Modeling reduplication with 2-way finite-state transducers

Hossep Dolatian, Jeffrey Heinz
Department of Linguistics
Institute for Advanced Computational Science
Stony Brook University
hossep.dolatian,jeffrey.heinz@stonybrook.edu

Abstract

This article describes a novel approach to the computational modeling of reduplication. Reduplication is a well-studied linguistic phenomenon. However, it is often treated as a stumbling block within finite-state treatments of morphology. Most finite-state implementations of computational morphology cannot adequately capture the productivity of unbounded copying in reduplication, nor can they adequately capture bounded copying. We show that an understudied type of finite-state machines, two-way finite-state transducers (2-way FSTs), captures virtually all reduplicative processes, including total reduplication. 2-way FSTs can model reduplicative typology in a way which is convenient, easy to design and debug in practice, and linguistically-motivated. By virtue of being finite-state, 2-way FSTs are likewise incorporable into existing finite-state systems and programs. A small but representative typology of reduplicative processes is described in this article, alongside their corresponding 2-way FST models.

1 Introduction

Reduplication is a cross-linguistically common word-formation process. Reduplication is roughly divided into two categories, total reduplication where an unbounded number of segments are copied (1) vs. partial reduplication where a bounded number of segments are copied (2). In spoken language, reduplication usually involves making at most two copies, though making three copies is attested in spoken language (3) and is common in sign language (Wilbur, 2005).

1) wanita→wanita∼wanita
'woman'→'women'
Indonesian (Cohn, 1989, 308)

2) takki→tak∼takki
'leg'→'legs' Agta (Moravcsik, 1978, 311)

3) roar→ roar∼roar∼roar
'give a shudder' →'continue to shudder'
Mokilese (Moravcsik, 1978, 301)

Most of the world's languages include at least one reduplicative process, with the most common reduplicative process being total reduplication. The WALS database documents that 278 out of 368 (75%) languages use both partial reduplication and total reduplication as productive morphological operations (Rubino, 2013). An extra 35 (10%) use only total reduplication as a productive morphological operation. The 55 (15%) remaining languages with no reduplicative processes include most Indo-European languages.[1]

Although reduplication has a rich history in morpho-phonology, it continues to present challenges for computational and mathematical linguistics (Sproat, 1992; Roark and Sproat, 2007). Within computational linguistics, most of morphology and phonology have been analyzed with finite-state calculus as rational languages and transductions (Kaplan and Kay, 1994; Beesley and Karttunen, 2003). However, reduplication cannot be easily modeled with the same finite-state systems used to model the rest of morpho-phonology. In the case of 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 discussed as being burdensome models because of the state explosion that partial reduplication causes (Roark and Sproat, 2007). This has lead some researchers to develop finite-state *approximations* of total reduplication (Walther, 2000; Beesley and Karttunen, 2000; Cohen-Sygal and Wintner, 2006; Hulden,

[1]This 15% is debatable because some argue that total reduplication is still used in those languages in one way or another (Stolz et al., 2011).

Proceedings of the 15th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology, pages 66–77

https://doi.org/10.18653/v1/P17

2009a; Hulden and Bischoff, 2009). These are approximations because they cannot model the productivity of total reduplication, the most common reduplicative process. Another alternative is to use formalisms that are beyond finite-state, e.g. queue-based CFGs (Savitch, 1989), MCFGs (Albro, 2000, 2005), and HPSG (Crysmann, 2017).

This article shows how a specific understudied type of finite-state technology actually can account for virtually all forms of bounded and unbounded reduplication as they are found in typological studies (Moravcsik, 1978; Rubino, 2005). This finite-state technology not only describes reduplication as a process which applies to infinitely many words of unbounded size, but it does so without the state-space explosion. The type of transducer which accomplishes this is known as a 2-way Finite-State Transducer or 2-way FST (Savitch, 1982; Engelfriet and Hoogeboom, 2001; Filiot and Reynier, 2016). While these computer scientists are well aware that 2-way FSTS can model unbounded copying, this is the first use of 2-way FSTs within computational linguisticst to our knowledge.[2]

2-way FSTs are distinguished from the more well-known (1-way) finite-state transducers or 1-way FSTs by allowing the machine to move back and forth on the input tape, but not on the output tape. It is this increased power of 2-way FSTs that allows them to adequately model reduplication without the difficulties of using 1-way FSTs.

In this paper, we focus on *deterministic* 2-way FSTs. Like 1-way FSTs, 2-way FSTs can be either deterministic or non-deterministic on the input. 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, deterministic 2-way FSTs are less expressive than non-deterministic 2-way FSTs (Culik and Karhumäki, 1986). For the typology of reduplication studied in this article, *deterministic* 2-way FSTs are sufficient. This result is in line with work showing 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).

[2] 2-way finite-state automata (2-way FSAs) have been used to model non-concatenative Semitic morphology (Narayanan and Hashem, 1993).

This article is organized as follows. 2-way finite-state transducers (2-way FSTs) are introduced in section §2, where we provide a formal definition (§2.1), discuss their computational properties (§2.2), and discuss their computational complexity (§2.3). In §3, we illustrate how 2-way FSTs can model reduplication, notably total reduplication (§3.1) and partial reduplication (§3.2). In section §4, we contrast 2-way FSTs with 1-way FSTs and show how the former are empirically adequate, practically convenient or useful, and linguistically-motivated for modeling reduplication. To illustrate this, we briefly discuss how we have used 2-way FSTs to develop the RedTyp database, a database of reduplicative processes with corresponding 2-way FSTs. Conclusions and directions for future research are in §5.

2 Two-way finite-state transducers: definition and properties

2.1 Definition

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, the read head. The FST begins at the first position of the tape reading σ_1, and the writing head of the FST is positioned 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 changes.
- The FST writes some string, possibly empty, to the output tape.
- The read head may move in one of three ways: it can either move to the left (-1), move to the right (+1), or stay (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. Note the writing head of the FST

can never move back along the output tape. It only ever advances as strings are written.

Below is a formalization of 2-way FSTs based on Filiot and Reynier (2016) and Shallit (2008). We adopt the convention that inputs to a 2-way FST are flanked with the start ($\rtimes$) and end boundaries ($\ltimes$). This larger alphabet is denoted by $\Sigma_{\ltimes}$.

4) **Definition:** A 2-way, deterministic FST is a six-tuple $(Q, \Sigma_{\ltimes}, \Gamma, q_0, F, \delta)$ such that:

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,
$\delta : Q \times \Sigma \rightarrow Q \times \Gamma^* \times D$ is the transition function where the direction $D = \{-1, 0, +1\}$.

A *configuration* of a 2-way 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 $(wqax, u) \rightarrow (wrax, uv)$. 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 $(wqax, u) \rightarrow (warx, uv)$. If the current configuration is $(waqx, u)$ and $\delta(q, a) = (r, v, -1)$ then the next configuration is $(wrax, uv)$. We write $(waqx, u) \rightarrow (wrax, uv)$.

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

Next we define the function that a 2-way FST $T = (Q, \Sigma_{\ltimes}, \Gamma, q_0, F, \delta)$ computes. For each string $w \in \Sigma^*$, $f_T(w) = u \in \Gamma^*$ provided there exists $q_f \in F$ such that $(q_0 \rtimes w \ltimes, \lambda) \rightarrow^+ (\rtimes w \ltimes q_f, u)$. Note that since a 2-way FST is deterministic, it follows that if $f_T(w)$ is defined then u is unique.

There are situations where a 2-way FST T crashes on some input w and hence $f_T(w)$ is undefined. 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.

There is one more way in which f_T may be undefined for some input. The input may cause the transducer to go into an infinite loop.[3] This occurs for input $wx \in \Sigma^*_{\ltimes}$ whenever there exist $q \in Q$ and $u, v \in \Gamma^*$ such that $(q_0 wx, \lambda) \rightarrow^+ (wqx, u) \rightarrow^+ (wqx, uv)$.

2.2 Computational properties

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 FSTs (Savitch, 1982; Aho et al., 1969). 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, as we will see, the output language of a 2-way FST can be a non-regular language such as L_{ww}. The next section will show that this additional power allows 2-way FSTs to productively model reduplication.

2-way FSTs are equivalent in expressivity to MSO-definable string transductions (Engelfriet and Hoogeboom, 2001) and to streaming string transducers (1-way FSTs with registers) (Alur, 2010). They are closed under composition (Chytil and Jákl, 1977) and their non-deterministic variants are invertible (Courcelle and Engelfriet, 2012). 2-way FSTs are less powerful than Turing machines because they cannot move back and forth over the output tape.

Note that given the difference in expressive power between 1-way and 2-way 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 a 1-way deterministic FST as 'rational functions', and they reserve the term 'regular functions' for functions describable with 2-way deterministic FSTs.

[3] Infinite loops can be prevented through carefully designing the 2-way FST. The 2-way FSTs which we have made do not suffer from infinite loops. Infinite loops can likewise be checked and stopped during run-time.

2.3 Computational complexity

Deterministic 1-way FSTs run in time linear to the length of the input string. Since 2-way FSTs can reread the input string, is this still the case? One useful metric for measuring the complexity of deterministic 2-way FSTs is in terms of the number of times the 2-way FST passes through the input (Baschenis et al., 2016). In the case of the reduplication examples in §3, a deterministic 2-way FST can be designed with only two passes through the input per copy. Thus, the run time for a deterministic 2-way FST modeling reduplication which makes at most n copies of an input string of length m is $2n \cdot m$. Since n is fixed by the reduplicative morpheme, the run time is still linear in the size of the input string.

Also, to our knowledge existing applications of regular functions have been efficient (Alur and Černý, 2011; Alur et al., 2014).

3 Illustrative use of two-way transducers for reduplication

Having established what 2-way FSTs are and how they behave, this section illustrates how they can be used model reduplication. We provide two illustrative examples: total reduplication (§3.1) and partial reduplication (§3.2).

3.1 Total reduplication

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). Examples are in Table 1.

input	gloss	output	gloss
buku	'book'	buku∼buku	'books'
wanita	'woman'	wanita∼wanita	'women'
hak	'right'	hak∼hak	'right'
kəra	'donkey'	kəra∼kəra	'donkeys'

Table 1: Total reduplication in Indonesian.

Figure 1 shows a 2-way FST that captures this total reduplication process. Basically, the 2-way FST in Figure 1 operates by:

1. reading the input tape once from left to right in order to output the first copy,
2. going back to the start of the input tape by moving left until the start boundary $\rtimes$ is reached,
3. reading the input tape once more from left to right in order to output the second copy.

Specifically, this figure is interpreted as follows. The symbol Σ stands for any segment in the alphabet except for $\{\rtimes, \ltimes\}$. The arrow from q_1 to itself means this 2-way FST reads Σ, writes Σ, and advances the read head one step to the right on the input tape. The boundary symbol $\sim$ is a symbol in the output alphabet Γ, and is not necessary. We include it only for illustration.

We show an example derivation in Figure 2 of /buku/→[buku∼buku] using the 2-way FST in Figure 1. The derivation shows the configurations of the computation for the input /buku/ and is step by step. Each tuple consists of four parts: *input string, output string, current state, transition*. In the *input string*, we underline the input symbol which FST will read next. The *output string* is what the 2-way FST has outputted up to that point. The symbol λ marks the empty string. The *current state* is what state the FST is currently in. The *transition* represents the used transition arc from input to output. In the first tuple, there is no transition arc used (N/A). But for other tuples, the form of the arc is:

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

3.2 Partial reduplication

Partial reduplication processes are also very common. A canonical example is initial-CV reduplication found in many Austronesian languages (Rubino, 2005). This section presents a simplified version of initial-CV reduplication from Bikol that is used to mark imperfective aspect (Mattes, 2007).[4] Examples are in Table 2.

input	gloss	output	gloss
ŋirit	'to laugh'	ŋi∼ŋirit	'laughing'
diretsjo	'to continue'	di∼diretsjo	'continuing'
trabaho	'to work'	ta∼trabaho	'working'
draɪf	'to drive'	da∼draɪf	'driving'

Table 2: Initial-CV reduplication in Bikol.

Initial-CV reduplication in Bikol has two phonological modifications processes[5] apply to the reduplicant, i.e. the smaller copy:

[4] Initial-CV reduplication in Bikol targets the root and is triggered by the addition of certain prefixes. For illustrative purposes, we set aside these prefixes.

[5] These modifications effects are often called TETU (or *the emergence of the unmarked*) effects in the linguistics literature (McCarthy and Prince, 1994, 1995).

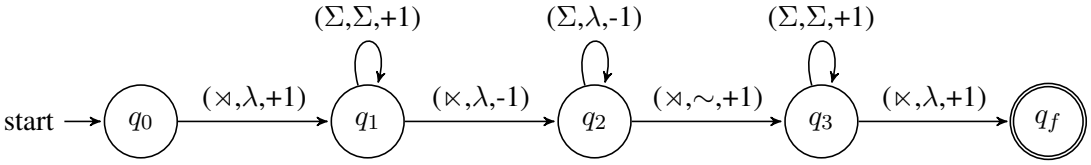


Figure 1: 2-way FST for total reduplication in Indonesian.

Outputting the first copy							
1. (⋊buku⋉,	λ,	q_0 ,	N/A)	7. (⋊buku⋉,	buku∼,	q_2,	$q_1 \xrightarrow[-1]{⋉:∼} q_2$)
2. (⋊buku⋉,	λ,	q_1,	$q_0 \xrightarrow[+1]{⋊:λ} q_1$)	8. (⋊buku⋉,	buku∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
3. (⋊buku⋉,	b,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	9. (⋊buku⋉,	buku∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
4. (⋊buku⋉,	bu,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	10. (⋊buku⋉,	buku∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
5. (⋊buku⋉,	buk,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)	11. (⋊buku⋉,	buku∼,	q_2,	$q_2 \xrightarrow[-1]{Σ:λ} q_2$)
6. (⋊buku⋉,	buku,	q_1,	$q_1 \xrightarrow[+1]{Σ:Σ} q_1$)				
Outputting the second copy							
12. (⋊buku⋉,	buku∼,	q_3,	$q_2 \xrightarrow[+1]{⋊:λ} q_3$)	15. (⋊buku⋉,	buku∼buk,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)
13. (⋊buku⋉,	buku∼b,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)	16. (⋊buku⋉,	buku∼buku,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)
14. (⋊buku⋉,	buku∼bu,	q_3,	$q_3 \xrightarrow[+1]{Σ:Σ} q_3$)	17. (⋊buku⋉,	buku∼buku,	q_f,	$q_3 \xrightarrow[+1]{⋉:⋉} q_f$)

Figure 2: Derivation of /buku/→[buku∼buku].

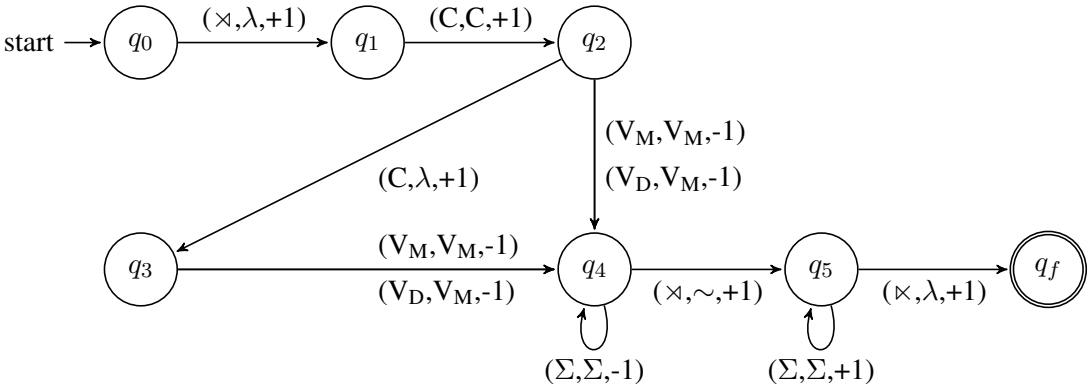


Figure 3: 2-way FST for initial-CV reduplication in Bikol.

- complex onsets are reduced to simple onsets, e.g. /trabaho/→[ta∼trabaho] 'working'
- diphthongs are reduced to monophthongs, e.g. /draɪf/→[da∼draɪf] 'driving'

The 2-way FST in Figure 3 captures the partial reduplication pattern and its modifications. The symbol V_M stands for monophthongs, V_D for diphthongs, and C for consonants. An example derivation of /draɪf/→[da∼draɪf] using our 2-way FST is provided in Figure 4.[6]

4 Contrasting 2-way FSTs with 1-way FSTs

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

We do not contrast 2-way FSTs with more powerful formalisms like pushdown transducers (Allauzen and Riley, 2012). We do not assume the former are superior to other such formalisms. Our goal is to show 2-way FSTs have practical and scientific utility in computational linguistics; thus, they merit further study.

4.1 Empirical coverage

In terms of empirical coverage, 2-way FSTs can model *virtually* the entire typology of reduplication (Moravcsik, 1978; Hurch, 2005; Inkelas and Zoll, 2005; Rubino, 2005; Samuels, 2010). This includes both local reduplication (as in the two examples from §3), but likewise non-local or 'wrong-side' reduplication (Riggle, 2004), internal reduplication (Broselow and McCarthy, 1983), multiple reduplication (Urbanczyk, 1999), subconstituent reduplication (Downing, 1998), and cases of interactions between reduplication and opaque phonological processes (overapplication, underapplication, backcopying) (McCarthy and Prince, 1995). This is especially the case for total reduplication which is the most widespread reduplicative process (Rubino, 2013) but which cannot be modeled with 1-way FSTs. In most cases, this will be inadequate because total reduplication is a productive grammatical process (Rubino, 2005, 2013).

We emphasize the term *virtually* because in our investigation we have found only two marginal cases of reduplication in the literature which cannot be modeled by 2-way FSTs unless certain plausible assumptions are made. These two cases involve reduplication producing suppletive allomorphs of morphemes as in Sye (Inkelas and Zoll, 2005, 52), and reduplication being blocked by homophony or haplology as in Kanuri (Moravcsik, 1978, 313). These two cases of 'under-generation' can be solved if we assume the language contains a finite number of suppletive allomorphs, and if we assume that there's either a finite number of banned identical sequences or a separate linguistic mechanism that filters out ill-formed homophonies.

Of course there are cases where 2-way FSTs can 'over-generate' and model unattested types of reduplication, e.g. reduplicate a word *n* times for some natural number *n* or reduplicate a word by reversing it. This over-generation can be addressed by either restricting the class of 2-way FSTs used (Dolatian and Heinz, 2018) or by not treating 2-way FSTs as having to be exact models of human cognition (Potts and Pullum, 2002). For further discussion and solutions on how 2-way FSTs can over- and under-generate, see Dolatian and Heinz (In press.).

4.2 Practical utility

To showcase empirical coverage of 2-way FSTs and their practical utility, we have constructed the RedTyp database[7] which contains entries for 138 reduplicative processes from 91 languages gleaned from various surveys (Rubino, 2005; Inkelas and Downing, 2015). 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 FSTs that model the 138 processes.[8] Each 2-way FST was designed manually, implemented in Python, and checked for correctness. On average, these 2-way FSTs had 8.8 states. This shows that 2-way 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 redupli-

[6] The FST treats the diphthong /aɪ/ as a single segment.

[7] A copy of RedTyp can be found online at our GitHub page https://github.com/jhdeov/RedTyp.

[8] 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.

Outputting reduplicant							
1. (⋊draɪf⋉,	λ,	q_0 ,	N/A)	4. (⋊draɪf⋉,	d,	q_3,	$q_2 \xrightarrow[+1]{\text{C:C}} q_3$)
2. (⋊draɪf⋉,	λ ,	q_1,	$q_0 \xrightarrow[+1]{\rtimes:\lambda} q_1$)	5. (⋊draɪf⋉,	da,	q_4,	$q_3 \xrightarrow[-1]{\text{V}_\text{D}:\text{V}_\text{M}} q_4$)
3. (⋊draɪf⋉,	d,	q_2,	$q_1 \xrightarrow[+1]{\text{C:C}} q_2$)				
Going back to the start of the tape							
6. (⋊draɪf⋉,	da,	q_4,	$q_4 \xrightarrow[-1]{\Sigma:\lambda} q_4$)	7. (⋊draɪf⋉,	da,	q_4,	$q_4 \xrightarrow[-1]{\Sigma:\lambda} q_4$)
Outputting the base							
8. (⋊draɪf⋉,	da∼,	q_5,	$q_4 \xrightarrow[+1]{\rtimes:\sim} q_5$)	11. (⋊draɪf⋉,	da∼draɪ,	q_5,	$q_5 \xrightarrow[+1]{\Sigma:\Sigma} q_5$)
9. (⋊draɪf⋉,	da∼d,	q_5,	$q_5 \xrightarrow[+1]{\Sigma:\Sigma} q_5$)	12. (⋊draɪf⋉,	da∼draɪf,	q_5,	$q_5 \xrightarrow[+1]{\Sigma:\Sigma} q_5$)
10. (⋊draɪf⋉,	da∼dr,	q_5,	$q_5 \xrightarrow[+1]{\Sigma:\Sigma} q_5$)	13. (⋊draɪf⋉,	da∼draɪf,	q_f,	$q_5 \xrightarrow[+1]{\ltimes:\lambda} q_5$)

Figure 4: Derivation of /draɪf/→[da∼draɪf].

cation.[9] On average, a language's phoneme inventory would include 22 consonants and 5 vowels (Maddieson, 2013a,b). In order to handle initial-CV, initial-CVC, or initial-CVCV reduplication with a 1-way FST, the FST would require at least an estimated 22, 110, and 2420 states respectively.

4.3 Linguistic motivation and origin semantics

Finally, using 2-way FSTs for reduplication is linguistically motivated and matches the intensional descriptions behind the linguistic generalizations on reduplication. 2-way FSTs do not approximate reduplication like 1-way FSTs do. They can fully and productively model reduplicative processes as they appear in the typology, including both partial and total reduplication. As said, this is because 1-way FSTs simply *remember* the possible shapes for a reduplicant when the number of possible shapes is (large yet) finite as in partial reduplication. When the number of possible shapes to remember is unbounded as in total reduplication, a 1-way FST cannot productively model reduplication. In contrast, a 2-way FST does not need to remember strings of segments in order to copy them, but *actively copies* them.

This contrast between copying and remembering can be formalized with the notion of the *origin semantics* of a transduction (Bojańczyk, 2014).

[9]The largest 2-way 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 FST would require over 1,000 states for this pattern of partial reduplication.

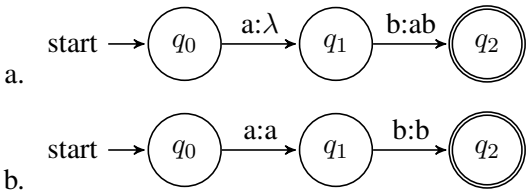


Figure 5: Pair of 1-way FSTs for the function f_{ab}.

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 string-to-string function f_{ab} which maps *ab* to itself, and every other string to the empty string: $f(x) = \{(ab, ab), (a, \lambda), (b, \lambda), ...\}$. This function can be modeled with at least two different 1-way FSTs as in Figure 5 which differ in when they output the output symbols *a* and *b*. In Figure 6, we show the origin information created by the two 1-way FSTs from Figure 5 for the mapping (*ab*, *ab*). The two 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 create differ origin information for their output.

This notion of origin semantics can be used to contrast how 1-way FSTs and 2-way FSTs model reduplication. Consider the case of Bikol initial-CV reduplication from section §3.2 and assume a smaller alphabet $\Sigma = \{\text{p,a,t}\}$. This function can

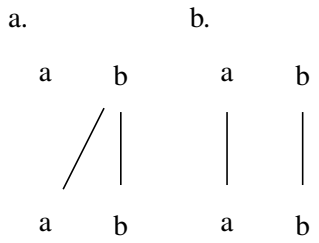


Figure 6: Origin information created by the 1-way FSTs (5) for the mapping *ab*→*ab*.

be modeled by the same 2-way FST in Figure 3. Because of the bound on the size of the reduplicant, this function can also be modeled with the 1-way FST in Figure 7.

The two transducers in Figures 3,7 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. Given the mapping /pat/→[pa∼pat], the two FSTs will create different origin information. Setting aside the word boundaries and reduplicant boundary ∼, the 1-way FST associates the second *pa* string of the output with the vowel *a* of the input as in Figure 8a. 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 Figure 8b.

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). In contrast, the origin information created by the 1-way FST does not match any 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 difference in behavior: the 1-way FST simply remembers what strings of segments to output twice, while the 2-way FST actively copies.

In Base-Reduplicant correspondence theory (BRCT), what matters for reduplication is not the relationship or correspondence between input and output segments in the reduplication, but between the two copies in the output (McCarthy and Prince, 1995). Origin semantics might be able to formalize the intuition behind BRCT with finite-state technology (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).

5 Conclusion

In summary, finite-state technology has often been argued to be incapable of adequately and efficiently capturing productive reduplication as used in natural language. However, this article shows that an understudied type of finite-state machinery—2-way finite-state transducers—can exactly model reduplication and its wide typology.

2-way FSTs can model the virtually entire typology of reduplication, without needing to approximate any processes (unlike 1-way FSTs). They likewise do not suffer from a state explosion for partial reduplication because the size of the 2-way FST is not dependent on the size of the alphabet. This allows 2-way FSTs to directly capture the copying aspect of reduplication instead of remembering all potential reduplicants. This makes 2-way FSTs be a practical, convenient, and concise tool to model reduplication. As a sign of their empirical coverage and utility, we developed the RedTyp database of reduplicative processes that contains 57 distinct 2-way FSTs which model common and uncommon reduplicative processes covered in the literature (Moravcsik, 1978).

Having showcased their utility, several avenues of future research remain, of which we highlight three. First, we have approached reduplication from the perspective of morphological generation. Given an input /buku/, a 2-way 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].[10]

A second, more practical, 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), `foma` (Hulden, 2009b), `open-fst` (Allauzen et al., 2007), and `pynini` (Gorman, 2016).

[10] One potential route may be the use of non-deterministic 2-way FSTs (Alur and Deshmukh, 2011).

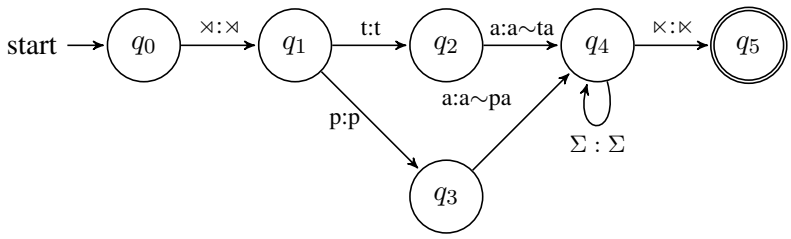
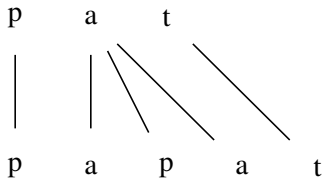


Figure 7: 1-way FST for partial reduplication.

a. Origin information of the 1-way FST



b. Origin information of the 2-way FST

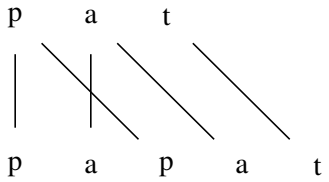


Figure 8: Origin information of /pat/→[pa∼pat] created by the 1-way FST (Figure 7) vs. the 2-way FST (Figure 3).

Third, it is theoretically interesting that within morpho-phonology, only reduplication requires the bidirectional power of 2-way FSTs. The bulk of morphology and phonology can be modeled with non-deterministic 1-way finite-state transducers (Beesley and Karttunen, 2003; Jardine, 2016) or subclasses of them (Chandlee, 2017). As a copying process, reduplication requires more than just 1-way finite-state technology. This may be a sign that it is of a different nature than the rest of morpho-phonology (Inkelas and Zoll, 2005; Urbanczyk, 2017). It is an open question if 2-way FSTs can likewise be used to model copying in other areas of natural language, including syntactic copying (Kobele, 2006).

Fourth, in the same way that Chandlee 2014; 2017 and Chandlee et al. (2014, 2015) have studied subclasses of 1-way FSTs and shown how they map to subclasses of morpho-phonology, we are currently investigating what proper subclasses of 2-way FSTs can be designed in order to make a tighter fit with reduplicative typology. This would open doors to not only better understanding the computational properties of reduplication, but to likewise develop learning algorithms for reduplication. As of now, we hypothesize that a large majority of reduplicative fall under a sub-class of 2-way FSTs (that we have discovered) based on a 2-way extension of the Output-Strictly Local subclass of 1-way FSTs (Chandlee et al., 2015). For more discussion of this subclass for reduplication and its learnability, see Dolatian and Heinz (2018).

In sum, the present study is the initial step in formalizing the wide typology of reduplicative processes into mathematically sound, yet expressively adequate, formal-language theoretic terms. Future work will include incorporating this technology into existing platforms and NLP systems, and further bridging the gaps between computational and theoretical morpho-phonology.

Acknowledgments

This research is supported by NIH grant #R01-HD087133 to JH. We thank the reviewers and audiences at CLS53, NAPhCX, the University of Delaware, and Stony Brook University.

References

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 *Finite-state phonology: proceedings of the 5th Workshop of SIGPHON*, pages 57–67.

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.

Cyril Allauzen and Michael Riley. 2012. A pushdown transducer extension for the OpenFst library. In *Implementation and Application of Automata*, pages 66–77, Berlin, Heidelberg. Springer.

Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst: A general and efficient weighted finite-state transducer library. In *Implementation and Application of Automata*, pages 11–23, Berlin, Heidelberg. Springer.

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, page 112.

Rajeev Alur and Jyotirmoy V. Deshmukh. 2011. Nondeterministic streaming string transducers. In *Automata, Languages and Programming*, pages 1–20, Berlin, Heidelberg. Springer.

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, pages 9:1–9:10, New York, NY, USA. ACM.

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, pages 599–610, New York, NY, USA. ACM.

Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. 2016. Minimizing Resources of Sweeping and Streaming String Transducers. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 114:1–114:14, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

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, pages 191–198, Stroudsburg, PA, USA. Association for Computational Linguistics.

Kenneth R. Beesley and Lauri Karttunen. 2003. *Finite-state morphology: Xerox tools and techniques*. CSLI Publications.

Mikołaj Bojańczyk. 2014. Transducers with origin information. In *Automata, Languages, and Programming*, pages 26–37, Berlin, Heidelberg. Springer.

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

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

Jane Chandlee. 2017. Computational locality in morphological maps. *Morphology*, pages 1–43.

Jane Chandlee, Angeliki Athanasopoulou, and Jeffrey Heinz. 2012. Evidence for classifying metathesis patterns as subsequential. In *The Proceedings of the 29th West Coast Conference on Formal Linguistics*, pages 303–309, Somerville, MA. Cascillida Press.

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Learning strictly local subsequential functions. *Transactions of the Association for Computational Linguistics*, 2:491–503.

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)*, pages 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, pages 42–51, Montreal, Canada. Association for Computational Linguistics.

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.

Michal P. Chytil and Vojtěch Jákl. 1977. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Automata, Languages and Programming*, pages 135–147, Berlin, Heidelberg. Springer.

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.

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.

Karel Culik and Juhani Karhumäki. 1986. The equivalence of finite valued transducers (on HDT0L languages) is decidable. *Theoretical Computer Science*, 47:71 – 84.

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

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

Hossep Dolatian and Jeffrey Heinz. In press. Reduplication with finite-state technology. In *Proceedings of the 53rd Annual Meeting of the Chicago Linguistics Society*.

Laura J Downing. 1998. Prosodic misalignment and reduplication. In Geert Booij and Jaap van Marle, editors, *Yearbook of Morphology 1997*, pages 83–120. Kluwer Academic Publishers, Dordrecht.

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

C. C. Elgot and J. 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.

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

Brian Gainor, Regine Lai, and Jeffrey Heinz. 2012. Computational characterizations of vowel harmony patterns and pathologies. In *The Proceedings of the 29th West Coast Conference on Formal Linguistics*, pages 63–71, Somerville, MA. Cascillida Press.

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*, pages 75–80. Association for Computational Linguistics.

Jeffrey Heinz and Regine Lai. 2013. Vowel harmony and subsequentiality. In *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 52–63, Sofia, Bulgaria. Association for Computational Linguistics.

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. 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*, pages 29–32. Association for Computational Linguistics.

Mans Hulden and Shannon T Bischoff. 2009. A simple formalism for capturing reduplication in finite-state morphology. In *Proceedings of the 2009 conference on Finite-State Methods and Natural Language Processing: Post-proceedings of the 7th International Workshop FSMNLP 2008*, pages 207–214, Amsterdam. IOS Press.

Bernhard Hurch, editor. 2005. *Studies on reduplication*. 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/.

Sharon Inkelas and Laura J Downing. 2015. What is reduplication? Typology and analysis part 1/2: The typology of reduplication. *Language and Linguistics Compass*, 9(12):502–515.

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.

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

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

Huan Luo. 2017. Long-distance consonant agreement and subsequentiality. *Glossa: a journal of general linguistics*, 2(1):125.

Ian Maddieson. 2013a. *Consonant Inventories*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

Ian Maddieson. 2013b. *Vowel Quality Inventories*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

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

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 *Proceedings of the North East Linguistic Society 24*, page 33379, Amherst, MA. Graduate Linguistic Student Association, University of Massachusetts.

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, pages 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, pages 297–304, Stroudsburg, PA, USA. Association for Computational Linguistics.

Amanda Payne. 2014. Dissimilation as a subsequential process. In *NELS 44: Proceedings of the 44th Meeting of the North East Linguistic Society*, volume 2, pages 79–90, Amherst, MA. Graduate Linguistic Student Association, University of Massachusetts.

Amanda Payne. 2017. All dissimilation is computationally subsequential. *Language: Phonological Analysis*, 93(4):e353–e371.

Christopher Potts and Geoffrey K Pullum. 2002. Model theory and the content of OT constraints. *Phonology*, 19(3):361–393.

Jason Riggle. 2004. Nonlocal reduplication. In *Proceedings of the 34th meeting of the North Eastern Einguistics Society*. Graduate Linguistic Student Association, University of Massachusetts.

Brian Roark and Richard Sproat. 2007. *Computational Approaches to Morphology and Syntax*. Oxford University Press, Oxford.

Carl Rubino. 2005. Reduplication: Form, function and distribution. In *Studies on reduplication*, pages 11–29. Mouton de Gruyter, Berlin.

Carl Rubino. 2013. *Reduplication*. Max Planck Institute for Evolutionary Anthropology, Leipzig.

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.

Marcel-Paul Schützenberger. 1975. Sur certaines opérations de fermeture dans les langages rationnels. In *Symposia Mathematica*, volume 15, pages 245–253.

Jeffrey Shallit. 2008. *A Second Course in Formal Languages and Automata Theory*, 1 edition. Cambridge University Press, New York, NY, USA.

Richard William Sproat. 1992. *Morphology and computation*. MIT press, Cambridge:MA.

Thomas Stolz, Cornelia Stroh, and Aina Urdze. 2011. *Total reduplication: The areal linguistics of a potential universal*, volume 8. Walter de Gruyter, Berlin.

Suzanne Urbanczyk. 1999. Double reduplications in parallel. In René Kager, Harry vn der Hulst, and Wim Zonneveld, editors, *The prosody-morphology interface*, pages 390–428. Cambridge University Press, Cambridge.

Suzanne Urbanczyk. 2017. Phonological and morphological aspects of reduplication.

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, pages 296–302, Stroudsburg, PA. Association for Computational Linguistics.

Ronnie B Wilbur. 2005. A reanalysis of reduplication in american sign language. In *Studies on reduplication*, pages 595–623. Berlin: Mouton de Gruyter.