c'$ then there exists a finite sequence of configurations $c_1, c_2 \ldots c_n$ with $n > 1$ such that $c = c_1 \to c_2 \to \ldots \to c_n = c'$.

Next we define the function that a 2-way D-FST T computes. For each string $w \in \Sigma^*$, $f_T(w) = u \in \Gamma^*$ provided there exists $q_f \in F$ such

that $(q_0 \rtimes w \ltimes, \lambda) \rightarrow^+ (\rtimes w \ltimes q_f, u)$. If $F_T(w) = u$ then u is unique because the sequence of configurations sequence is determined deterministically.

There are configurations which halt the computation of a 2-way D-FST T on some input w, and then we say T is undefined on w. If the configuration is (qax, u) and $\delta(q, a) = (r, -1, v)$ then the derivation crashes and the transduction $f_T(ax)$ is undefined. Likewise, if the configuration is (wq, u) and $q \notin F$ then the transducer crashes and the transduction f_T is undefined on input w. Another way that f_T may be undefined for some input is if the input causes the transducer to go into an infinite loop.[6] This occurs for input $wx \in \Sigma^*_{\ltimes}$ whenever there exist $q \in Q$ and $u, v \in \Gamma^*$ such that $(q_0 wx, \lambda) \rightarrow^+ (wqx, u) \rightarrow^+ (wqx, uv)$.

3.2 *Illustration of two-way transducers for reduplication*

Having established what 2-way D-FSTs are, this section illustrates how they can be used to model reduplication. We provide two examples: total reduplication and partial initial-CVC reduplication. Both examples use deterministic 2-way FSTs.

Some useful terms are 'passes' and 'rewinds'. A pass (rewind) is when a 2-way D-FST moves left-to-right (right-to-left) from some position to another over the input.

Total reduplication is cross-linguistically the most common reduplicative process (Rubino 2005), and it is used in an estimated 85% of the world's languages (Rubino 2013). A canonical example is total reduplication in Indonesian which marks plurality (Cohn 1989).

(3)	a.	buku→ buku∼buku	'book'→'books'
	b.	wanita→ wanita∼wanita	'woman'→'women'
	c.	hak→ hak∼hak	'right'→'rights'
	d.	kəra→ kəra∼kəra	'donkey'→'donkeys'

Figure 1 shows a 2-way D-FST that captures this total reduplication process. The boundary symbol ∼ is a symbol in the output alphabet Γ, and is not necessary. We include it only for illustration. The 2-way D-FST in Figure 1 operates as follows:

[6] In practice, infinite loops are not a problem. It can be checked whether an input leads the 2-way D-FST into an infinite loop during run-time, in which case the computation can be halted.

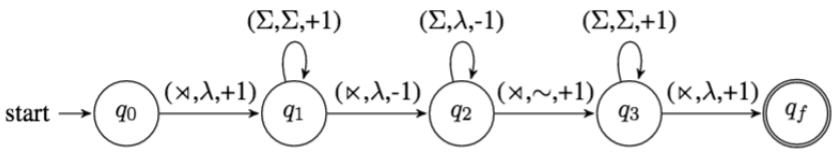


Figure 1: 2-way D-FST for total reduplication in Indonesian.

1. **First Pass**: It reads the input tape from left to right and outputs the first copy.
2. **Rewind**: When it reaches the end boundary ⋉, it 'rewinds' or goes back to the start of the input tape by moving left until the start boundary ⋊ is reached.
3. **Second Pass**: It reads the input tape once more from left to right and outputs the second copy.

The transition arcs are interpreted as follows. The symbol Σ is a variable representation of any alphabet symbol except for $\{\rtimes, \ltimes\}$. The arrow from q_1 to itself means this 2-way D-FST reads Σ, writes Σ, and advances the read head one step to the right on the input tape.

Table 1 shows an example derivation for *buku→buku∼buku* using the 2-way D-FST in Figure 1. The derivation shows the step-by-step configurations for the computation. The tuples in Table 1 consist of three parts. The first two represent the configuration and the third part shows the transition exercised to reach this configuration from the previous one. The underlined input symbol is what the FST will read next. In the first tuple, there is no transition used (N/A). Transitions in the other tuples are given in the form shown below.

$$\textit{input state} \xrightarrow[\text{direction}]{\text{input symbol:output string}} \textit{output state}$$

Partial reduplication processes are also very common. A common example is initial-CVC reduplication as in Agta (Moravcsik 1978, 311).

(4) a. takki→ tak∼takki 'leg'→'legs'
b. uffu→ uf∼uffu 'thigh'→'thighs'

The 2-way D-FST in Figure 2 expresses partial initial-CVC reduplication. An example derivation of takki→tak∼takki using our 2-way D-FST is provided in Table 2.

Outputting the first copy							
1.	($\boldsymbol{q}_0$⋊buku⋉,	λ,	N/A)	2.	(⋊$\boldsymbol{q}_1$buku⋉,	λ,	$q_0 \xrightarrow[+1]{⋊:λ} q_1$)
3.	(⋊b$\boldsymbol{q}_1$uku⋉,	b,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	4.	(⋊bu$\boldsymbol{q}_1$ku⋉,	bu,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)
5.	(⋊buk$\boldsymbol{q}_1$u⋉,	buk,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	6.	(⋊buku$\boldsymbol{q}_1$⋉,	buku,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)
Going back to the start of the tape							
7.	(⋊buk$\boldsymbol{q}_1$u⋉,	buku∼,	$q_1 \xrightarrow[-1]{⋉:∼} q_2$)	8.	(⋊bu$\boldsymbol{q}_2$ku⋉,	buku∼,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
9.	(⋊b$\boldsymbol{q}_2$uku⋉,	buku∼,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)	10.	(⋊$\boldsymbol{q}_2$buku⋉,	buku∼,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
11.	($\boldsymbol{q}_2$⋊buku⋉,	buku∼,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)				
Outputting the second copy							
12.	(⋊$\boldsymbol{q}_3$buku⋉,	buku∼,	$q_2 \xrightarrow[+1]{⋊:λ} q_3$)	15.	(⋊buk$\boldsymbol{q}_3$u⋉,	buku∼buk,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)
13.	(⋊b$\boldsymbol{q}_3$uku⋉,	buku∼b,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)	16.	(⋊buku$\boldsymbol{q}_3$⋉,	buku∼buku,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)
14.	(⋊bu$\boldsymbol{q}_3$ku⋉,	buku∼bu,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)	17.	(⋊buku⋉$\boldsymbol{q}_f$,	buku∼buku,	$q_3 \xrightarrow[+1]{⋉:⋉} q_f$)

Table 1: Derivation o /buku/ → [buku∼buku

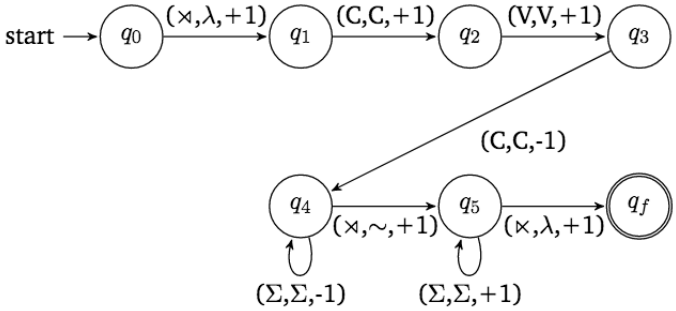


Figure 2: 2-way D-FST for initial-CVC reduplication in Agta.

Outputting reduplicant					Outputting the base			
1.	($\boldsymbol{q}_0$⋊takki⋉,	λ,	N/A)		8.	(⋊$\boldsymbol{q}_5$takki⋉,	tak∼,	$q_4 \xrightarrow[+1]{⋉:∼} q_5$)
2.	(⋊$\boldsymbol{q}_1$takki⋉,	λ ,	$q_0 \xrightarrow[+1]{⋊:λ} q_1$)		9.	(⋊t$\boldsymbol{q}_5$akki⋉,	tak∼t,	$q_5 \xrightarrow[+1]{Σ:Σ} q_5$)
3.	(⋊t$\boldsymbol{q}_2$akki⋉,	t,	$q_1 \xrightarrow[+1]{C:C} q_2$)		10.	(⋊ta$\boldsymbol{q}_5$kki⋉,	tak∼ta,	$q_5 \xrightarrow[+1]{Σ:Σ} q_5$)
4.	(⋊ta$\boldsymbol{q}_3$kki⋉,	ta,	$q_2 \xrightarrow[+1]{V:V} q_3$)		11.	(⋊tak$\boldsymbol{q}_5$ki⋉,	tak∼tak,	$q_5 \xrightarrow[+1]{Σ:Σ} q_5$)
5.	(⋊t$\boldsymbol{q}_4$akki⋉,	tak,	$q_3 \xrightarrow[-1]{C:C} q_4$)		12.	(⋊takk$\boldsymbol{q}_5$i⋉,	tak∼takk,	$q_5 \xrightarrow[+1]{Σ:Σ} q_5$)
Going back to the start of the tape					13.	(⋊takki$\boldsymbol{q}_5$⋉,	tak∼takki,	$q_5 \xrightarrow[+1]{Σ:Σ} q_5$)
6.	(⋊$\boldsymbol{q}_4$takki⋉,	tak,	$q_4 \xrightarrow[-1]{Σ:λ} q_4$)		14.	(⋊takki⋉$\boldsymbol{q}_f$,	tak∼takki,	$q_5 \xrightarrow[+1]{⋉:λ} q_f$)
7.	($\boldsymbol{q}_4$⋊takki⋉,	tak,	$q_4 \xrightarrow[-1]{Σ:λ} q_4$)					

Table 2: Derivation of /takki/→[tak∼takki].

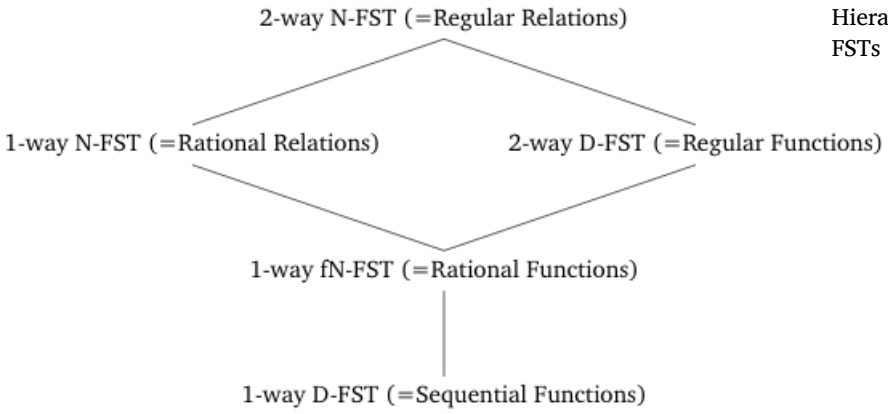


Figure 3: Hierarchy of FSTs

3.3 *Generative capacity and computational complexity*

With respect to acceptors, 1-way and 2-way finite-state acceptors are equivalent in expressive power. Both define the regular languages (Hopcroft and Ullman 1969; Shallit 2008). However, with respect to transducers, 1-way FSTs are strictly less expressive than 2-way D-FSTs (Savitch 1982; Aho et al. 1969; Filiot and Reynier 2016). For a 1-way FST, both the input language and the output language must be regular languages. A 1-way FST thus cannot have its output language be the non-regular copy language $L_{ww} = \{ww | w \in \Sigma^*\}$. In contrast, the output language of a 2-way D-FST can be a non-regular language such as L_{ww}.

Figure 3 shows the hierarchy of FSTs, adapted from Filiot and Reynier (2016, p. 8). Different FSTs have different generative capacity, based on whether the FST is deterministic (D-FST), non-deterministic (N-FST), 1-way, 2-way, and/or functional (f-FST).

2-way D-FSTs are equivalent in expressivity to MSO-definable string transductions (Engelfriet and Hoogeboom 2001) and to streaming string transducers (Alur 2010).[7] 2-way D-FSTs are less powerful than Turing machines because they cannot move back and forth on the output tape. They are closed under composition (Chytil and Jákl 1977). 2-way N-FSTs are invertible (Courcelle and Engelfriet 2012).

[7] A streaming-string transducer (SST) is a 1-way FST that uses finitely many registers of unbounded size. These registers allow it to keep track of previous information on the input tape, thus simulating 2-way D-FSTs.

Because of the difference in expressivity between 1-way and 2-way D-FSTs, it makes sense to give the classes of functions that they compute different names. We follow Filiot and Reynier (2016) who identify the class of functions describable with 1-way deterministic FSTs as 'sequential functions', with 1-way functional non-deterministic FSTs as 'rational functions', and with 2-way deterministic FSTs as 'regular functions'. The non-deterministic counterparts for 1-way and 2-way D-FSTs are respectively the 'rational relations' and 'regular relations'.

1-way D-FSTs run in time linear to the length of the input string. As for 2-way D-FSTs, one useful metric for measuring their complexity is in terms of the number of times the 2-way D-FST passes through the input (Baschenis et al. 2016). In the case of the reduplication examples in §3.2, the 2-way D-FSTs used only two passes through the input, one for each copy. Thus, the run time for those 2-way D-FSTs is at most $2n \cdot m$ where n is the number of passes and m is the length of the input. Since n here is fixed at 2, the run time is still linear in the size of the input string. To our knowledge existing applications of regular functions have been efficient (Alur and Černý 2011; Alur et al. 2014).

4 CONTRASTING 2-WAY D-FSTS WITH 1-WAY FSTS

Having illustrated how 2-way D-FSTs can model reduplication, here we contrast 2-way FSTs with 1-way FSTs on three criteria: empirical coverage, practical utility, and intensional description.

4.1 *Empirical coverage of the typology and productivity*

In terms of empirical coverage, 2-way D-FSTs can effectively model *virtually* the entire typology of reduplication as described by Moravcsik (1978), Hurch (2005), Inkelas and Zoll (2005), Rubino (2005), and Samuels (2010). We review part of this typology in §6. This stands in stark contrast to 1-way FSTs discussed in §2 We say *virtually* because there are two cases in the literature which require further discussion. These are discussed in §6.6.2.

4.2 *Practical utility and the RedTyp database*

To showcase the empirical coverage of 2-way D-FSTs and their practical utility, we have constructed the RedTyp database[8] which contains entries for 138 reduplicative processes from 91 languages. These were gleaned from various surveys (Rubino 2005; Inkelas and Downing 2015a). 50 of these processes were from Moravcsik (1978), an early survey which is representative of the cross-linguistically most common reduplicative patterns.

RedTyp contains 57 distinct 2-way D-FSTs that model the 138 processes. Each 2-way D-FST was designed manually, implemented in Python, and checked for correctness. On average, these 2-way D-FSTs had 8.8 states. This shows that 2-way D-FSTs are concise and convenient computational descriptions and models for reduplicative morphology. This is in contrast to 1-way FSTs which suffer from an explosion of states when modeling partial reduplication.[9]

To our knowledge, the only other database on reduplication is the Graz Database on Reduplication (Hurch 2005 ff.). However, RedTyp differs from the Graz Database because the latter does not include computational representations or implementations of its entries.

RedTyp offers a useful corpus of reduplicative patterns for research. For example as described in §6, we have used this database to identify subclasses of 2-way D-FSTs for classifying the typology of reduplication. This corpus could be used to test for other universal computational properties of reduplication. Since it contains 2-way D-FSTs, it can also be used to generate reduplicated forms. Such data sets could can be used to test morphological learning algorithms.

One shortcoming is that RedTyp under-represents cases of opacity in reduplication because our main source, Moravcsik (1978), did not list opaque cases. Only 5% of RedTyp displays opacity wherein phonological processes will exceptionally apply either across both or neither copies because of a drive to maintain identity between the two

[8] An anonymous copy of RedTyp can be found online at our anonymous GitHub page `https://github.com/anonling/RedTyp`.

[9] The largest 2-way D-FST in RedTyp is for verbal reduplication in Kinande (Downing 2000) with 29 states. This pattern depends on the size of the root and the number and type of suffixes and prefixes around it. In contrast, we estimate a deterministic 1-way D-FST would require over 1,000 states for this pattern of partial reduplication.

copies (McCarthy and Prince 1995). RedTyp focuses on morphological copying, not syntactic copying (cf. Kobele 2006).

4.3 *Linguistic motivation with origin semantics*

Importantly, using 2-way D-FSTs for reduplication is linguistically motivated and matches the intensional descriptions behind the linguistic generalizations on reduplication.

2-way D-FSTs do not approximate reduplication like 1-way FSTs do. 2-way D-FSTs do not copy by remembering strings of segments (see §2). Instead they *actively and literally copy*.

This contrast between copying and remembering can be formalized with the notion of the *origin semantics* of a transduction (Bojańczyk 2014).[10] Given a string-to-string function, the origin semantics of a function is the origin information of each symbol o_n in the output string. This is the position i_m of the read head on the input tape when the transducer had outputted o_n. To illustrate, consider a partial string-to-string function f_{ab} which maps *ab* to itself:

$$f(x) = \{(ab, ab)\}$$

This function can be modeled with at least two different 1-way FSTs as in the top row of Figure 4 which differ in when they output the output symbols *a*, *b*. In the bottom row of Figure 4, we visualize the origin information created by the two FSTs for the mapping (*ab*, *ab*) as graphs called *origin graphs* (Bojanczyk et al. 2017). The FSTs model the same function and are equivalent in their general semantics of what they output; however, they are not equivalent in their origin semantics because they use different origin information for their outputs.

This notion of origin semantics can be used to contrast how 1-way FSTs and 2-way FSTs model reduplication. Consider the toy example of initial-CV reduplication with a small alphabet $\Sigma = \{p,a,t\}$. This function can be modeled by either a 1-way or 2-way FST as in Figure 5. The two transducers in Figures 5 are equivalent in their general semantics because they can output the same string. For example, given the input *pat*, both FSTs will output *pa∼pat*. However, the two FSTs differ in their origin semantics for the mapping *pat→pa∼pat*. Setting

[10] For an application of origin semantics to MCFGs and potentially machine translation, see Nederhof and Vogler (2019).

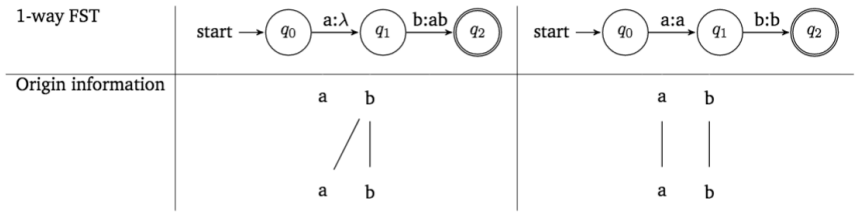


Figure 4: air of 1-way FSTs for the ction f_{ab} and the origin information ted by them he mapping $ab \rightarrow ab$.

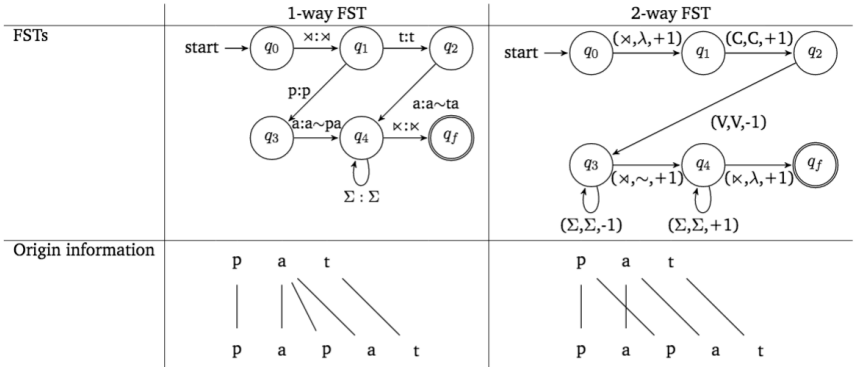


Figure 5: 1-way and 2-way FSTs for initial-CV reduplication and the origin information created by them for the mapping *pat→pa~pat*.

aside boundary symbols $\rtimes, \ltimes, \sim$, the 1-way FST associates the second *pa* string of the output with the vowel *a* of the input as in the bottom middle column of Figure 5. This is because the second *pa* was outputted when the 1-way FST was reading the *a* in the input. In contrast, the 2-way FST associates each segment in the output with an identical segment in the input as in the bottom right column of Figure 5.

The origin information created by the 2-way FST matches theoretical treatments of how the reduplicant's segments are individually associated with identical segments in the input (Marantz 1982; Inkelas and Zoll 2005).[11] In contrast, the origin information created by the 1-

[11] In Base-Reduplicant correspondence theory or BRCT (McCarthy and Prince 1995), what matters for reduplication is not the relationship or correspondence between the input and output segments, but between the two copies in the output. Origin semantics might be able to formalize the intuition behind BRCT with finite-state technology, e.g. output symbols with the same origin are in correspondence. The only computational implementation of BRCT to our knowledge (Albro 2000, 2005) uses MCFGs to do so. Note however that the empirical validity of BRCT is questionable (Inkelas and Zoll 2005; McCarthy et al. 2012).

way FST does not match linguistic intuitions of reduplication because non-identical segments are associated. This difference in the origin semantics of the 1-way FST and 2-way FST formalizes their behavior: the 1-way FST simply remembers what strings of segments to output twice (Roark and Sproat 2007, 54), while the 2-way FST actively copies.

5 LINGUISTIC MOTIVATIONS FOR SUBCLASSES OF TRANSDUCERS

Having shown the utility of 2-way D-FSTs for reduplication, the next two sections show that reduplication does not require the full power of 2-way D-FSTs but falls within certain subclasses. This means that reduplication has a demarcable generative capacity or complexity. In this section, we discuss the subclasses of 1-way FSTs that have been proposed to model segmental phonology (§5.1), specifically the Output-Strictly Local (OSL) functions and Sequential (Seq) functions. In §5.2, we discuss subclasses of 2-way FSTs and design new subclasses based on the concatenation of OSL and Seq functions. We explain the intuition behind using concatenation-based subclasses for reduplication (§5.3). The next section §6 goes over the typology of reduplication and show how it fits into these subclasses.

5.1 *Computational typology of phonology and 1-way transducers*

It is known that 1-way finite-state machines can model all attested phonological processes (Johnson 1972; Kaplan and Kay 1994; Mohri 1997). However, phonological processes do not require the full power of 1-way finite-state machines (Heinz 2007; Chandlee 2014). Subclass hierarchies have been discovered for 1-way FSAs (McNaughton and Papert 1971; Rogers and Pullum 2011; Heinz and Idsardi 2013) and 1-way FSTs (Garcia et al. 1990; Gainor et al. 2012; Heinz and Lai 2013; Chandlee et al. 2014, 2015). Some of these subclasses have been argued to characterize different types of phonological well-formedness conditions and transformations (Heinz 2018; Chandlee and Heinz 2018; Chandlee et al. 2018). We give a brief and informal overview.

A common intuition in linguistic theory is that phonological processes are local or subject to adjacency constraints (Odden 1994). For example, a common phonological process is post-nasal voicing (5a) whereby voiceless stops are voiced after nasals. This process is local

in the sense that the trigger to voicing (the nasal) is within a finite bound from the target of voicing (the stop). The symbols N, T, D represent nasals, voiceless stops, and voiced stops. Another local process is nasal spread whereby a vowel becomes nasalized after a nasal or nasalized vowel (5b). Nasal spread is iterative in that when a nasal can triggers nasalization on a subsequent vowel, the newly nasalized vowel can then nasalize its subsequent vowel (5b-iii). The symbols V,$\tilde{\text{V}}$ represent vowels and nasalized vowels.

(5) a. **Post-nasal voicing**
[+ stop, -voice] → [+ voice] / [+ nasal] _ *or* T → D / N_
i. /ata/ → [ata]
ii. /anta/ → [anda]

b. **Nasal spread**
[+ vowel] → [+ nasal] / [+ nasal] _ *or* V → $\tilde{\text{V}}$ / {N,$\tilde{\text{V}}$}_
i. /atapa/ → [atapa]
ii. /anapa/ → [anãpa]
iii. /anaapa/ → [anããpa]

Both processes are intuitively local. This intuition corresponds to Strict Locality in formal language theory (Rogers and Pullum 2011; Chandlee 2014). Formal definitions can be found in Chandlee et al. (2014, 2015). Informally, given an input string w, a function is Input-Strictly Local for a natural number k (k-ISL) if generating the output correspondent of some input symbol w_i relies on information about the current input symbol w_i and the $k-1$ most recently seen input symbols. Post-nasal voicing is a 2-ISL function and it is computed by the 1-way FST in Figure 6. The symbol ? marks any other segment which isn't in an existing transition arc (Beesley and Karttunen 2003). The state labels are interpreting as keeping track of the last seen input symbol. The state labeled as *N* is where the post-nasal consonant is generated as voiced. The state label's is interpreted as saying that a nasal was recently seen.

Output-Strictly Local functions for a number k (k-OSL) are analogously understood. A function is k-OSL if generating the output correspondent of w_i relies on information about the current input symbol w_i and the $k-1$ most recently seen output symbols. An OSL function is L-OSL (R-OSL) if the input is processed from the left (right). Nasal spread is a 2-L-OSL function and is computed by the 1-way FST in

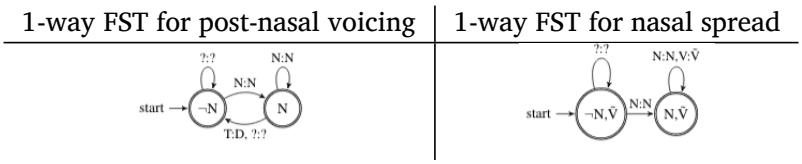


Figure 6: 1-way FSTs post-nasal voicing and nasal spread

Figure 7: Hierarchy of 1-way FSTs

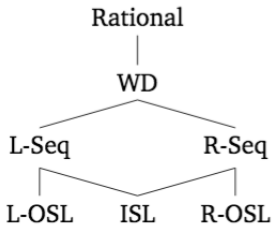


Figure 6. The state labels are interpreting as keeping track of the last recently generated output symbol. The state labeled as N, V is where a nasalized vowel is generated; the state label is interpreted as saying that most recent output symbol was a nasal or nasalized vowel.

ISL and OSL functions are part of a larger hierarchy of functions which are computed by 1-way FSTs. They are shown in Figure 7. The Sequential functions correspond to 1-way deterministic FSTs (1-way D-FST), while rational functions correspond to 1-way functional non-deterministic FSTs (1-way fN-FST) from Figure 3. WD stands for Weakly Deterministic functions which can compute some patterns in vowel harmony (Heinz and Lai 2013) and tone (Jardine 2016).

For modeling reduplication, a relevant process is truncation, such as nickname formation in English. This truncation process will output the first (C)VC of the input but delete everything after.

(6) **English truncation**

a.	dʒɛfri→dʒɛf	‘Jeffrey’→‘Jeff’
b.	deɪvɪd→deɪv	‘David’→‘Dave’
c.	ælən→æl	‘Alan’→‘Al’

English nickname formation is a 3-L-OSL function because it requires a window of size three in the output tape. The window keeps track of the last 2 symbols on the output tape and the current input

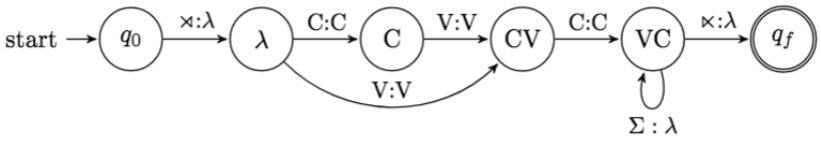


Figure 8: L 1-way FST for English nickname formation

symbol. The 3-OSL 1-way FST function in Figure 8 outputs up until the first VC of the input; it then stops outputting anything after that.[12]

A significant proportion of segmental phonology can be modeled with ISL and OSL functions (Chandlee 2014; Chandlee and Heinz 2018; Chandlee et al. 2018). Long-distance processes in phonology are however neither ISL or nor OSL. For example, Kikongo nasal harmony (7) requires the higher subclass of Sequential functions (Gainor et al. 2012). In Kikongo, alveolar stops like *d* or *l* surface as *n* if a nasal precedes them anywhere in the input. There can be any number of vowels and consonants intervening between the triggering nasal and the target alveolar (7c). This long-distance information means that the 1-way FST must keep track of whether a nasal consonant was seen *anywhere* in the input stem before it will output the alveolar.

(7) **Kikongo nasal harmony**

a.	/sakid-ila/→[sakid-ila]	'to congratulate for'
b.	/mant-ila/→[mant-ina]	'to climb for'
c.	/tunik-idi/→[tunik-ini]	'we ground'

The above pattern cannot be modeled by an ISL or OSL function but requires a Sequential 1-way FST as in Figure 9. A Seq 1-way FST is a deterministic 1-way FST that will read the input in only one direction (here left-to-right) and can use any information that it had found in the input string when processing the next input symbol.

To summarize, different types of phonological processes are computed by different subclasses of rational functions and with different subclasses of 1-way FSTs. The relatively low complexity of this subclasses has opened doors to understanding the cognitive limitations and learnability of phonological processes (Heinz 2018). In the next

[12] See Chandlee (2017) on why this function is necessarily OSL and not ISL. We have simplified the analysis by not considering cases of complex onsets in the input, e.g. stivn → stiv ('Steven'→'Steve').

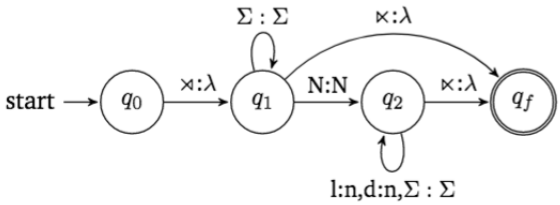
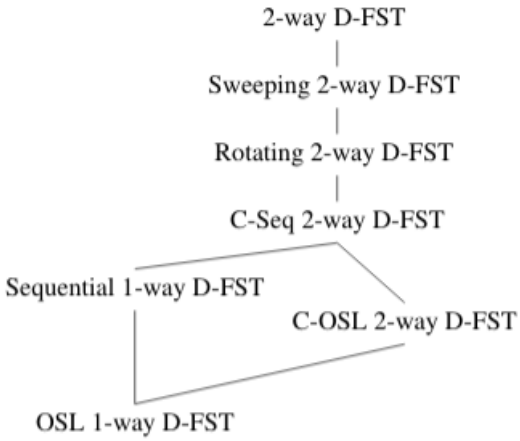


Figure 9: Sequential 1 FST for Kiko nasal harmo

Figure 10: Hierarchy of subclasses for 2-way FSTs



section, we show that extending OSL and Sequential functions into 2-way FSTs opens similar doors for the typology of reduplication.

5.2 *Subclasses of 2-way finite-state transducers*

Unlike for 1-way FSTs, there is much less work on subclasses for 2-way FSTs. Some intuitive subclasses have been proposed in the literature. The typology of reduplication inspired us to devise additional subclasses. All of these subclasses, shown in Figure 10, restrict:

1. where the 2-way FST can rewind in the input,
2. what it can output while it is rewinding, and
3. what information can be transferred across multiple passes, i.e., if a later pass depends on an earlier pass.

At the top of the hierarchy are 2-way D-FSTs which correspond to regular functions. In regards to the first restriction, a 2-way FST is a **sweeping** transducer if the read head can change direction only at

the ends of the input (Baschenis et al. 2015, 2016, 2017a). A sweeping transducer is a generalization of similarly defined sweeping automata (sweeping 2-way FSAs) (Sipser 1980). For example, the 2-way FST for total reduplication in Figure 1 is a sweeping transducer. The only time the FST moves right-to-left is going from the end boundary $\ltimes$ to the start boundary $\rtimes$. In contrast, the 2-way FST for initial-CVC partial reduplication in Figure 2 is not a sweeping transducer. It rewinds from the third input segment C to the beginning $\rtimes$. However, the partial reduplication function computed by this 2-way FST can be computed by a sweeping transducer, which we show in the next section.

2-way D-FSTs are more expressive than deterministic sweeping 2-way D-FSTs. Consider the function $u_1\#u_2\#\ldots\#u_n \mapsto u_n\ldots u_2u_1$ where the input is a sequence of strings u_i separated by the special symbol # (Baschenis et al. 2016). The output is formed by reversing these strings and deleting the #'s. This function can be computed by a deterministic 2-way D-FST but not by a sweeping transducer.

In regards to the second restriction, a sweeping transducer is a **rotating** transducer if it does not output anything while it's moving right-to-left (Baschenis et al. 2017b). The 2-way FST for total reduplication is a rotating transducer because it outputted nothing while moving right-to-left. Sweeping transducers are more expressive than rotating transducers. A sweeping transducer can compute the mirror function $w \rightarrow ww^r$, but a rotating transducer cannot.

As for the third restriction, we develop a set of concatenated-defined subclasses.

(8) *Subclasses of Regular Functions*

a. **C-Seq function**: A Concatenated-Sequential function f is the concatenation of n Sequential functions s_i, e.g. $f(x) = s_1(x) \cdot s_2(x) \cdot \ldots \cdot s_n(x)$. f is C-L-Seq (C-R-Seq) if the component Seq functions read the input left-to-right (right-to-left).[13]

[13] In terms of function combinatorics for regular string transformations (Alur et al. 2014; Dave et al. 2018), the class of C-Seq functions involves the use of a 'sum combinator' $\bigotimes$ that concatenates the output of two or more Seq functions: $f(x) = s_1(x) \bigotimes s_2(x) \bigotimes \ldots s_n(x)$ where s_i is a Seq function. This is similar to the use of product automata. See Alur et al. (2014) for details.

b. **C-OSL function**: A Concatenated-OSL function f is the concatenation of n Output-Strictly Local functions o_i, e.g. $f(x) = o_1(x) \cdot o_2(x) \cdot \ldots \cdot o_n(x)$. f is C-L-OSL (C-R-OSL) if the component OSL functions read the input left-to-right (right-to-left).

Rotating transducers are more expressive than C-Seq transducers which are more expressive than C-OSL transducers. Examples witnessing these separations are drawn from the typology of reduplication in §6. C-Seq functions are more expressive than sequential functions (= 1-way D-FSTs) which are more expressive than OSL functions. A set of definitions is provided below for easier reference.

(9) *Subclasses of 2-way D-FSTs*

a. **Sweeping 2-way FST**: A 2-way FST which can change direction only at the ends of the input

b. **Rotating 2-way FST**: A sweeping 2-way FST which outputs nothing while moving right-to-left

c. **C-Seq 2-way FST**: A rotating 2-way FST that computes a Concatenated Sequential function

d. **C-OSL 2-way FST**: A rotating 2-way FST that computes a Concatenated Output-Strictly Local function

We have found that virtually the entire typology of reduplication can be modeled with deterministic rotating 2-way FSTs. Further, the bulk of the typology can be modeled with C-OSL functions. A few minor cases require C-Seq functions; these mostly involve infixal or internal reduplication. A smaller set of cases require the full power of rotating transducers; though these cases are not clear-cut. Before going through the typology in detail in §6, we illustrate the insight behind C-OSL functions and how they compute reduplicative processes.

5.3 *Illustrating C-OSL*

Intuitively, a C-OSL function is a function that takes as input the string x, gives x to n many separate 1-way FSTs which are OSL, and concatenates their output. To illustrate the insight behind C-OSL functions, consider initial-CVC partial reduplication from Agta again (Moravcsik 1978, 311) from (1a) repeated in (10a).

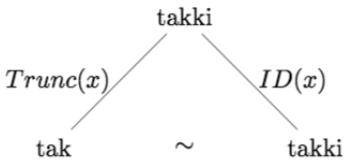


Figure 11: Initial-CVC plication as ncatenation f functions.

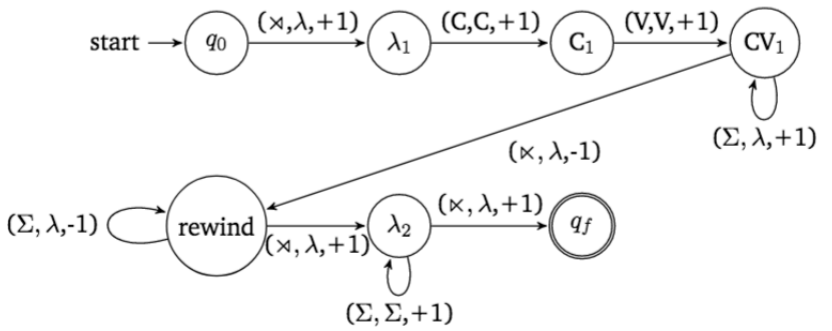


Figure 12: C-OSL 2-way D-FST for initial-CVC partial reduplication

(10) a. takki→tak∼takki ‘leg’→‘legs’
b. takki→tak~~ki~~∼takki

As an input-to-output function, reduplication may be viewed as submitting the same input to two separate functions in parallel and concatenating their output as in Figure 11. The first function, here labeled $Tr(x)$, truncates the input to the first CVC while the second function, $ID(x)$, is the identity function. The outputs of these two functions, *tak* and *takki*, are concatenated to form the reduplicated output: *tak∼takki*. In (10b), I explicitly show how initial-CVC reduplication can be seen as truncating the first copy: takki→tak~~ki~~∼takki where truncated material is shown in strike-through.

The truncation function $Tr(x)$ is a 3-L-OSL function because it outputs a truncation of the input to just the first CVC. This is similar to English nickname formation from §5.1. The identity function $ID(x)$ is both 1-L-OSL and 1-R-OSL. Thus both $Tr(x)$ and $ID(x)$ are L-OSL and hence their concatenation is C-OSL. Figure 12 illustrates a 2-way D-FST for initial-CVC reduplication which is formulated as a concatenation of these two OSL functions. Contrast this model of initial-CVC reduplication (shown in Figure 12) with the non-rotating 2-way D-FST shown in Figure 2. (It is the the additional state CV_1 and its transition arcs in Figure 12 which make this D-FST rotating.)

To summarize this section, understanding reduplicative processes as C-OSL and C-Seq functions is intuitive. This analysis echoes Steriade (1988)'s treatment of partial reduplication as total reduplication followed by truncation and Inkelas and Zoll (2005)'s treatment of total reduplcation as morphological doubling.

6 COMPUTATIONAL TYPOLOGY OF REDUPLICATION

This section provides a detailed, comprehensive review of the RedTyp typology and classifies RedTyp's reduplicative processes according to the computational classification introduced in the last section. The main finding is that most processes are C-OSL (§6.2) and most of the ones that are not are C-Seq (§6.3). There are few (and questionable) cases where reduplication needs the full power of 2-way D-FSTs (§6.4). We give an overview in §6.5.

Note that all partial reduplicative processes can be computed by 1-way FSTs. However, in order to get the linguistically-motivated origin semantics right (§4.3), we need the additional power of 2-way FSTs. Thus in this section, when we discuss how partial reduplicative cases fit into the subclasses of 2-way FSTs in this section, we mean in terms of them generating the right origin semantics.

6.1 *Preliminaries to the typology*

Although reduplication is cross-linguistically ubiquitous, there is a wide cross-linguistic variation in a) what substring or subsequence gets repeated, b) where the copied substring or subsequence is placed in the output, and c) whether and how phonological processes interact with copying. This section provides a brief but representative typology of reduplication compiled from various surveys (Moravcsik 1978; Rubino 2005; Inkelas and Downing 2015a).

We emphasize that our reported typology is descriptive and not theoretical. Various theoretical frameworks have been developed to account for the range of variation on reduplication (Marantz 1982; McCarthy and Prince 1995; Spaelti 1997; Raimy 2000; Inkelas and Zoll 2005; Frampton 2009; Samuels 2010; McCarthy et al. 2012; Kirchner 2010, 2013). The reader is referred to elsewhere for theoretical overviews (Raimy 2011; Urbanczyk 2007, 2011; Inkelas and Downing 2015a,b).

We define the following *descriptive* terms which will be useful in categorizing different reduplicative processes:

(11) *Terminology for categorizing the typology:*

- **reduplicant**: The substring in the output which was created via copying
- **base**: The substring in the output which was not created via copying.
- **target**: the substring in the input which will be copied or duplicated
- **anchor point**: the position in the input where the target starts or ends
- **source**: the morphological or phonological constituent in the input which contains the target.

The output-based terms *base* and *reduplicant* are common in the literature on reduplication (McCarthy and Prince 1995) though their definition is problematic (Shaw 2005; Haugen 2009). Anchor points have been proposed for reduplication (Fitzpatrick 2006; Raimy 2009) and other non-concatenative processes (Yu 2007; Samuels 2010). We introduce the input-based terms *source* and *target* in order to better fully describe reduplication as an input-to-output function. This section goes through the typology of reduplication, organized in terms of how they vary in the source, target, and/or reduplicant. These variations align with what type of 2-way FST is needed to compute them. Example FSTs are given in the appendix.

6.2 *Most reduplication is C-OSL*

Most reduplicative processes are C-OSL. We go through common and some uncommon reduplicative processes and show they are C-OSL. Informally, some criteria for a C-OSL function is that each of the component functions

1. reference only a finite and bounded number of the most recently generated output symbols, meaning that each of the functions
2. do not depend on any long-distance information in the input,
3. do not use any finite lookahead or finite lookback on the input
4. do not rely on deleted material, and

5. do not rely on any information from the other function

The two copies are thus independent of each other. The two passes over the input are likewise independent of each other.

6.2.1 Total and word-initial partial reduplication

Total reduplication and word-initial partial reduplication are C-L-OSL, which means they are the concatenation of two L-OSL functions.

Consider total reduplication first.

(12) a. **Total reduplication**
Indonesian (Cohn 1989)
buku→ buku∼buku ‘book’→‘books’

b. **Initial-CVC reduplication**
Agta (Moravcsik 1978, 311)
takki→ tak∼takki ‘leg’→‘legs’

For total reduplication, the two OSL functions are identity $ID(x)$. (Total reduplication is also C-R-OSL.)

For partial reduplication, there is limited variety in the shape of the copied material, the *reduplicant*. Some languages have a partial reduplicative process that copies the first *C* or consonant of the word (13a), first CV or consonant-vowel sequence (CV) of the word (13b), first CVC sequence (13c), or first CV(C)CV sequence (13d). In general, the copied material has to fit into some template of a particular size.

(13) *Common prefixal partial reduplicative patterns*

a. **Initial-C reduplication**
Shilh (Moravcsik 1978, 308)
gen→g∼gen ‘to sleep’→‘to be sleeping’

b. **Initial-CV reduplication**
Sundanese (Moravcsik 1978, 319)
guyon→gu∼guyon ‘to jest’→‘to jest repeatedly’

c. **Initial-CVC reduplication**
Panganisan (Rubino 2005, 11)
baley→bal∼baley ‘town’→‘towns’

d. **Initial-CV(C)CV reduplication**
Dyirbal (McCarthy et al. 2012, 187)

Table 3: OSL treatment for total and initial partial reduplication

	Total (12a)	Initial-C (13a)	Initial-CV (13b)	Initial-CVC (13c)	Initial-CV(C)CV (13d)
Input x	buku	gen	guyon	baley	balgan
Components	$ID(x) \cdot ID(x)$	$Tr(x) \cdot ID(x)$	$Tr(x) \cdot ID(x)$	$Tr(x) \cdot ID(x)$	$Tr(x) \cdot ID(x)$
Output	buku∼buku	~~gen~~∼gen	gu~~yon~~∼guyon	bal~~ey~~∼baley	balga~~n~~∼balgan
Subclass	C-L-OSL C-R-OSL	C-L-OSL	C-L-OSL	C-L-OSL	C-L-OSL

a. baniŋu→bani∼baniŋu ‘come’
b. balgan→balga∼balgan ‘laugh’

As for the partial reduplication functions in (13), they are all C-L-OSL just like initial-CVC reduplication from section §5.3. They involve the concatenation of a truncation $Tr(x)$ and identity function $ID(x)$. The truncation function varies in terms of how much word-initial material is faithfully outputted.

Table 3 illustrates these examples where the truncated material is shown in strike-through. The output of the two component OSL function are separated by ∼ for illustration.

6.2.2 Variation in the number and placement of copies

The typology is larger than the above examples. Some languages create three copies of the input (triplication) instead of just two (14a). Some reduplicative processes are suffixal; they specify that the location of the target be a word-final substring (14b) instead of word-initial substring (13d). Some reduplicative process are wrong-sided by making the target and the reduplicant not adjacent in the output, i.e., copying the final CVC and placing it at the beginning of the output (14c vs. 13c). There are likewise cases where both the base and the reduplicant are shortened or truncated in the output, e.g., truncating both copies to CV (14d).

(14) *Variation in number and reduplicant placement*

a. **Total triplication**
Mokilese (Moravcsik 1978, 301)
roar→ roar∼roar∼roar ‘give a shudder’ →‘continue to shudder’

b. **Final-CVCV reduplication**
Siriono (Moravcsik 1978, 308)
erasi→erasi∼rasi ‘he is sick’→‘he continues being sick’

Table 4: C-OSL treatment for less common reduplication patterns

	Triplication (14a)	Final-CVCV (14b)	Wrong-sided (14c)	Abbreviated (14d)
Input x	roar	erasi	qanga	toni
Components	$ID(x) \cdot ID(x) \cdot ID(x)$	$ID(x) \cdot Tr(x)$	$ID(x) \cdot Tr(x)$	$Tr(x) \cdot Tr(x)$
Output	roar∼roar∼roar	erasi∼erasi	qanga∼qan~~ga~~	to~~ni~~∼to~~ni~~
Subclass	C-L-OSL C-R-OSL	C-R-OSL	C-L-OSL	C-L-OSL

c. **Initial-CVC reduplication and opposite-edge or wrong-sided placement**
Koryat (Riggle 2004, 3)
qanga→qanga∼qan ‘fire’→‘fire (ABS)’

d. **Abbreviated reduplication (*Kager-Hamilton Problem*)**
Guarijio Caballero (2006)[14]
toni→to∼to ‘to boil’→‘to start boiling’
muhiba→mu∼mu ‘to throw’→‘to start throwing’

All these processes are still C-OSL, however. They only differ in the number and order of concatenated functions, the direction in which the input is read, and whether all or none of the functions are identity. Their computation is visualized in Table 4. Triplication is C-L-OSL and C-R-OSL; it involves concatenating three identity functions. Suffixal reduplication like final-CVCV reduplication is C-R-OSL because it is the concatenation of an identity function and an R-OSL truncation function. The truncation function reads the input right-to-left and deletes everything to the left of the final CVCV. Wrong-sided initial-CVC reduplication is C-L-OSL. It differs from initial-CVC reduplication by ordering identity before truncation. Abbreviated reduplication is C-L-OSL. Unlike initial-CV copying, it is composed of two L-OSL truncation functions instead of just one.

6.2.3 Copying a morphological subconstituent

In the above examples, the *source* was the entire input. But unlike concatenative morphology, reduplication is often sensitive to word-internal morphological constituents, contra Bracket Erasure (Kiparsky 1982). In these cases, the reduplicant usually has semantic scope over

[14] Such reduplication is often argued to be unattested and is called the *Kager-Hamilton Problem* (Idsardi and Raimy 2008). See Caballero (2006) for discussion on what prosodic and morphological factors condition this rare type of reduplication.

the entire input. For example, some languages have reduplication target a morphological subconstituent within the input as the source, such as a root/stem (15a,15b) or affix (15c), and whether for total reduplication (15a,15c) or partial reduplication (15b). The source and reduplicant are usually adjacent; though there are some cases where the two copies are non-adjacent in the output, e.g., Madurese copies the root-final CVC and places it at the beginning of the output (15d).

(15) *Copying from a morphological subconstituent*

a. **Total reduplication of the stem**
KiHehe (Aronoff 1988, 8)
ku-haata→ku-haata∼haata
'to ferment'→'to start fermenting'

b. **Initial CV reduplication of the stem**
Bikol (Mattes 2007, 84)
na-murak→na-mu∼murak
'to flower'→'decorating with flowers'

c. **Total reduplication of an affix (prefix)**
Hungarian (Inkelas and Downing 2015a, 505)

a. el-megy→el∼el-megy	'he goes there'→'he occasionally goes there'
b. bele-nez→bele∼bele-nez	'he looks into it'→'he occasionally looks into it'

d. **Root-final CVC reduplication and word-initial placement**
Madurese (Brown 2017, 964)
pa-jalan-an→lan∼pa-jalan-an 'pedestrian'→'pedestrians'

More cases of reduplication targeting a morphological subconstituent are well-attested (Shaw 2005; Inkelas and Zoll 2005; Haugen 2009; Hyman 2009; Inkelas 2014). The above cases are C-OSL if the relevant morphological boundaries are present in the input. Their computation is visualized in Table 5. Each process uses two functions $L(x), R(x)$ which generate the two copies, reference the morphological boundaries, and they crucially output these boundaries.

For total copying in KiHehe and Hungarian, the function is C-L-OSL and C-R-OSL. For total stem copying in KiHehe, the first function $L(x)$ outputs everything up until the root right-boundary '$\}_r$'. The sec-

Table 5: C-OSL treatment for copying morphological subconstituents

	KiHehe	Bikol	Hungarian	Madurese
	(15a)	(15b)	(15c)	(15d)
Input x	$\mathrm{ku}\{_r\mathrm{haata}\}_r$	$\mathrm{na}\{_r\mathrm{murak}\}_r$	${}_p\{\mathrm{el}\}_p\mathrm{megy}$	$\mathrm{pa}\{_r\mathrm{jalan}\}_r\mathrm{an}$
Components	$L(x)\cdot R(x)$	$L(x)\cdot R(x)$	$L(x)\cdot R(x)$	$L(x)\cdot R(x)$
Output	$\mathrm{ku}\{_r\mathrm{haata}\}_r \sim$ ~~ku~~ $\{_r\mathrm{haata}\}_r$	$\mathrm{na}\{_r\mathrm{mu}$~~rak~~$\}_r \sim$ ~~na~~ $\{_r\mathrm{murak}\}_r$	$\{_p\mathrm{el}\}_p$~~megy~~ $\sim$ $\{_p\mathrm{el}\}_p\mathrm{megy}$	~~pa~~$\{_r$~~ja~~$\mathrm{lan}\}_r$ ~~an~~ $\sim$ $\mathrm{pa}\{_r\mathrm{jalan}\}_r\mathrm{an}$
Subclass	C-L-OSL C-R-OSL	C-L-OSL	C-L-OSL C-R-OSL	C-R-OSL

ond function $R(x)$ outputs nothing until it sees the root left-boundary '$\{_r$'; it outputs this and everything after it. Both functions are both L-OSL and R-OSL, thus KiHehe is C-L-OSL and C-R-OSL. Prefix copying in Hungarian is similarly defined but for the prefix boundaries '$\{_p$' and '$\}_p$'. Partial stem copying in Bikol is only C-L-OSL. The function $L(x)$ outputs everything up until it outputs the string '$\{_rCV$'; it deletes everything after that. The function $R(x)$ outputs nothing up until it sees the root left-boundary '$\{_r$'; it outputs this and everything after it. Non-local copying in Madurese is C-R-OSL. The function $L(x)$ reads the input right-to-left; it outputs nothing until it sees the root right-boundary '$\}_r$'; it outputs this and the first CVC that it sees. After that, it deletes everything. The function $R(x)$ is the identity function.

Even though some have argued against the use of morpheme boundaries in morpho-phonological representations (Anderson 1992; Stump 2001), morphological boundaries must be part of the input for a finite-state systems like ours (e.g. Karttunen 1983; Koskenniemi 1984; Roark and Sproat 2007).[15] Consider partial stem copying in Bikol: $\mathrm{na}\{_r\mathrm{murak}\}_r \rightarrow \mathrm{na}\{_r\mathrm{mu}\{_r\mathrm{murak}\}_r$. Without the root boundaries, we could not distinguish the prefixed input $\mathrm{na}\{_r\mathrm{murak}\}_r$ from a hypothetical mono-morphemic input $\mathrm{namurak}$. Without some way to encode the relevant morphological constituents in the input, we simply cannot define this reduplication function with any type of 1-way or 2-way FSTs. The use of such boundaries in finite-state morphology is standard practice.

[15] It should be noted though that HPSG-based approaches to computational morphology (Bonami and Crysmann 2013; Bonami and Crysmann; Crysmann and Bonami 2016) do not need morpheme boundaries as symbols in their alphabet. One reason is because they can essentially directly capture hidden morphological structure or constituency.

6.2.4 Copying a prosodic subconstituent

Besides morphological subconstituents, the source can be also be metrical or prosodic subconstituent such as the stressed syllable (16a), the first syllable (16b), or the first foot (16c). The source can also be a morphophonological constituent, e.g. a prosodic stem (16d). In Chumash, the prosodic stem (underlined) consists of all the segments in the morphological stem alongside any prefixal consonants that are syllabified with the morphological stem.

(16) *Copying from a metrical or prosodic subconstituent*

a. **CV-reduplication of the stressed syllable**
Chamorro (Inkelas and Downing 2015a, 507)
hu.gán.do→hu.gá∼gan.do 'play'→'playing'

b. **Total reduplication of the initial syllable**
Hiaki (Haugen 2009)
vu.sa→vu∼vu.sa 'awaken'
vam.se→vam∼vam.se 'hurry'

c. **Total reduplication of the initial foot**
Yidiny (Marantz 1982, 453)
(gindal)ba→gindal∼gindalba 'lizard sp.'→'lizards'

d. **Initial-CVC reduplication of the prosodic stem**
Chumash (Downing 1998, 101)
s+tʃeq→s-tʃeq∼tʃeq 'it is very torn'
s+ikuk→s+ik∼s-ikuk 'he is chopping'

If the relevant prosodic boundaries are in the input, the computation is C-OSL. The computation proceeds the same as for copying a morphological constituent. Table 6 shows this with syllable boundaries $(_s)_s$, foot boundaries $(_f)_f$, stressed syllable boundaries $(_S)_S$, and prosodic stem boundaries $(_{PS})_{PS}$.[16] Given an unsyllabified input, these prosodic boundaries can be generated via a 1-way FST (Hulden 2006; Yu 2017) which can be ISL because it uses finite lookahead on the input (Strother-Garcia 2018, 2019, see also).

[16] The second function $R(x)$ in stressed syllable CV-copying must change stressed *á* to unstressed *a*. Stressed syllable is also C-R-OSL if the first function $L(x)$ is R-OSL and outputs the right-boundary $)_S$. Generating the prosodic stem in Chumash requires reference to morphological boundaries too.

	Chamorro (16a)	Hiaki (16b)		Yidiny (16c)	Chumash (16d)
Input x	hugándo	vusa	vamse	gindalba	s+ikuk
Syllabify x	$\text{hu}(_S\text{gán})_S\text{do}$	$(_s\text{vu})_s(_s\text{sa})_s$	$(_s\text{vam})_s(_s\text{se})_s$	$(_f\text{gindal})_f\text{ba}$	$(_{PS}\text{s+ikuk})_{PS}$
Components	$L(x)\cdot R(x)$	$L(x)\cdot R(x)$	$L(x)\cdot R(x)$	$L(x)\cdot R(x)$	$L(x)\cdot R(x)$
Output	$\text{hu}(_S\text{gá}$ ~~$)_S$ndo~~ $\sim$ ~~$\text{hu}(_S$~~gan$)_S$do	$(_s\text{vu})_s$ ~~$(_s$sa$)_s$~~ $\sim$ $(_s\text{vu})_s(_s\text{sa})_s$	$(_s\text{vam})_s$ ~~$(_s$se$)_s$~~ $\sim$ $(_s\text{vam})_s(_s\text{se})_s$	$(_f\text{gindal})_f$ ~~ba~~ $\sim$ $(_f\text{gindal})_f\text{ba}$	$(_{PS}\text{sik}$ ~~uk~~$)_{PS}$ $\sim$ $(_{PS}\text{sikuk})_P\text{S}$
Subclass	C-L-OSL C-R-OSL	C-L-OSL	C-L-OSL	C-L-OSL	C-L-OSL

Table 6: C-OSL treatm for copying prosodic subconstitue

However, if the input to reduplication lacks boundaries, then reduplication is C-Seq because we need finite lookahead to know if some consonant is part of the relevant prosodic constituent. Consider initial syllable copying in Hiaki vusa→vu∼vusa. In the first function $L(x)$, the consonant *s* is not generated because it is part of the next syllable. We know because it precedes a vowel. In contrast for vamse→vam∼vamse, the consonant *m* is copied because it precedes consonant. The use of such information from finite lookahead on the input cannot be computed by an OSL function.

This section presented cases in RedTyp which are C-OSL. They comprise the bulk of reduplicative typology. Of the 138 reduplicative processes in RedTyp (§4.2), 121 (87%) were C-OSL.

6.3 *Some reduplication is C-Seq*

This section goes through some types of reduplication which are not C-OSL but are instead C-Seq. Informally, a reduplicative function is C-Seq if its component functions do not rely on any information from the other function or its output. A component function can use finite lookback, finite lookback, or even long-distance information in the input.

6.3.1 Internal reduplication and gray areas between C-OSL or C-Seq

One problematic area for C-OSL are internal reduplication functions which seem infixal (Broselow and McCarthy 1983; Gafos 1998; Spaelti 1997). Some of these are C-OSL, some are not. These functions are C-OSL if the truncation functions can uniquely determine what segments to delete based only on what it has outputted.

In the previous sections, the reduplication's target can be thought of as a contiguous substring that is determined by scanning either the left or right edge of the source. In those cases, the target was at the edge of the source and the reduplicant was placed at the left or right edge of the base or entire output. However, there are cases of *internal*

Case	Mangarayi (17a)	Chamorro (17b)	Quileute (17c)
Input x	gabuji	nalaŋ	$t^{ʕ}$iko
Output	gababuji	nalalaŋ	$t^{ʕ}$itko
Infixal treatment	gab⌣ab⌢uji	nala⌣la⌢ŋ	$t^{ʕ}$i⌣t⌢ko
C-OSL treatment	gab~~uji~~⇝~~g~~abuji	nala~~ŋ~~⇝~~na~~laŋ	$t^{ʕ}$i~~ko~~⇝t~~i~~ko
Subclass	C-L-OSL	C-R-OSL	C-L-OSL

Table 7: Infixal vs. C-OSL treatment of internal reduplication

or *infixal* reduplication where the target is a substring *inside* the source that's not strictly adjacent to the source's edges (17a,17b) and where the reduplicant is placed inside the base in the output (17c). In (17c), the word-initial C is copied and placed after the first vowel.

(17) *Internal reduplication cases which are arguably not C-OSL*

a. **Leftmost-VCC* reduplication**
Mangarayi (Raimy 2000, 135)
gabuji→g-ab⇝abuji 'old person'→'old persons'

b. **Rightmost-CV reduplication**
Chamorro (Inkelas and Zoll 2005, 107)
nalaŋ→nala⇝laŋ 'hungry'→'very hungry'

c. **Initial-C reduplication and internal placement**
Quileute (Broselow and McCarthy 1983, 44)
$t^{ʕ}$iko→$t^{ʕ}$i⇝tko 'he failed sp.'→'he failed (freq.)'

Table 7 visualizes these processes. A traditional analysis is that the reduplicant is infixal (Broselow and McCarthy 1983) where $<>$ marks infixation, e.g., Mangarayi gabuji→gab$<$ab$>$buji. However, their treatment as C-OSL is somewhat counter-intuitive because a C-OSL 2-way function models these processes as concatenating two truncation functions: gab~~uji~~⇝~~g~~abuji. The first function $L(x)$ outputs the first C*VC* substring and deletes everything after that. The second function $R(x)$ deletes all word-initial strings of consonants C*; once it sees a vowel V, it outputs it and everything after it

6.3.2 Internal or non-contiguous reduplication which is C-Seq

A main reason why Mangarayi and the other functions in Table 7 are C-OSL is because the target and deleted materials do not have the same shape. Knowing what to delete or generate doesn't need any finite lookahead or lookback over the input, just over the output. However, other cases of internal and non-contiguous reduplication do require

such finite lookback/lookahead over the input. This makes them C-Seq. In (18a), the penultimate syllable is reduplicated. In (18b), the word-initial CV is copied and placed before the final C. In most of these cases, the target is a contiguous substring in the input. In some cases, the target is not contiguous (18c). In (18c), the input's first CV and final C are copied and placed together at the beginning of the output.

(18) *Internal reduplication which are C-Seq*

a. **Penultimate syllable reduplication**
Samoan (Moravcsik 1978, 301,310)

a.lo.fa→a.lo∼.lo.fa	'he loves'→'they love'
ta.o.to→to.o∼o.to	'he lies'→'they lie'

b. **Initial-CV reduplication and internal placement**
Creek (Riggle 2004, 3)

fayatk + i:→fayat∼fa-k + i:	'crooked'→'crooked (pl.')

c. **Double-sided reduplication**
Nisgha (Urbanczyk 2007, 474)[17]

lú:t'ux^{w}→lúx^{w}∼lút'ux^{w}	'to value'→'to value (pl.)'

These C-Seq processes are visualized in Table 8. Consider penultimate syllable copying in Samoan with two truncation functions a.lo.fa→a.lo.~~fa~~∼~~a~~.lo.fa. If the input is read left-to-right, the first function must output everything up until the penultimate syllable: *a.lo.~~fa~~*. This is not OSL because knowledge about whether some syllable is the penultimate or not requires finite lookahead on the input. If the input is instead read right-to-left, the first truncation function is still not OSL. The function would delete the last vowel and the last consonant; but once it sees the penultimate vowel, the function cannot determine if this vowel is the penultimate or not based on only what it has outputted. The function would need lookback access to the input. In both left-to-right and right-to-left cases, the truncation functions are Seq. The other processes in Table 8 are not C-OSL for similar reasons.

Although penultimate syllable copying is not C-OSL, Samoan has penultimate stress (Zuraw et al. 2014). Stressed syllable copying is C-OSL. This is an argument for reanalyzing Samoan as instead copying the stressed syllable. This relates to Nelson (2003, 117)'s hypothesis that any references to the penultimate position in reduplication must

[17] We set aside issues in predicting the quality of the vowel aside (Shaw 2005).

Case	Samoan (16a)	Creek (18b)	Nisgha (18c)
Input x	a.lo.fa	fayatk	lú:t'ux^w
Components	$L(x) \cdot R(x)$	$L(x) \cdot R(x)$	$L(x) \cdot R(x)$
Infixal treatment	a.lo$<$lo$>$fa	fayat$\smile$fa$\frown$k	lúx$^w\sim$lút'ux^w
C-Seq treatment	a.lo~~.fa~~$\sim$~~a.~~lo.fa	fayat~~k~~$\sim$fa~~yatk~~	lú~~t'ux~~x$^w\sim$lút'ux^w
Subclass	C-Seq	C-Seq	C-Seq

Table 8: Non-C-OSL patterns of internal or non-contiguous reduplication

be prosodic. Note that for Creek and Nisgha, the component functions are Seq however they do not generate the right origin semantics because of unbounded word-internal deletion. The first function for Nisgha deletes everything except for the first CV and last C lu~~t'ux~~x^w. If read left-to-right, in order to generate the final C, we need to move to the next symbol and check if it $\ltimes$; thus the final C is generated as an output correspondent for the end-boundary $\ltimes$. If read left-to-right, in order to generate the initial CV, we move on to the preceding symbol and check if it is $\rtimes$; thus the first CV are generated as output correspondents to the start-boundary $\rtimes$. This is because Seq functions are deterministic. In contrast, the right origin semantics would be generated if the component functions were non-deterministic or if the function was computed by a full 2-way FST.[18]

6.4 *Grey areas between C-Seq and rotating transducers*

The above cases with infixation showed that capturing the right origin semantics might require classes which are more expressive than C-OSL and C-Seq. In this section, we go through more cases. Some are ambiguously C-Seq depending on the analysis; others must use rotating or even unrestricted 2-way D-FSTs in order to capture the right origin semantics.

6.4.1 Reduplication with syllable-count

Reduplication that is sensitive to syllables may involve iteration (19a) or minimality requirements on what is reduplicated (19b). Both examples are from Mandarin for different reduplicative processes. In (19a),

[18] To exactly capture the right origin semantics, it is a possible that a subclass of *Streaming-string transducers* (1-way FSTs with registers) (Alur and Černý 2011; Alur and Deshmukh 2011) are a suitable alternative for modeling infixal reduplication. Discovering subclasses of SSTs and their relations to subclasses of of 2-way D-FSTs is a worthwhile open question.

reduplication is iterative because each syllable undergoes total reduplication: an input of the form A.B has A.A.B.B as the output. In (19b), a word undergoes total reduplication if it is monosyllabic, otherwise the morpheme *meei* is added.

(19) *Reduplication and syllable number*

a. **Iterative reduplication of syllables**
Mandarin (Moravcsik 1978, 314)
huang.jang→huang∼huang-jang∼jang
'flustered'→'flustered (vivid form)'

b. **Minimality in reduplication**
Mandarin (Moravcsik 1978, 305-6).

jang→jang∼jang	'sheet'→'every sheet'
jia.luen→meei-jia.luen	'gallon'→'every gallon'

Iterative copying (19a) is C-OSL if the number of iterations (=number of syllables) is bounded. The individual syllables must also be uniquely identifiable in the input. For Mandarin, the function is made up of four concatenated OSL truncation function. The first two functions output everything up until the medial syllable boundary '.' while the latter two delete everything up until the '.' boundary.

(20) *Mandaring iterative reduplication as C-OSL*:
$L1(x) \cdot L2(x) \cdot R1(x) \cdot R2(x)$
huang.~~jang~~∼huang.~~jang~~∼~~huang~~.jang∼~~huang~~.jang.

We are unaware of any examples showing reduplicative processes which iterate over inputs that have at least three syllables. So while Mandarin provides examples of *A→A∼A* and *A.B→A.A∼B.B*, we have no examples of **A.B.C→A.A∼B.B∼C.C*. We likewise have not seen cases of trisyllabic iterative copying in other languages. This is computationally significant. If Mandarin allowed iterative copying over trisyllabic words, then generating the right origin semantics would need as many passes over the input as there are syllables in the input. The function would either need the full power of a 2-way D-FST in order to generate the right origin semantics, otherwise we could use a 1-way FST that has the linguistically-unmotivated origin semantics.

As for minimality requirements (19b), this cannot be computed by a C-Seq transducer with the right origin semantics. In order to

reduplicate a monosyllabic input: *jang→jang∼jang*, we use two concatenated identity functions. But to block reduplication in a bisyllabic input *jia.luen→meei-jia.luen*, we need to check that the input does not contain any medial syllable boundaries. The first function would need to use to finite lookahead before choosing to output the first segment *j* or the prefix *meei*. As with the infixation cases in §6.3.2, a C-Seq 2-way FST can do so but it then generates the wrong origin semantics because it associates the output segment *j* with the input syllable boundary '.'. Generating the right origin semantics needs a rotating 2-way D-FST that involves three passes. The first pass reads the input and checks if it is monosyllabic or not. If yes, the second and third passes apply the identity function: jang∼jang. If no, the second pass outputs the prefix and the base meei-jia.luen; there is no third pass.

6.4.2 Phonological changes to the reduplicant

The previous section illustrated how C-Seq 2-way FSTs are distinct from 2-way rotating 2-way FSTs. In the latter, a pass can transfer information (e.g., *is the input monosyllabic*) to a later pass. Similar information transfer is required in certain cases where phonological processes interact with reduplication. We first go over cases where we arguably do not need such information transfer.

Reduplicative patterns do not only involve copying. In addition to copying segments, a reduplicative process may involve a host of other phonological transformations (Steriade 1988; Raimy 2011). When reduplication interacts with phonological transformations, the complexity may change. Some of these phonological changes affect only the reduplicant (21), including prosodic modifications to the reduplicant (21a), simplifications of the reduplicant (21b), or creating non-identity across the two copies (21c).[19]

(21) *Phonological modifications within the reduplicant*

a. **Vowel lengthening in the reduplicant**
Papago (Moravcsik 1978, 308,317)

[19] These simplifications are often called TETU (or *the emergence of the unmarked*) effects (McCarthy and Prince 1994, 1995). Cases of echo-reduplication like (21c) are highly common and found even in Indo-European languages such as English *book-**schm**ook* (Nevins and Vaux 2003). It is often sensitive to phrasal or syntactic factors (Fitzpatrick-Cole 1994; Lidz 2001).

Table 9: C-OSL treatment for phonological changes to the reduplicant

	Papago (21a)	Tagalog (21b)	Turkish (21c)
Input x	bana	$\text{mag}\{_r\text{trabaho}\}_r$	kitap
Components	$M(Tr(x)) \cdot ID(x)$	$M(Tr(x)) \cdot R(x)$	$ID(x) \cdot M(ID(x))$
Innermost function	ba ~~na~~	$\text{mag}\{_r\text{tra ~~baho~~}\}_r$	kitap
Composition	baa	$\text{mag}\{_r\text{t ~~r~~ a}$	mitap
Concatenation	baa ∼ bana	mag-ta ∼ ~~mag~~ $\{_r\text{trabaho}\}_r$	kitap∼mitap
Subclass	C-L-OSL	C-L-OSL	C-L-OSL

bana→baa∼bana ‘coyote’→‘coyotes’

b. **Complex onset reduction in the reduplicant**
Tagalog (Rubino 2005, 18)
mag-trabaho→mag-ta∼trabaho ‘work’→‘will work’

c. **Pre-specified segment(s) in the reduplicant**
Turkish (Moravcsik 1978, 323)
kitap→kitap∼mitap ‘book’→‘books and the like’

In (21a), the initial-CV is copied; the reduplicant’s vowel is lengthened. In (21b), the stem’s initial-CV is copied. If the stem starts with a complex onset, then the complex onset is reduced to CV. In (21c), the input undergoes total copying; the second copy starts with /m-/ which replaces any word-initial onset.

These phonological modifications can be understood in terms of function composition as a formal analog to phonological rule ordering (Steriade 1988). This is shown in Table 9. Consider Papago lengthening. This is computed by a C-OSL function which is the concatenation of modified truncation function $M(Tr(x))$ and identity $ID(x)$. The function $M(Tr(x))$ is the composition of a truncation function $Tr(x)$ that deletes the string *bana* to *ba* inside by a modification function $M(x)$ that lengthens word-initial *ba* to *baa*. Both truncation and modification are L-OSL, and their composition is C-L-OSL. Complex onset reduction in Tagalog and echo reduplication in Turkish are likewise C-L-OSL and consist of the concatenation of a composed L-OSL function with some other L-OSL function.

However, this does not mean that all hypothetical cases of reduplicant modifications are C-OSL. Such processes can be C-Seq or higher if composition of truncation or identity function with the modification function is Seq or higher. For example, if complex onset reduction in the reduplicant deleted the first consonant: *mag-trabaho→magra∼trabaha*, this process would be C-Seq and not C-OSL. In the first copy, the truncation function would generate $\text{mag}\{_r\text{tra}$~~*baho*~~, while

the modification function would generate *mag{$_r$~~t~~ra*. Deleting only the root-initial consonant *t* if it precedes a consonant is not OSL because we need finite lookahead on the input. Interestingly, this type of cluster reduction is argued to be unattested in reduplication (Zukoff 2017, 25). This may either be an accidental gap or evidence that reduplication modification must be C-OSL. To our knowledge, there is no a typological survey of attested reduplicant modifications to settle this.

6.4.3 Phonological changes to or across both copies

Phonological changes may likewise affect both copies or apply across the boundary between the copies (22). Some involve a phonological process which is productive in the language (22a), others involve a phonological process which is not found anywhere else in the language outside of reduplication (22b). The former set of cases are often called *normal application* of phonological rules, while the latter are *juncture effects* which are morpheme-specific phonological processes.

(22) *Phonological modifications across the two copies*

a. **Normal application of nasal substitution**
Balangao (McCarthy and Prince 1995, 85)

a. /maN + tagtag/→	[ma + nagtag]	‘running’
b. /maN-RED + tagtag/→	[ma + nagta∼tagtag],	‘running
	*[ma + nagta∼nagtag]	everywhere’

b. **Phonology across the boundary (*juncture effects*)**
Dakota (Inkelas and Zoll 2005, 101)

a. /skokpá→o/-skókpa∼kpa	‘to be scooped out’
b. /čap/→čap∼čap-a	‘trot’
c. /žat/ →žag∼žat-a	‘curved’

In (22a), the prefix ma*N*- can trigger reduplication of the root/stem. Nasal substitution combines the prefix’s nasal with an adjacent voiceless consonant into a single nasal that has the place of articulation of the consonant. Nasal substitution applies only to the segment next to the prefix, regardless of whether that consonant is part of the reduplicant or not. In (22b), the final syllable of the root is copied and placed at end the left edge of the input. If there are two coronals across the reduplicative boundary (b), then the first coronal becomes dorsal. The final /a/ is epenthesized.

Table 10: Order of compositions for different reduplication-phonology interactions

	Normal application	Juncture effect	Over-application
Language	Balangao (22a)	Dakota (22b)	Madurese (23)
Input x	maN$\{_r$tagtag$\}_r$	žat	neyat
Order of composition	1. Copy	1. Copy	1. Modify
	2. Modify	2. Modify	2. Copy
Components	$M(L(x)\cdot R(x))$	$M(L(x)\cdot R(x))$	$L(M(x))\cdot R(M(x))$
Innermost functions	maN$\{_r$tagtag$\}_r$ ~ ~~man~~ $\{_r$tagtag$\}_r$	žat ~ žata	nẽỹãt
Outer function	ma~~N~~$\{_r$ŋagtag$\}_r$ ~ $\{_r$tagtag$\}_r$	žag ~ žata	~~nẽỹãt~~∼nẽỹãt

We likewise find phonological processes or rules interacting *differently* in the context of reduplication. For example in Madurese, there is a phonological process of nasal spread in which nasality is spread from nasals onto sequences of glides and vowels (23a). Reduplication copies the final CVC and places it at the beginning of the output (23b). If a vowel in the base is nasalized by a nasal, its nasality will transfer to the reduplicant as well. Because the reduplicant does not contain any nasals to trigger nasal spread, nasal spread in the reduplicant is treated as an over-application of the phonological process of nasal spread.

(23) **Over-application of nasal spread**
Madurese (McCarthy and Prince 1995, 30),(Cohn 1993, 358)
a. /neyat/→ [nẽỹãt] ‘intention
b. /RED-neyat/→ [ỹãt∼nẽỹãt] ‘intentions’

Traditionally, these cases can be thought as a composition of a morphological rule of reduplication (C-OSL) and a phonological rule (that is independently ISL, OSL, or Sequential) (Raimy 2000; Inkelas and Zoll 2005). If the morphological rule precedes the phonological rule, then we have normal application. If the morphological rule outputs reduplicant boundaries and precedes the phonological rule, then we have juncture effects. And, if the phonological rule precedes the morphological rule then we have over-application. Table 10 visualizes these three types of interactions as rule or function composition.

Computationally, we can treat all these cases as composition of a C-OSL/C-Seq function for reduplication with an OSL/Seq function for phonology in either order. We conjecture C-OSL/C-Seq functions are not closed under composition, but we do not prove it. This means that composition may create a rotating 2-way FST.

Whether a case of normal application, juncture effect, or overapplication is C-OSL, C-Seq, or higher depends on the complexity of the individual functions. In fact, the above three examples can be done

with a C-Seq transducer. To illustrate, consider over-application in Madurese. Nasal spread is an L-OSL function $M(x)$. Reduplication is the concatenation of an R-OSL truncation function $L(x)$ and an L/R-OSL identity function $R(x)$. To generate overapplication, reduplication is instead the concatenation of two *modified* functions $L(M(x)) \cdot R(M(x))$. The first function is the composition of truncation over nasal spread. The composition of these two OSL rules of different directions is L-Seq because nasalization relies on deleted information from the input.[20] The second function is the composition of identity over nasal spread; this is L-OSL. Together, Madurese overapplication is C-L-Seq.

It is an open question if there are cases of normal application, juncture effects, and over-application which *cannot* be treated with a C-Seq formalization but require an unrestricted rotating 2-way FST. Solving this require an in-depth knowledge of both the morphology and phonology of any such example (Inkelas and Zoll 2005).

6.4.3.1 Under-application and Back-copying

In contrast to the over-application of phonological processes in reduplication, we likewise find cases of *under-application*. For example in Akan, velar consonants become palatalized before nonlow front vowels: /k,g/→[tɕ, dʝ]/ _ /i,e/ as in (24a). Akan likewise has a process of initial-CV reduplication where the reduplicant V is a pre-specified non-low front vowel /ɪ/ (24b).[21] However if the reduplicant C is a velar, it will not be palatalized before the reduplicant's non-low front vowel /ɪ/ (24c). Thus the rule under-applies. The velar will only palatalize if both copies of the velar in the reduplicant and base are preceded by a non-low front vowel (24d).

(24) **Under-application of palatalization in reduplication**
Akan (McCarthy and Prince 1995, 83-93)
(Schachter and Fromkin 1968, 89))
a. /ke/→ [tɕɪ] 'divide'

[20] As with infixal reduplication (§6.3.2), the C-Seq transducer needs finite look-ahead into the end boundary ⋉ and this makes it not have the exact origin semantics that we want.

[21] The reduplicant V in Akan gets its front/back features from vowel harmony. For illustration, we represent it simply as /ɪ/.

b. /si/→ [si∼siʔ] ‘stand’
c. /kaʔ/→ [kɪ∼kaʔ],*[tɕɪ∼kaʔ] ‘bite’
d. /ge/→ [dʝɪ∼dʝe] ‘receive’

Cases of apparent under-application or over-application in reduplication are termed *opacity effects* (cf. the transparency of normal application (22a)). They are often understood as being caused by a need to maintain identity between the two copies that reduplication created (Wilbur 1973; McCarthy and Prince 1995). A more drastic version of identity is back-copying whereby the reduplicant undergoes some phonological rule, and then the effects of this rule are transferred onto the base. It is reported that in Malay, nasality spreads from a nasal consonant onto a sequence of vowels. Nasality can spread over glides and /h/. Plurality is marked by total reduplication. If nasal spread applies across the two copies, nasality will transfer onto both copies.

(25) **Back-copying of nasal spread**
Malay (McCarthy and Prince 1995, 85)
a. /hamə/→ [hãmə̃∼hãmə̃],*[hamə̃∼hãmə̃] ‘germ’→‘germs’

These opacity effects are controversial both theoretically and empirically (Inkelas and Zoll 2005; Samuels 2010; Kiparsky et al. 2010; McCarthy et al. 2012). Many cases of under-application have been reanalyzed as either unproductive (McCarthy et al. 2012) or due to morpheme-specific rules (Inkelas and Zoll 2005).[22] In fact, Akan palatalization (24) is the classical case study on under-application but it is likely a synchronically unproductive and fossilized rule (Silverman 2002; Adomako 2018) Empirically, there have been little if any convincing cases of back-copying and some are arguably due to morphological factors outside of reduplication (McLaughlin 2005). The Malay data itself has not been successively reproduced (Kiparsky et al. 2010).

Because under-application and back-copying have weak empirical backing, there is a limited attested typology of these processes. It is thus unclear whether we can make any computational generalizations about them. But putting aside these empirical problems, Akan under-application can be modeled with a C-Seq function which uses

[22] An exception is Tonkawa (Gouskova 2007) which is arguably a bona fide case of under-application.

finite lookahead on the input. It is the concatenation of a modified truncation function and a modified identity function. The first function truncates the input $C_1V_2\Sigma^*$ to C_1i and applies palatalization if V_2 is /i,e/. The second function applies palatalization to the input.

Malay back-copying is not C-Seq. This is because nasalization requires unbounded lookahead on the input. The function requires an unrestricted rotating transducer with three passes over the input.[23] In the first pass, we output nothing but we check if the input ends in a N(V+G)* sequence where G stands for glides and /h/: ~~hamə~~. If yes, the second pass applies nasal spread starting from the first segment: ~~hamə~~∼h̃ãmə̃. The third pass does the same: ~~hamə~~∼h̃ãmə̃∼h̃ãmə̃.

6.5 *Overview of the typology summary*

To summarize, we cataloged a wide variety of attested reduplicative patterns. All of it can be computed with deterministic 2-way FSTs. Most common and uncommon types of reduplication can be computed with the subclass of C-OSL functions, including total reduplication (12a), common partial reduplication patterns (13), triplication (14a), suffixal reduplication (14b), non-local reduplication (14c), and abbreviated reduplication (14d). Subconstituent reduplication is likewise C-OSL if the relevant morphological (15) or prosodic boundaries (16) are present in the input. In fact, of the 138 reduplicative processes in RedTyp (§5.2), 121 (87%) are C-OSL.[24]

We analyzed the typology in terms of generating the right origin semantics. To do so, some less common types of reduplication are C-OSL, C-Seq, or higher. This is largely because of the need for finite lookahead on the input. Some but not all types of infixal or non-contiguous reduplication is C-OSL (§6.3.1) and some are C-Seq (§6.3.2). In the latter case, generating the right origin semantics can

[23] Malay back-copying can likewise be treated as the composition of a C-OSL function for triplication hamə∼hamə∼hamə, followed by an OSL function for nasal spread hamə̃∼hãmə̃∼hãmə̃, followed by an OSL function that deletes everything before the first ∼ boundary hãmə̃∼hãmə̃. This analysis is inspired by Reiss and Simpson (2009).

[24] Although the cross-linguistic typology on reduplication is overwhelmingly C-OSL, our numbers from RedTyp do not mean that we estimate that 13% of the cross-linguistic typology of reduplicative processes is not C-OSL. RedTyp likely under-represents cases of opacity. Such cases can be non-C-OSL.

require full 2-way FSTs because of the need for finite lookahead. Some cases like iterative reduplication (19a) are C-OSL if the input is at most bisyllabic; otherwise generating the right origin semantics needs an unrestricted 2-way D-FST. Minimality requirements (19b) likewise require full 2-way D-FSTs. When reduplication interacts with phonological processes, the computation can range anywhere from C-OSL or full 2-way D-FSTs depending on the individual phonological process and the order of function composition.

This concludes the section on how various subclasses of 2-way D-FSTs map to certain divisions in the reduplicative typology. The above cases are representative of common and uncommon reduplicative processes. There are other subtle variations for reduplication in natural language, such cases of allomorphy (Spaelti 1997), multiple reduplicants (Urbanczyk 1999; Urbancyzk 2001; Fitzpatrick and Nevins 2004; Fitzpatrick 2006), among others. We will not discuss these cases because a full typology is beyond the scope of this paper. However, all virtually all attested reduplicative processes can be modeled with 2-way FSTs. Fitting the *entire* attested typology into the right subclasses is a fruitful research direction. The next section looks at cases where 2-way D-FSTs arguably over-generate or under-generate the typology, even with these well-defined subclasses.

6.6 *Issues in over- and under-generation*

Here, we address the questions whether and how 2-way D-FSTs under- and over-generate reduplicative processes.

6.6.1 Over-generation with 2-way D-FSTs

One way to interpret the contribution we have made is that we are advocating the following hypothesis:

(26) **2-way Hypothesis:** Reduplication is whatever that can be computed with 2-way D-FSTs.

This is not, in fact, a position we advocate. We think this hypothesis is false because it overgenerates in ways we think are linguistically bizarre. For example, 2-way D-FSTs can map words to their reverse ($w \mapsto w^r$) and to a copy of itself and its mirror image ($w \mapsto ww^r$). None of these transformations are attested morphologically. Some overgeneration can be avoided by hypothesizing stronger computational prop-

erties; that is, focusing on *subclasses* of 2-way D-FSTs which cannot generate the above unattested patterns.

This leads to another hypothesis.

(27) **C-OSL Hypothesis:** Reduplication is whatever that can be computed with C-OSL 2-way D-FSTs.

The C-OSL hypothesis is well supported because it covers the bulk of reduplicative typology as shown in §6. Rarer reduplicative patterns require more powerful subclasses of 2-way D-FSTs.

Even this hypothesis can be said to suffer from overgeneration. For example, while this excludes the reversal and mirror image processes above, it permits total reduplication of a word up to some large natural number n ($w \mapsto w^n$), or partial reduplication up to some natural number n of segments.

Nonetheless, not all issues in overgeneration can be reduced to computation or computability. Some are certainly due to external factors. To illustrate, total reduplication in most *spoken* languages creates at most two copies. The creation of three copies (= triplication) is relatively rare in spoken languages, e.g. Thao (Blust 2001). In sign languages, we find the reverse situation: creating two copies is rare but triplication is common, e.g. ASL (Wilbur 2005). The difference between sign and spoken reduplication is more likely due to modality and not to the computation.[25]

6.6.2 Under-generation with 2-way D-FSTs

Non-computational factors can also help us understand apparent cases of under-generation. There are two cases we discuss here: abstract morphemic copying and reduplication with haplology. Both of these can be explained as involving interactions between reduplication and other linguistic modules (the lexicon) or processes (filters).

6.6.2.1 Undergenaration of abstract morphemic copying

Abstract morphemic copying is when the input to the copying mecha-

[25] A similar point can be made for the role of pivot or anchor points in reduplication. Cross-linguistically, most reduplicative processes target specific positions in the word which are perceptually or psycho-linguistically more salient (Samuels 2010; Raimy 2009; Idsardi and Raimy 2008), e.g. the first syllable and not the third syllable. The choice of these pivots is likely functional, not computational.

Table 11: Abstract morphemic copying in Sye

Morphemes	a.	/√FALL/	b.	/√FALL + RED/	c.	/FUT-√FALL-RED/
Output		omol		omol∼omol		cw-*amol*∼*omol*
						*[cw-*omol*∼*omol*
						*[cw-*amol*∼*amol*
Gloss		'fall'		'fall all over'		'they will fall all over'

nism is not a string of phonological segments but a more abstract morphological entity, i.e. a morpheme or morpho-syntactic feature bundle (Inkelas and Zoll 2005). This is in contrast to examples reviewed earlier, where the source of reduplication was a string of segments which may contain morpheme boundaries. Such a case occurs in Sye in Table 11.

In Sye, a stem may have multiple suppletive allomorphs used in different morphological contexts. For example, the abstract morpheme √FALL in Table 11a has two allomorphs *amol* and *omol*, such that *amol* is used after future morpheme and certain other tense morphemes while *omol* is used elsewhere. As for reduplication, total reduplication is used to mark intensification (Table 11b), When total reduplication applies in a context that requires using one of the allomorphs, we have an allomorph mismatch between the two copies (Table 11c).

Inkelas and Zoll (2005) analyze Sye as involving morphological copying. The copies are not in phonological correspondence because they are different allomorphs of the same morpheme. What was copied was an abstract morpheme √FALL. Its two copies were later spelled-out as two different allomorphs. Inkelas and Zoll (2005)'s analysis for Sye is controversial (Frampton 2009); but there are a few other languages which show that the reduplicant is copying an abstract morphological entity (Inkelas and Downing 2015a,b; Hyman et al. 2008).

Cases of morphological copying for *suppletive* roots can be modeled with a 2-way D-FST that copies an abstract pre-spelled-out morphological entity, e.g. a root morpheme √FALL or a root index (Harley 2014) which can be represented as a finite string of symbols. This is followed by a 1-way FST that models spell-out such that it is equipped with knowledge over what all the *finite* pairs of morphemes and their suppletive allomorphs are. We make the safe assumption that the number of morphemes in a language that show suppletion is finite.[26]

[26] An FST which handles spell-out would resemble the lexical transducer used in the `xfst` finite-state package (Beesley and Karttunen 2003).

6.6.2.2 Undergeneration of reduplication with haplology

Another case of potential under-generation is when reduplication is affected by anti-homophony constraints or haplology, i.e. when reduplication is blocked because it would create a sequence of identical syllables or feet that is dis-preferred by speakers (Yip 1995; Nevins 2012).

For example in Kanuri (Moravcsik 1978, 313), total reduplication is used to form glossonyms (28b). However reduplication is blocked if it creates sequence of identical syllables/feet (28b).

(28) Reduplication and haplology in Kanuri

a.	kanəmbu	'Kanembu tribe'
	kanəmbu∼kanəmbu	'language of the Kanembu tribe'
b.	karekare	'Karekare tribe'
	*karekare∼karekare	'language of the Karekare tribe'

A 2-way D-FST can encode this requirement that the input must not itself be a sequence of identical syllables or feet. However that would require that the 2-way D-FST knows what all the finitely possible sequences of syllables and feet are in the language.

Two alternatives to this solution are possible. One is using a copy-and-filter mechanism (Golston 1995). The 2-way D-FST would handle the copying. The output of the copying process would be fed to a phonological system which would filter out any homophonous sequences of syllables or feet. The other alternative is to argue that the Kanuri input stems which contain a repeated sequence of symbols /karekare/ are underlyingly already reduplicated via lexical reduplication /kare + RED/. Such arguments have been brought up for superficially similar haplology effects in Manam (Buckley 1997). From this approach, there is then no haplology problem for 2-way D-FSTs.

In sum, although virtually the entire typology of reduplication can be modeled with 2-way D-FSTs, there are complications if one wishes to model the full interface of reduplicative morphology with other systems. However, as a reviewer point outs, it may not be desirable, from a linguistic perspective at least, to model the interfaces in this way. On the other side of the coin, some conceptual problems arise

with over-generation of 2-way D-FSTs because of the power that they require to handle copying in the first place.

7 CONCLUSION

The present study has taken a step in formalizing the wide typology of reduplicative processes in formal-language theoretic terms. We showed that 2-way D-FSTs, which are are an understudied type of finite-state transducer, can easily model reduplication because they can reread their input multiple times in multiple directions. In addition to modeling reduplicative morphology as copying, 2-way D-FSTs do not suffer from state explosion nor do they assume finite bounds on the input, unlike 1-way FSTs. In terms of strong generative capacity, 2-way FSTs actively copy segments instead of memorizing segments. A diagnostic for copying vs. remembering is the origin semantics of the function.

This article also presented the RedTyp database, which provides concrete examples of 2-way DFSTs modeling a range of cross-linguistic reduplicative morphemes. Furthermore, we showed that the typology of reduplication can be modeled with subclasses of 2-way FSTs that are essentially defined as concatenations of simple subclasses of 1-way FSTs.

Having showcased the utility of 2-way D-FSTs for modeling reduplication, we conclude with three avenues of future research.q

First, we have approached reduplication from the perspective of morphological generation. Given an input buku, a 2-way D-FST can generate the output buku∼buku easily. On the other hand, it is an open question as to how to do morphological analysis with 2-way FSTs to get the inverse relation of buku∼buku→buku.

A second area of research is the integration of 2-way FSTs into natural language processing. This obviously has many aspects. A first step may be the integration of 2-way FSTs into existing platforms such as `xfst` (Beesley and Karttunen 2003), `open-fst` (Allauzen et al. 2007), `foma` (Hulden 2009b), and `pynini` (Gorman 2016).[27]

[27] In fact, the team behind Thrax (Tai et al. 2011) have recently been exploring the use of multi-pushdown transducers (MPDT) to generate reduplication (Richard Sproat, p.c.). An open question is comparing the generative capacity of

A third promising area of research is developing learning models based on the computational models we developed here. One approach builds on Chandlee et al's 2015 learning results of OSL functions (Dolatian and Heinz 2018). Another approach probes the learnability of reduplicative patterns with neural networks (Nelson et al. 2020).

REFERENCES

Kwasi ADOMAKO (2018), Velar palatalization in Akan: A reconsideration, Journal of West African Languages, 45(2).

Alfred V. AHO, John E. HOPCROFT, and Jeffrey D. ULLMAN (1969), A general theory of translation, Mathematical Systems Theory, 3(3):193–221.

Daniel M. ALBRO (2000), Taking Primitive Optimality Theory Beyond the Finite State, in Jason EISNER, Lauri KARTTUNEN, and Alain THÉRIAULT, editors, Finite-state phonology: proceedings of the 5th Workshop of SIGPHON, pp. 57–67, Luxembourg, URL http://aclanthology.coli.uni-saarland.de/pdf/W/W00/W00-1806.pdf.

Daniel M. ALBRO (2005), Studies in Computational Optimality Theory, with Special Reference to the Phonological System of Malagasy, Ph.D. thesis, University of California, Los Angeles, Los Angeles.

Raquel G ALHAMA and Willem ZUIDEMA (2019), A review of computational models of basic rule learning: The neural-symbolic debate and beyond, Psychonomic bulletin & review, pp. 1–21.

Raquel Garrido ALHAMA (2017), Computational modelling of Artificial Language Learning: Retention, Recognition & Recurrence, Ph.D. thesis, Universiteit van Amsterdam.

Cyril ALLAUZEN, Michael RILEY, Johan SCHALKWYK, Wojciech SKUT, and Mehryar MOHRI (2007), OpenFst: A general and efficient weighted finite-state transducer library, in Jan HOLUB and Jan ŽĎÁREK, editors, Implementation and Application of Automata, pp. 11–23, Springer, Berlin, Heidelberg.

Rajeev ALUR (2010), Expressiveness of streaming string transducers, in Proceedings of the 30th Annual Conference on Foundations of Software Technology and Theoretical Computer Science,, volume 8, p. 1–12, doi:10.4230/LIPIcs.FSTTCS.2010.1, URL http://dx.doi.org/10.4230/LIPIcs.FSTTCS.2010.1.

Rajeev ALUR and Jyotirmoy V. DESHMUKH (2011), Nondeterministic Streaming String Transducers, in Luca ACETO, Monika HENZINGER, and Jiří SGALL, editors, Automata, Languages and Programming, pp. 1–20, Springer, Berlin, Heidelberg, ISBN 978-3-642-22012-8.

MPDTs and 2-way FSTs.

Rajeev ALUR, Adam FREILICH, and Mukund RAGHOTHAMAN (2014), Regular Combinators for String Transformations, in Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, pp. 9:1–9:10, ACM, Vienna, Austria, ISBN 978-1-4503-2886-9, doi:10.1145/2603088.2603151, URL http://doi.acm.org/10.1145/2603088.2603151.

Rajeev ALUR and Pavol ČERNÝ (2011), Streaming Transducers for Algorithmic Verification of Single-pass List-processing Programs, in Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '11, pp. 599–610, ACM, Austin, Texas, USA, ISBN 978-1-4503-0490-0, doi:10.1145/1926385.1926454, URL http://doi.acm.org/10.1145/1926385.1926454.

Qatherine ANDAN, Outi BAT-EL, Diane BRENTARI, and Iris BERENT (2018), ANCHORING is amodal: Evidence from a signed language, Cognition, 180:279–283.

Stephen R ANDERSON (1992), A-morphous morphology, volume 62, Cambridge University Press, Cambridge.

Mark ARONOFF (1988), Head operations and strata in reduplication: A linear treatment, in Geert BOOIJ and Jaap VAN MARLE, editors, Yearbook of Morphology, volume 1, pp. 1–15, Foris, Dordrecht.

Félix BASCHENIS, Olivier GAUWIN, Anca MUSCHOLL, and Gabriele PUPPIS (2015), One-way definability of sweeping transducers, in 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Compu Bangalore, India, URL https://hal.archives-ouvertes.fr/hal-01219509.

Félix BASCHENIS, Olivier GAUWIN, Anca MUSCHOLL, and Gabriele PUPPIS (2016), Minimizing Resources of Sweeping and Streaming String Transducers, in Ioannis CHATZIGIANNAKIS, Michael MITZENMACHER, Yuval RABANI, and Davide SANGIORGI, editors, 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), volume 55 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 114:1–114:14, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, ISBN 978-3-95977-013-2, ISSN 1868-8969, doi:10.4230/LIPIcs.ICALP.2016.114, URL http://drops.dagstuhl.de/opus/volltexte/2016/6249.

Félix BASCHENIS, Olivier GAUWIN, Anca MUSCHOLL, and Gabriele PUPPIS (2017a), One-way definability of two-way word transducers, CoRR, abs/1706.01668, URL http://arxiv.org/abs/1706.01668.

Félix BASCHENIS, Olivier GAUWIN, Anca MUSCHOLL, and Gabriele PUPPIS (2017b), Untwisting two-way transducers in elementary time, in 32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017, pp. 1–12, IEEE Computer Society, ISBN

978-1-5090-3018-7, doi:10.1109/LICS.2017.8005138, URL https://doi.org/10.1109/LICS.2017.8005138.

Kenneth BEESLEY and Lauri KARTTUNEN (2003), Finite-state morphology: Xerox tools and techniques, CSLI Publications, Stanford, CA.

Kenneth R. BEESLEY and Lauri KARTTUNEN (2000), Finite-state non-concatenative morphotactics, in Proceedings of the 38th Annual Meeting on Association for Computational Linguistics, ACL '00, pp. 191–198, Association for Computational Linguistics, Hong Kong, doi:10.3115/1075218.1075243, URL https://doi.org/10.3115/1075218.1075243.

Iris BERENT, Outi BAT-EL, Diane BRENTARI, Amanda DUPUIS, and Vered VAKNIN-NUSBAUM (2016), The double identity of linguistic doubling, Proceedings of the National Academy of Sciences, 113(48):13702–13707.

Iris BERENT, Outi BAT-EL, and Vered VAKNIN-NUSBAUM (2017), The double identity of doubling: Evidence for the phonology–morphology split, Cognition, 161:117–128.

Iris BERENT, Amanda DUPUIS, and Diane BRENTARI (2014), Phonological reduplication in sign language: Rules rule, Frontiers in psychology, 5:560.

Steven BIRD and T Mark ELLISON (1994), One-level phonology: Autosegmental representations and rules as finite automata, Computational Linguistics, 20(1):55–90.

Robert A BLUST (2001), Thao triplication, Oceanic Linguistics, 40(2):324–335.

Mikołaj BOJAŃCZYK (2014), Transducers with origin information, in Javier ESPARZA, Pierre FRAIGNIAUD, Thore HUSFELDT, and Elias KOUTSOUPIAS, editors, Automata, Languages, and Programming, pp. 26–37, Springer, Berlin, Heidelberg.

Mikolaj BOJANCZYK, Laure DAVIAUD, Bruno GUILLON, and Vincent PENELLE (2017), Which Classes of Origin Graphs Are Generated by Transducers, in Ioannis CHATZIGIANNAKIS, Piotr INDYK, Fabian KUHN, and Anca MUSCHOLL, editors, 44th International Colloquium on Automata, Languages, and Programming (ICALP 2017), volume 80 of Leibniz International Proceedings in Informatics (LIPIcs), pp. 114:1–114:13, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, ISBN 978-3-95977-041-5, ISSN 1868-8969, doi:10.4230/LIPIcs.ICALP.2017.114, URL http://drops.dagstuhl.de/opus/volltexte/2017/7398.

Olivier BONAMI and Berthold CRYSMANN (), The role of morphology in constraint-based lexicalist grammars, in Andrew HIPPISLEY and Gregory T. STUMP, editors, The Cambridge Handbook of Morphology, pp. 609–656), address = Cambridge, publisher = Cambridge University Press.

Olivier BONAMI and Berthold CRYSMANN (2013), Morphotactics in an information-based model of realisational morphology, in Proceedings of HPSG, volume 34, pp. 27–47, CSLI Publications, Stanford, CA.

Ellen Broselow and John McCarthy (1983), A theory of internal reduplication, The Linguistic Review, 3(1):25–88.

Jason Brown (2017), Non-adjacent reduplication requires spellout in parallel, Natural Language & Linguistic Theory, pp. 1–23.

Eugene Buckley (1997), Integrity and correspondence in Manam double reduplication, in Proceedings of NELS, volume 28, pp. 59–67.

Gabriela Caballero (2006), "Templatic backcopying" in Guarijio abbreviated reduplication, Morphology, 16(2):273–289.

Jane Chandlee (2014), Strictly Local Phonological Processes, Ph.D. thesis, University of Delaware, Newark, DE.

Jane Chandlee (2017), Computational locality in morphological maps, Morphology, 27(4):1–43.

Jane Chandlee, Angeliki Athanasopoulou, and Jeffrey Heinz (2012), Evidence for Classifying Metathesis Patterns as Subsequential, in Jaehoon Choi, E. Alan Hogue, Jeffrey Punske, Deniz Tat, Jessamyn Schertz, and Alex Trueman, editors, The Proceedings of the 29th West Coast Conference on Formal Linguistics, pp. 303–309, Cascillida Press, Somerville, MA.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz (2014), Learning Strictly Local Subsequential Functions, Transactions of the Association for Computational Linguistics, 2:491–503, URL `http://aclweb.org/anthology/Q14-1038`.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz (2015), Output Strictly Local Functions, in Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015), pp. 112–125, Chicago, USA.

Jane Chandlee and Jeffrey Heinz (2012), Bounded copying is subsequential: Implications for metathesis and reduplication, in Proceedings of the 12th Meeting of the ACL Special Interest Group on Computational Morphology and Phonology, SIGMORPHON '12, pp. 42–51, Association for Computational Linguistics, Montreal, Canada, URL `http://dl.acm.org/citation.cfm?id=2390930.2390936`.

Jane Chandlee and Jeffrey Heinz (2018), Strict locality and phonological maps, Linguistic Inquiry, 49(1):23–60.

Jane Chandlee, Jeffrey Heinz, and Adam Jardine (2018), Input strictly local opaque maps, Phonology, 35(2):171–205.

Christian Choffrut (1977), Une caractérisation des fonctions séquentielles et des fonctions sous-séquentielles en tant que relations rationnelles, Theoretical Computer Science, 5(3):325–337, ISSN 0304-3975, doi:https://doi.org/10.1016/0304-3975(77)90049-4, URL `http://www.sciencedirect.com/science/article/pii/0304397577900494`.

Michal P. Chytil and Vojtěch Jákl (1977), Serial composition of 2-way finite-state transducers and simple programs on strings, in Arto Salomaa and Magnus Steinby, editors, Automata, Languages and Programming, pp. 135–147, Springer, Berlin, Heidelberg, ISBN 978-3-540-37305-6.

Alexander Clark (2017), Computational learning of syntax, Annual Review of Linguistics, 3:107–123.

Alexander Clark and Ryo Yoshinaka (2012), Beyond semilinearity: Distributional learning of parallel multiple context-free grammars, in International Conference on Grammatical Inference, pp. 84–96.

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

Alexander Clark and Ryo Yoshinaka (2016), Distributional learning of context-free and multiple context-free grammars, in Jeffrey Heinz and José M. Sempere, editors, Topics in Grammatical Inference, pp. 143–172, Springer, Berlin, Heidelberg.

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

Abigail C Cohn (1989), Stress in Indonesian and bracketing paradoxes, Natural language & linguistic theory, 7(2):167–216.

Abigail C Cohn (1993), The status of nasalized continuants, in Marie K Huffman and Rena A Krakow, editors, Nasals, nasalization, and the velum, volume 5 of Phonetics and Phonology, pp. 329–367, Academic Press, Inc., San Diego, CA.

Bruno Courcelle and Joost Engelfriet (2012), Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach, Cambridge University Press.

Berthold Crysmann (2017), Reduplication in a computational HPSG of Hausa, Morphology, 27(4):527–561.

Berthold Crysmann and Olivier Bonami (2016), Variable morphotactics in information-based morphology, Journal of Linguistics, 52(2):311–374.

Karel Culik and Juhani Karhumäki (1986), The equivalence of finite valued transducers (on HDT0L languages) is decidable, Theoretical Computer Science, 47:71 – 84, ISSN 0304-3975, doi:https://doi.org/10.1016/0304-3975(86)90134-9, URL `http://www.sciencedirect.com/science/article/pii/0304397586901349`.

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

Vrunda DAVE, Paul GASTIN, and Shankara Narayanan KRISHNA (2018), Regular Transducer Expressions for Regular Transformations, in Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '18, pp. 315–324, ACM, New York, NY, USA, ISBN 978-1-4503-5583-4, doi:10.1145/3209108.3209182, URL http://doi.acm.org/10.1145/3209108.3209182.

Hossep DOLATIAN and Jeffrey HEINZ (2018), Learning reduplication with 2-way finite-state transducers, in Olgierd UNOLD, Witold DYRKA, , and Wojciech WIECZOREK, editors, Proceedings of Machine Learning Research: International Conference on Grammatical Inference, volume 93 of Proceedings of Machine Learning Research, pp. 67–80, Wroclaw, Poland.

Laura J DOWNING (1998), Prosodic misalignment and reduplication, in Geert BOOIJ and Jaap VAN MARLE, editors, Yearbook of Morphology 1997, pp. 83–120, Kluwer Academic Publishers, Dordrecht.

Laura J DOWNING (2000), Morphological and prosodic constraints on Kinande verbal reduplication, Phonology, 17(01):1–38.

Laura J DOWNING (2003), Compounding and tonal non-transfer in Bantu languages, Phonology, 20(1):1–42.

Laura J DOWNING (2006), Canonical forms in prosodic morphology, number 12 in Oxford studies in Theoretical Linguistics, Oxford University Press.

Calvin C ELGOT and Jorge E MEZEI (1965), On relations defined by generalized finite automata, IBM Journal of Research and development, 9(1):47–68.

Joost ENGELFRIET and Hendrik Jan HOOGEBOOM (2001), MSO Definable String Transductions and Two-way Finite-state Transducers, ACM Trans. Comput. Logic, 2(2):216–254, ISSN 1529-3785, doi:10.1145/371316.371512, URL http://doi.acm.org/10.1145/371316.371512.

Emmanuel FILIOT and Pierre-Alain REYNIER (2016), Transducers, Logic and Algebra for Functions of Finite Words, ACM SIGLOG News, 3(3):4–19, ISSN 2372-3491, doi:10.1145/2984450.2984453, URL http://doi.acm.org/10.1145/2984450.2984453.

Justin FITZPATRICK (2006), Sources of Multiple Reduplication in Salish and Beyond, Studies in Salishan 7, pp. 211–240.

Justin FITZPATRICK and Andrew NEVINS (2004), Linearizing nested and overlapping precedence in multiple reduplication, in University of Pennsylvania Working Papers in Linguistics, pp. 75–88.

Jennifer FITZPATRICK-COLE (1994), The prosodic domain hierarchy in reduplication, Ph.D. thesis, Stanford University, Stanford, CA.

John FRAMPTON (2009), Distributed reduplication, MIT Press.

Diamandis GAFOS (1998), A-templatic reduplication, Linguistic Inquiry, 29(3):515–527.

Brian GAINOR, Regine LAI, and Jeffrey HEINZ (2012), Computational Characterizations of Vowel Harmony Patterns and Pathologies, in Jaehoon CHOI, E. Alan HOGUE, Jeffrey PUNSKE, Deniz TAT, Jessamyn SCHERTZ, and Alex TRUEMAN, editors, The Proceedings of the 29th West Coast Conference on Formal Linguistics, pp. 63–71, Cascillida Press, Somerville, MA.

Pedro GARCIA, Enrique VIDAL, and José ONCINA (1990), Learning Locally Testable Languages in the Strict Sense, in Proceedings of the Workshop on Algorithmic Learning Theory, pp. 325–338.

Gerald GAZDAR and Geoffrey K PULLUM (1985), Computationally relevant properties of natural languages and their grammars, in The Formal complexity of natural language, pp. 387–437, Springer.

Jila GHOMESHI, Ray JACKENDOFF, Nicole ROSEN, and Kevin RUSSELL (2004), Contrastive focus reduplication in English (the salad-salad paper), Natural Language & Linguistic Theory, 22(2):307–357.

Chris GOLSTON (1995), Syntax outranks phonology: evidence from Ancient Greek, Phonology, 12(3):343–368.

Kyle GORMAN (2016), Pynini: A Python library for weighted finite-state grammar compilation, in Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata, pp. 75–80, Association for Computational Linguistics, Berlin, Germany, doi:10.18653/v1/W16-2409, URL `http://www.aclweb.org/anthology/W16-2409`.

Maria GOUSKOVA (2007), The reduplicative template in Tonkawa, Phonology, 24(3):367–396.

Jiatao GU, Zhengdong LU, Hang LI, and Victor O.K. LI (2016), Incorporating Copying Mechanism in Sequence-to-Sequence Learning, in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 1631–1640, Association for Computational Linguistics, Berlin, Germany.

Heidi HARLEY (2014), On the identity of roots, Theoretical linguistics, 40(3/4):225–276.

Jason D HAUGEN (2009), What is the Base for Reduplication?, Linguistic Inquiry, 40(3):505–514.

Jeffrey HEINZ (2007), The Inductive Learning of Phonotactic Patterns, Ph.D. thesis, University of California, Los Angeles.

Jeffrey HEINZ (2018), The computational nature of phonological generalizations, in Larry HYMAN and Frans PLANK, editors, Phonological Typology, Phonetics and Phonology, chapter 5, pp. 126–195, De Gruyter Mouton.

Jeffrey HEINZ and William IDSARDI (2013), What Complexity Differences Reveal About Domains in Language, Topics in Cognitive Science, 5(1):111–131.

Jeffrey HEINZ and Regine LAI (2013), Vowel Harmony and Subsequentiality, in Andras KORNAI and Marco KUHLMANN, editors, Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13), pp. 52–63, Association for Computational Linguistics, Sofia, Bulgaria, URL http://www.aclweb.org/anthology/W13-3006.

John E HOPCROFT and Jeffrey D ULLMAN (1969), Formal languages and their relation to automata, Addison-Wesley Longman Publishing Co., Inc., Boston:MA.

Mans HULDEN (2006), Finite-State Syllabification, in Anssi YLI-JYRÄ, Lauri KARTTUNEN, and Juhani KARHUMÄKI, editors, Finite-State Methods and Natural Language Processing. FSMNLP 2005. Lecture Notes in Computer Science, volume 4002, Springer, Berlin/Heidelberg.

Mans HULDEN (2009a), Finite-state machine construction methods and algorithms for phonology and morphology, Ph.D. thesis, The University of Arizona, Tucson, AZ.

Mans HULDEN (2009b), Foma: a Finite-State Compiler and Library, in Proceedings of the Demonstrations Session at EACL 2009, pp. 29–32, Association for Computational Linguistics, Athens, Greece, URL http://www.aclweb.org/anthology/E09-2008.

Mans HULDEN and Shannon T BISCHOFF (2009), A Simple Formalism for Capturing Reduplication in Finite-State Morphology, in Jakub PISKORSKI, Bruce WATSON, and Anssi YLI-JYRÄ, editors, Proceedings of the 2009 conference on Finite-State Methods and Natural Language Processing: Post-proceedings of the 7th International Workshop FSMNLP 2008, pp. 207–214, IOS Press, Amsterdam, ISBN 978-1-58603-975-2, URL http://dl.acm.org/citation.cfm?id=1564035.1564059.

Bernhard HURCH, editor (2005), 28, Walter de Gruyter, Berlin.

Bernhard HURCH (2005 ff.), Graz Database on Reduplication, last accessed 10-26-2017 from http://reduplication.uni-graz.at/redup/.

Bernhard HURCH and Veronika MATTES (2009), Introduction: Diachrony and productivity of reduplication, Morphology, 19(2):107–112.

Larry M HYMAN (2009), The natural history of verb-stem reduplication in Bantu, Morphology, 19(2):177–206.

Larry M HYMAN, Sharon INKELAS, and Galen SIBANDA (2008), Morphosyntactic correspondence in Bantu reduplication, Current Studies in Linguistics, pp. 273–309, The MIT Press, Cambridge, MA.

William IDSARDI and Eric RAIMY (2008), Reduplicative Economy, in Bert VAUX and Andrew NEVINS, editors, Rules, constraints, and phonological phenomena, chapter 5, pp. 149–184, Oxford University Press.

Sharon INKELAS (2014), The interplay of morphology and phonology, Oxford Surveys in Syntax & Morphology, Oxford University Press, Oxford.

Sharon INKELAS and Laura J DOWNING (2015a), What is Reduplication? Typology and Analysis Part 1/2: The Typology of Reduplication, Language and Linguistics Compass, 9(12):502–515.

Sharon INKELAS and Laura J DOWNING (2015b), What is Reduplication? Typology and Analysis Part 2/2: The Analysis of Reduplication, Language and Linguistics Compass, 9(12):516–528.

Sharon INKELAS and Cheryl ZOLL (2005), Reduplication: Doubling in Morphology, Cambridge University Press, Cambridge.

Adam JARDINE (2016), Computationally, tone is different, Phonology, 33(2):247–283.

C Douglas JOHNSON (1972), Formal aspects of phonological description, Mouton, The Hague.

Ronald M KAPLAN and Martin KAY (1994), Regular models of phonological rule systems, Computational linguistics, 20(3):331–378.

Lauri KARTTUNEN (1983), KIMMO: a general morphological processor, in Texas Linguistic Forum, volume 22, pp. 163–186.

Paul KIPARSKY (1982), Lexical morphology and phonology, in I.-S. YANG, editor, Linguistics in the morning calm: Selected papers from SICOL-1981, pp. 3–91, Hansin, Seoul.

Paul KIPARSKY et al. (2010), Reduplication in stratal OT, Reality exploration and discovery: pattern interaction in language & life, pp. 125–142.

Jesse Saba KIRCHNER (2010), Minimal reduplication, University of California, Santa Cruz.

Jesse Saba KIRCHNER (2013), Minimal reduplication and reduplicative exponence, Morphology, 23(2):227–243.

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

Kimmo KOSKENNIEMI (1984), A general computational model for word-form recognition and production, in Proceedings of the 10th international conference on Computational Linguistics, pp. 178–181, Association for Computational Linguistics.

D Terence LANGENDOEN (1981), The generative capacity of word-formation components, Linguistic Inquiry, 12(2):320–322.

Jeffrey LIDZ (2001), Echo reduplication in Kannada and the theory of word-formation, Linguistic review, 18(4):375–394.

Huan LUO (2017), Long-Distance Consonant Agreement and Subsequentiality, Glossa: a journal of general linguistics, 2(1):1–25, doi:http://doi.org/10.5334/gjgl.42.

Alexis MANASTER-RAMER (1986), Copying in natural languages, context-freeness, and queue grammars, in Proceedings of the 24th annual meeting on Association for Computational Linguistics, pp. 85–89, Association for Computational Linguistics.

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

Gary F MARCUS, Sugumaran VIJAYAN, S Bandi RAO, and Peter M VISHTON (1999), Rule learning by seven-month-old infants, Science, 283(5398):77–80.

Veronika MATTES (2007), Reduplication in Bikol, Ph.D. thesis, University of Graz, Graz, Austria.

John J MCCARTHY, Wendell KIMPER, and Kevin MULLIN (2012), Reduplication in harmonic serialism, Morphology, 22(2):173–232.

John J MCCARTHY and Alan PRINCE (1995), Faithfulness and reduplicative identity, in Jill N. BECKMAN, Laura Walsh DICKEY, and Suzanne URBANCZYK, editors, Papers in Optimality Theory, Graduate Linguistic Student Association, University of Massachusetts, Amherst, MA.

John J MCCARTHY and Alan S PRINCE (1994), The emergence of the unmarked: Optimality in prosodic morphology, in Mercé GONZÀLEZ, editor, Proceedings of the North East Linguistic Society 24, p. 333–79, Graduate Linguistic Student Association, University of Massachusetts, Amherst, MA.

Fiona MCLAUGHLIN (2005), Reduplication and consonant mutation in the Northern Atlantic languages, in Hurch (2005), pp. 111–134.

Robert MCNAUGHTON and Seymour A PAPERT (1971), Counter-Free Automata, The MIT Press.

Mehryar MOHRI (1997), Finite-State Transducers in Language and Speech Processing, Computational Linguistics, 23(2):269–311.

Edith MORAVCSIK (1978), Reduplicative constructions, in Joseph GREENBERG, editor, Universals of Human Language, volume 1, pp. 297–334, Stanford University Press, Stanford, California.

Ajit NARAYANAN and Lama HASHEM (1993), On Abstract Finite-state Morphology, in Proceedings of the Sixth Conference on European Chapter of the Association for Computational Linguistics, EACL '93, pp. 297–304, Association for Computational Linguistics, Utrecht, The Netherlands, ISBN 90-5434-014-2, doi:10.3115/976744.976779, URL `https://doi.org/10.3115/976744.976779`.

Mark-Jan NEDERHOF and Heiko VOGLER (2019), Regular transductions with MCFG input syntax, in Proceedings of the 14th International Conference on

Finite-State Methods and Natural Language Processing, pp. 56–64, Association for Computational Linguistics, Dresden, Germany, URL https://www.aclweb.org/anthology/W19-3109.

Esa NELIMARKKA, Harri JÄPPINEN, and Aarno LEHTOLA (1984), Two-way finite automata and Dependency Grammar: A parsing method for inflectional free word order languages, in Proceedings of the 10th international conference on Computational linguistics, pp. 389–392, Association for Computational Linguistics.

Max NELSON, Hossep DOLATIAN, Jonathan RAWSKI, and Brandon PRICKETT (2020), Probing RNN Encoder-Decoder Generalization of Subregular Functions Using Reduplication, in Proceedings of the Society for Computation in Linguistics, volume 3.

Nicole Alice NELSON (2003), Asymmetric anchoring, Ph.D. thesis, Rutgers University, New Brunswick, NJ.

Andrew NEVINS (2004), What UG can and can't do to help the reduplication learner, in Aniko CSLRMAZ, Andrea GUALMINI, and Andrew NEVINS, editors, MIT Working Papers in Linguistics 48, pp. 113–126, MITWPL, Cambridge,MA.

Andrew NEVINS (2012), Haplological dissimilation at distinct stages of exponence, The morphology and phonology of exponence, pp. 84–116.

Andrew NEVINS and Bert VAUX (2003), Metalinguistic, shmetalinguistic: the phonology of shmreduplication, in Proceedings from the Annual Meeting of the Chicago Linguistic Society, volume 39, pp. 702–721, Chicago Linguistic Society.

David ODDEN (1994), Adjacency parameters in phonology, Language, pp. 289–330.

John J OHALA, Joseph Paul STEMBERGER, and Marshall LEWIS (1986), Reduplication in Ewe: morphological accommodation to phonological errors, Phonology, 3:151–160.

Amanda PAYNE (2014), Dissimilation as a Subsequential Process, in Jyoti IYER and Leland KUSMER, editors, NELS 44: Proceedings of the 44th Meeting of the North East Linguistic Society, volume 2, pp. 79–90, Graduate Linguistic Student Association, University of Massachusetts, Amherst, MA.

Amanda PAYNE (2017), All dissimilation is computationally subsequential, Language: Phonological Analysis, 93(4):e353–e371, doi:doi:10.1353/lan.2017.0076.

Brandon PRICKETT, Aaron TRAYLOR, and Joe PATER (2018), Seq2Seq Models with Dropout can Learn Generalizable Reduplication, in Proceedings of the Fifteenth Workshop on Computational Research in Phonetics, Phonology, and Morphology, pp. 93–100.

Michael O RABIN and Dana SCOTT (1959), Finite automata and their decision problems, IBM journal of research and development, 3(2):114–125.

Eric RAIMY (2000), The Phonology and Morphology of Reduplication, Berlin: Mouton de Gruyter.

Eric RAIMY (2009), Deriving Reduplicative Templates in a Modular Fashion, in Eric RAIMY and Charles E. CAIRNS, editors, Contemporary views on architecture and representations in phonology, number 48 in Current Studies in Linguistics, pp. 383–404, MIT Press, Cambridge, MA.

Eric RAIMY (2011), Reduplication, in Marc VAN OOSTENDORP, Colin EWEN, Elizabeth HUME, and Keren RICE, editors, The Blackwell companion to phonology, volume 4, pp. 2383–2413, Wiley-Blackwell, Malden, MA.

Charles REISS and Marc SIMPSON (2009), Reduplication as Projection.

Jason RIGGLE (2004), Nonlocal reduplication, in Kier MOULTON and Matthew WOLF, editors, Proceedings of the 34th meeting of the North Eastern Einguistics Society, Graduate Linguistic Student Association, University of Massachusetts, Amherst, MA.

Brian ROARK and Richard SPROAT (2007), Computational Approaches to Morphology and Syntax, Oxford University Press, Oxford.

James ROGERS and Geoffrey PULLUM (2011), Aural Pattern Recognition Experiments and The Subregular Hierarchy, Journal of Logic, Language and Information, 20:329–342.

Carl RUBINO (2005), Reduplication: Form, function and distribution, in Hurch (2005), pp. 11–29.

Carl RUBINO (2013), Reduplication, Max Planck Institute for Evolutionary Anthropology, Leipzig, URL `http://wals.info/chapter/27`.

Bridget SAMUELS (2010), The topology of infixation and reduplication, The Linguistic Review, 27(2):131–176.

Walter J SAVITCH (1982), Abstract machines and grammars, Little Brown and Company, Boston.

Walter J SAVITCH (1989), A formal model for context-free languages augmented with reduplication, Computational Linguistics, 15(4):250–261.

Paul SCHACHTER and Victoria FROMKIN (1968), A Phonology of Akan: Akuapem, Asante, Fante., UCLA Working Papers in Phonetics 9.

Marcel-Paul SCHÜTZENBERGER (1975), Sur certaines opérations de fermeture dans les langages rationnels, in Symposia Mathematica, volume 15, pp. 245–253.

Hiroyuki SEKI, Takashi MATSUMURA, Mamoru FUJII, and Tadao KASAMI (1991), On multiple context-free grammars, Theoretical Computer Science, 88(2):191–229.

Hiroyuki SEKI, Ryuichi NAKANISHI, Yuichi KAJI, Sachiko ANDO, and Tadao KASAMI (1993), Parallel multiple context-free grammars, finite-state translation systems, and polynomial-time recognizable subclasses of lexical-functional grammars, in Proceedings of the 31st annual meeting on Association for Computational Linguistics, pp. 130–139, Association for Computational Linguistics.

Jeffrey SHALLIT (2008), A Second Course in Formal Languages and Automata Theory, Cambridge University Press, New York, NY, USA, 1 edition, ISBN 0521865727, 9780521865722.

Patricia A SHAW (2005), Non-adjacency in reduplication, in Hurch (2005), pp. 161–210.

Daniel SILVERMAN (2002), Dynamic versus static phonotactic conditions in prosodic morphology, Linguistics, 40(1; ISSU 377):29–60.

Michael SIPSER (1980), Lower bounds on the size of sweeping automata, Journal of Computer and System Sciences, 21(2):195–202.

Philip SPAELTI (1997), Dimensions of variation in multi-pattern reduplication, Ph.D. thesis, University of California, Santa Cruz.

Richard William SPROAT (1992), Morphology and computation, MIT press, Cambridge:MA.

Donca STERIADE (1988), Reduplication and syllable transfer in Sanskrit and elsewhere, Phonology, 5(1):73–155.

Thomas STOLZ, Cornelia STROH, and Aina URDZE (2011), Total reduplication: The areal linguistics of a potential universal, volume 8, Walter de Gruyter, Berlin.

Kristina STROTHER-GARCIA (2018), Imdlawn Tashlhiyt Berber syllabification is quantifier-free, in Proceedings of the Society for Computation in Linguistics (SCiL) 2018, volume 1, pp. 145–153, doi:10.7275/R5J67F4D.

Kristina STROTHER-GARCIA (2019), Using model theory for grammatical inference: a case study from phonology, Ph.D. thesis, University of Delaware.

Gregory T STUMP (2001), Inflectional morphology: A theory of paradigm structure, volume 93, Cambridge University Press.

Terry TAI, Wojciech SKUT, and Richard SPROAT (2011), Thrax: An open source grammar compiler built on openfst, in IEEE Automatic Speech Recognition and Understanding Workshop, volume 12.

Suzanne URBANCYZK (2001), Patterns of reduplication in Lushootseed, Psychology Press.

Suzanne URBANCZYK (1999), Double reduplications in parallel, in René KAGER, Harry VN DER HULST, and Wim ZONNEVELD, editors, The prosody-morphology interface, pp. 390–428, Cambridge University Press, Cambridge.

Suzanne URBANCZYK (2007), Themes in phonology, in Paul DE LACY, editor, The Cambridge Handbook of Phonology, pp. 473–493.

Suzanne URBANCZYK (2011), Reduplication, URL http://oxfordindex.oup.com/view/10.1093/obo/9780199772810-0036.

Rachelle WAKSLER (1999), Cross-linguistic evidence for morphological representation in the mental lexicon, Brain and language, 68(1-2):68–74.

Markus WALTHER (2000), Finite-state reduplication in one-level prosodic morphology, in Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference, NAACL 2000, pp. 296–302, Association for Computational Linguistics, Seattle, Washington, URL http://dl.acm.org/citation.cfm?id=974305.974344.

Ronnie B WILBUR (2005), A reanalysis of reduplication in American Sign Language, in Hurch (2005), pp. 595–623.

Ronnie Bring WILBUR (1973), The phonology of reduplication, Ph.D. thesis, University of Indiana, Bloomington, Indiana.

Colin WILSON (2019), Re (current) reduplication: Interpretable neural network models of morphological copying, Proceedings of the Society for Computation in Linguistics, 2(1):379–380.

Moira YIP (1995), Repetition and its avoidance: The case of Javanese, in Keiichiro SUZUKI and Dirk ELZINGA, editors, Proceedings of the south western Optimality Theory workshop 1995. Arizona phonology conference Volume 5, pp. 238–262, University of Arizona, Tucson, AZ.

Alan CL YU (2007), A natural history of infixation, 15, Oxford University Press.

Kristine YU (2017), Advantages of constituency: computational perspectives on Samoan word prosody, in International Conference on Formal Grammar 2017, p. 105–124, Spring, Berlin.

Sam ZUKOFF (2017), Indo-European reduplication: Synchrony, diachrony, and theory, Ph.D. thesis, Massachusetts Institute of Technology.

Kie ZURAW, M Yu KRISTINE, and Robyn ORFITELLI (2014), The word-level prosody of Samoan, Phonology, 31(2):271–327.