Learning reduplication with 2-way finite-state transducers

Hossep Dolatian HOSSEP.DOLATIAN@STONYBROOK.EDU
Jeffrey Heinz JEFFREY.HEINZ@STONYBROOK.EDU
Department of Linguistics
Institute for Advanced Computational Science
Stony Brook University
Stony Brook, NY 11794, USA

Editor: Editor's name

Abstract

Reduplication is a cross-linguistically common and productive word-formation mechanism. However, there are little to no learning results concerning it. This is partly due to the high computational complexity associated with copying, which often goes beyond standard finite-state technology and partly due to the absence of concrete computational models of reduplicative processes. We show here that reduplication can be modeled accurately and succinctly with 2-way finite-state transducers. Based on this finite-state representation, we identify a subclass of 2-way FSTs based on copying and Output Strictly Local functions. These so-called Concatenated Output Strictly Local functions (C-OSL) can model the majority of attested reduplicative processes we have surveyed. We introduce a simple extension to the inference algorithm for OSL functions that trivially leads to a provably correct learning result for C-OSL functions under the assumption that function concatenation is overtly marked.

Keywords: reduplication, grammatical inference, copying, Output Strictly Local functions, two-way finite-state transducer

1. Introduction

Reduplication is a cross-linguistically common word-formation mechanism (Moravcsik, 1978; Rubino, 2005; Inkelas and Downing, 2015). It is divided into two types: total reduplication, where an unbounded number of segments are copied (1a), and partial reduplication, where a bounded number are copied (1b).[1] Total reduplication occurs in an estimated 83% of the world's languages, while partial reduplication occurs in an estimated 75% (Rubino).

(1)	a. wanita→wanita∼wanita	'woman'→'women'	Indonesian (Cohn, 1989, 308)
	b. takki→tak∼takki	'leg'→'legs'	Agta (Moravcsik, 1978, 311)

Within linguistics, reduplication is well-studied. However, there is little to no work on learning reduplication, whether in machine learning or grammatical inference. To our knowledge, the only algorithm designed specifically for learning reduplication is Nevins (2004) who uses principles-and-parameters. Outside of linguistics, one method for introducing copying into neural networks was introduced by Gu et al. (2016), though they did

1. In all examples, the ∼ symbol indicates the juncture between the copy and the original form.

not study reduplication specifically. The absence of learning research on reduplication is possibly due to how reduplication is often modeled with complex computational machinery, e.g. MCFGs (Albro, 2005) or HPSG (Crysmann, 2017), that are more complicated than the finite-state technology used to model the rest of morphology (Beesley and Karttunen, 2003) and phonology (Kaplan and Kay, 1994).

The state-of-the-art in finite-state technology as currently practiced cannot adequately and elegantly describe many cases of reduplication, especially unbounded total reduplication (Culy, 1985; Hulden, 2009; Roark and Sproat, 2007; Chandlee, 2014). As for partial reduplication, mainstream finite-state techniques (1-way finite-state transducers) require memorizing all possible types of reduplicants, which leads to an explosion in the number of states needed to model partial reduplication (Hulden, 2009; Roark and Sproat, 2007; Chandlee and Heinz, 2012). Consequently, it is difficult and unwieldy to design these machines, use these machines, and to model partial reduplication in a way that is connected to how it is understood within the theoretical linguistic tradition.

The present work aims to fill this gap by a) providing a useful and intuitive computational representation for reduplication, b) identifying a subclass of 2-way FSTs which covers the majority of the linguistic typology on reduplication, and c) providing a learning algorithm for reduplication based on that computational representation and subclass.

For (a), we use two-way finite-state transducers (2-way FSTs) (Engelfriet and Hoogeboom, 2001). They are an understudied type of finite-state technology which *can* accurately model virtually *all* of the reduplicative patterns found in typological studies on reduplication. These differ from the conventionally used (1-way) FSTs by being able to move the reading head on the input tape either left-to-right or right-to-left (hence 2-way). This makes them able to read portions of the input tape more than once. They differ from Turing Machines in that the head on the output tape still only moves in one direction.

The use of 2-way FSTs provides not only an alternative computational implementation for reduplication, but it also provides a different perspective on how computationally complex reduplication is. This leads to our second contribution (b) where we show that the majority of reduplicative patterns fit into a subclass of 2-way FSTs which we call **Concatenated Output Strictly Local** functions (C-OSL). These functions are an extension of the OSL functions studied by Chandlee et al. (2015), which in turn are one way to generalize Strictly Local (SL) sets to functions (McNaughton and Papert, 1971; Garcia et al., 1990). This computational classification categorizes the typology in a mathematical manner and opens doors for our third contribution (c) of getting learnability results for reduplication by developing a learning algorithm C-k-OSL functions.[2]

We flesh out these contributions as follows. Mathematical preliminaries are provided in Section 2. In Section 3, we define our enriched finite-state representation (2-way FSTs) and discuss how it models reduplication . In section Section 4, we discuss how formal language theory provides subclasses for different types of FSTs which map to different phenomena in morpho-phonology and reduplication. We summarize the relevant literature on subclasses for 1-way FSTs in Section 4.1. We introduce our own C-OSL subclass for 2-way FSTs in

2. Like the SL sets, the OSL class, and hence the C-OSL class, is parameterized by a positive integer k, which specifies the size of the locality window. For k-OSL functions, this means that the state of the model is determined solely by the most recent $k-1$ segments that were written to output.

Section 4.2 and show how it maps to the typology of reduplication. A learning result for this 2-way FST subclass is provided in Section 5. Conclusions are in Section 6.

2. Preliminaries

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_1w_2. The definition of the prefixes, suffixes, and unique k-length suffix of a string w are (2),(3),(4) respectively.

(2) $\textbf{Pref}(w) = \{p \in \Sigma^* \mid (\exists s \in \Sigma^*)[w = ps]\}$

(3) $\textbf{Suff}(w) = \{s \in \Sigma^* \mid (\exists p \in \Sigma^*)[w = ps]\}$

(4) $\forall n \in \mathbb{N}$, $\textbf{Suff}^{\mathbf{n}}(w)$ is the suffix of w of length n if $|w| \geq n$; otherwise $\textbf{Suff}^{\mathbf{n}}(w) = w$

For a string set S, its longest common prefix is defined in (5):

(5) $\textbf{lcp}(S) = p \in \bigcap_{w \in S} \text{Pref}(w)$ such that $\forall p' \in \bigcap_{w \in S} \text{Pref}(w), |p'| \leq |p|$

Let $f : A \to B$ be a function f with domain A and co-domain B. For the input alphabet Σ and output alphabet Δ, the prefix function $f^p : \Sigma^* \to \Delta^*$ associated to f is defined in (6). The tails of a string $w \in \Sigma^*$ with respect to a function $f : \Sigma^* \to \Delta^*$ are defined in (7).

(6) $f^p(w) = \textbf{lcp}(f(w\Sigma^*))$

(7) $\textbf{tails}_f(x) = \{(y, v) \mid f(xy) = uv \wedge \ u = f^p(x)\}$

Illustrations of the above concepts can be found in Chandlee et al. (2015).

3. 2-way FSTs: A computational model of reduplication

Most transformations and processes in morphology and phonology can be modeled with 1-way FSTs (Roark and Sproat, 2007; Chandlee, 2014) which read the input once and in one direction. However, reduplication has been a difficult process to model with 1-way FSTs (Roark and Sproat, 2007). It has been considered a stumbling block to computational morphology since the earliest years of the field (Culy, 1985).

In this section, we alternatively model reduplication with 2-way FSTs. These are an enriched class of finite-state transducers which can read the input tape multiple times by being able to go back and forth on the input tape (thus being two-way). Although as automata, a 2-way finite-state automaton (2-way FSA) is just as expressive as 1-way finite-state automaton (1-way FSA) (Shallit, 2008), a 2-way finite-state transducer (2-way FST) is strictly more powerful than a 1-way finite-state transducer (1-way FST) as witnessed by total reduplication (Filiot and Reynier, 2016). Below is a formalization of 2-way FSTs based on Filiot and Reynier (2016) and Shallit (2008).

Definition 1 (2-way deterministic FST) *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,*
$\delta : Q \times \Sigma \to Q \times \Gamma^* \times D$ *is the transition function where the direction* $D = \{-1, 0, +1\}$.

The direction of a transition can be to move to the next symbol on the input tape (+1), stay put (0), or move to the previous symbol (-1). A 1-way FST is simply a 2-way FST where the direction parameter for every transition arc is set to +1 (i.e. only read from left to right). Inputs to a 2-way FST are flanked with the start ($\rtimes$) and end boundaries ($\ltimes$). This larger alphabet is denoted by $\Sigma_{\ltimes}$.

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 $(wqax, u) \to (wrax, uv)$. 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 $(waqx, u) \to (wrax, uv)$.

The transitive closure of $\to$ is denoted with $\to^+$. Thus, if $c \to^+ c'$ then there exists a finite sequence of configurations $c_1, c_2 \dots c_n$ with $n > 1$ such that $c = c_1 \to c_2 \to \dots \to c_n = c'$. 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 $(q_0 \rtimes w \ltimes, \lambda) \to^+ (\rtimes w \ltimes q_f, u)$. 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. This occurs for input $wx \in \Sigma_{\ltimes}^*$ whenever there exist $q \in Q$ and $u, v \in \Gamma^*$ such that $(q_0 wx, \lambda) \to^+ (wqx, u) \to^+ (wqx, uv)$.

To illustrate, consider total reduplication in Indonesian (8) (Cohn, 1989, 185).

(8)	a.	buku → buku~buku	'book'→'books'
	b.	wanita → wanita~wanita	'woman'→'women'
	c.	maʃarakat → maʃarakat~maʃarakat	'society'→'societies' 'lack'→'lacks'

This total reduplication process can be modeled with the 2-way FST in Figure 1. Note that each transition arc is a 3-tuple of *(input symbol, output symbol, direction)*. The symbol Σ is a variable ranging over any symbol in the alphabet except for the edge boundaries.

Virtually all forms of reduplication that we surveyed can be modeled with 2-way FSTs. We have constructed the RedTyp database[3] which contains entries for 138 reduplicative

3. A copy of RedTyp can be found online at https://github.com/jhdeov/RedTyp.

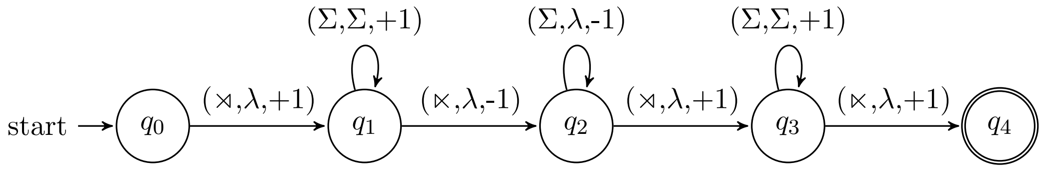


Figure 1: 2-way FST for total reduplication

processes from 90 languages gleaned from various surveys (Moravcsik, 1978; Inkelas and Downing, 2015). It contains 57 distinct 2-way FSTs for them. An example of a 2-way FST for partial reduplication is given in Section 4.2. On average, these 2-way FSTs had 8.8 states and were easy to write, debug, and implement in Python. The 2-way FSTs directly captured the natural language data without requiring any approximations or restrictions on the size of the input, nor did they require a high number of states in order to remember all possible reduplicant sizes. This shows the merit of 2-way FSTs in computationally modeling reduplication in a way that is transparent, concise, convenient, and linguistically motivated.

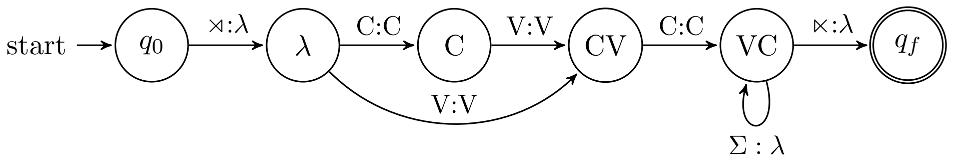
These 2-way FSTs are linguistically motivated in two senses: their ability to model total reduplication productively, and their ability to capture the copying aspect of total and partial reduplication. 1-way FSTs for reduplication are often criticized for treating reduplication as a memorization task whereby the machine simply memorizes a finite list of all possible combinations of two or more copies (Roark and Sproat, 2007). For total reduplication, 1-way FSTs do not exactly capture it but approximate it (Hulden, 2009). When given a new word, a 1-way FST cannot apply total reduplication to it; this means that they cannot capture the productivity of total reduplication.

As for partial reduplication, a 1-way FST can capture the productivity of partial reduplication. This is because the set of pairs of partial copies is finite since only a bounded number of segments are copied. However, 1-way FSTs do not capture the the intensional description behind partial reduplication as a copying task and not a memorization task. Theoretically, reduplication is a copying task where a segment in the input is produced multiple times in the output. The subtle difference between remembering vs. copying in the computational modeling of reduplication is beyond the scope of this article, but see Dolatian and Heinz (2018) which explains this distinction using origin semantics (Bojańczyk, 2014).

4. Computational typology with FSTs

2-way FSTs are equivalent in expressivity to MSO logical transductions and Streaming String Transducers (Engelfriet and Hoogeboom, 2001; Alur, 2010). They are closed under composition and their non-deterministic variants are invertible (Courcelle and Engelfriet, 2012). Here we show that reduplication does not require the full power of 2-way FSTs but falls within certain subclasses, and thus has a demarcable computational complexity.

Segmental phonology can be modeled with 1-way FSAs and FSTs; however, it does not require their full power (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 (Heinz and Lai, 2013; Chandlee et al., 2014, 2015). Some



$$\begin{tikzcd} \text{start} \arrow[r] & q_0 \arrow[r, "⋊:\lambda"] & \lambda \arrow[r, "C:C"] \arrow[rr, bend right, "V:V"'] & C \arrow[r, "V:V"] & CV \arrow[r, "C:C"] & VC \arrow[r, "⋉:\lambda"] \arrow[loop below, "\Sigma:\lambda"] & q_f \end{tikzcd}$$

Figure 2: OSL 1-way FST for English nickname formation

of these subclasses have been argued to characterize different types of phonological well-formedness conditions and transformations (Heinz, 2018). In Section 4.1, we discuss the subclass of Output-Strictly Local (OSL) functions (Chandlee et al., 2015). In Section 4.2, we propose the C-OSL subclass of functions describable with 2-way FSTs based on these OSL functions, and show how they model the bulk of reduplicative typology.

4.1. OSL subclass of 1-way FSTs

As said, virtually all segmental phonology can be modeled with proper subclasses of 1-way FSTs. One subclass in specific is Output Strictly Local (OSL) functions. OSL functions can be either left-OSL (L-OSL) if they read the input and write the output left-to-right, or right-OSL (R-OSL) if they read the input and write the output right-to-left. We give a formal definition of L-OSL functions in Definition 2 from Chandlee et al. (2015).

Definition 2 (Left Output-Strictly Local functions) *A function f is k-L-OSL for an integer k if for all w_1, w_2 in Σ^*, whenever $\mathbf{Suff}^{k-1}(f^p(w_1)) = \mathbf{Suff}^{k-1}(f^p(w_2))$, it is the case that $\mathbf{tails}_f(w_1) = \mathbf{tails}_f(w_2)$.*

Informally, a 1-way FST models a k-OSL function if the states of the 1-way FST keep track of the last $k-1$ output symbols which were written to the output tape.

An example of an OSL function is nickname formation in English. This truncation process will output the first (C)VC[4] of the input but delete everything after.[5]

(9)	a.	dʒɛfɹi→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 symbol. Intuitively, the 3-OSL 1-way FST function in Figure 2 will output up until the first VC of the input and then stop outputting anything after that.[6]

4. (C)VC is any string consisting of a vowel-consonant sequence optionally preceded by a consonant.

5. In (9), the symbols on the left are drawn from the International Phonetic Alphabet (International Phonetic Association, 1999).

6. See Chandlee (2017) on why this function is necessarily OSL and not other subregular functions such as Input-Strictly Local (ISL). We have simplified the analysis by not considering cases of complex onsets in the input, e.g. stivɪn → stiv (‘Steven’→‘Steve’).

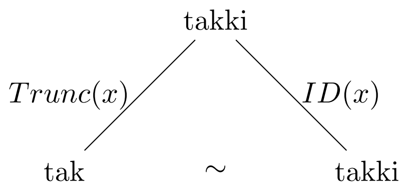


Figure 3: Initial-CVC reduplication as a concatenation of functions.

A significant proportion of segmental phonology can be modeled with OSL functions.[7] Their relatively low complexity has opened doors to understanding the cognitive limitations and learnability of phonological processes (Chandlee et al., 2015). Chandlee et al. (2015) introduce a learning algorithm *OSLFIA* and study it. Their main result is Theorem 3.

Theorem 3 (Learnability of k-L-OSL functions) *OSLFIA identifies k-L-OSL functions in polynomial time and data.*

Readers are referred to de la Higuera (1997) for details on learning in polynomial time and data and Eyraud et al. (2016) for a discussion of related issues. The next section shows how extending OSL functions to 2-way FSTs opens similar doors for reduplication.

4.2. C-OSL subclass of 2-way FSTs

We propose below the C-OSL subclass for 2-way FSTs.

Definition 4 (Concatenated Output-Strictly Local functions) *: A function f is Concatenated k-Output-Strictly Local (C-k-OSL) for an integer k if the function is the concatenation of two or more k-OSL functions, e.g. $f(x) = o_1(x) \cdot o_2(x) \cdot \ldots \cdot o_n(x)$ where o_i is a k-OSL function.*

A function is C-L-OSL if each of its component OSL functions is L-OSL. It is C-R-OSL if each of its component OSL functions is R-OSL.

Intuitively, a C-OSL function is a function that takes as input x, gives x to n many separate 1-way FSTs which are OSL, and concatenates their output.[8] To illustrate the insight behind C-OSL functions, consider initial-CVC partial reduplication from Agta (10).

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

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 3.

7. Higher classes of 1-way FSTs (Weakly Deterministic and Regular) have been proposed for cases when segmental processes interact with morphology or when suprasegmental processes such as tone are involved (Heinz and Lai, 2013; Jardine, 2016a).

8. In terms of function combinatorics for regular string transformations (Alur et al., 2014), the class of C-OSL functions involves the use of a 'sum combinator' $\bigotimes$ that concatenates the output of two or more OSL functions: $f(x) = o_1(x) \bigotimes o_2(x) \bigotimes \ldots o_n(x)$ where o_i is an OSL function. This is similar to the use of product automata. See Alur et al. (2014) for details.

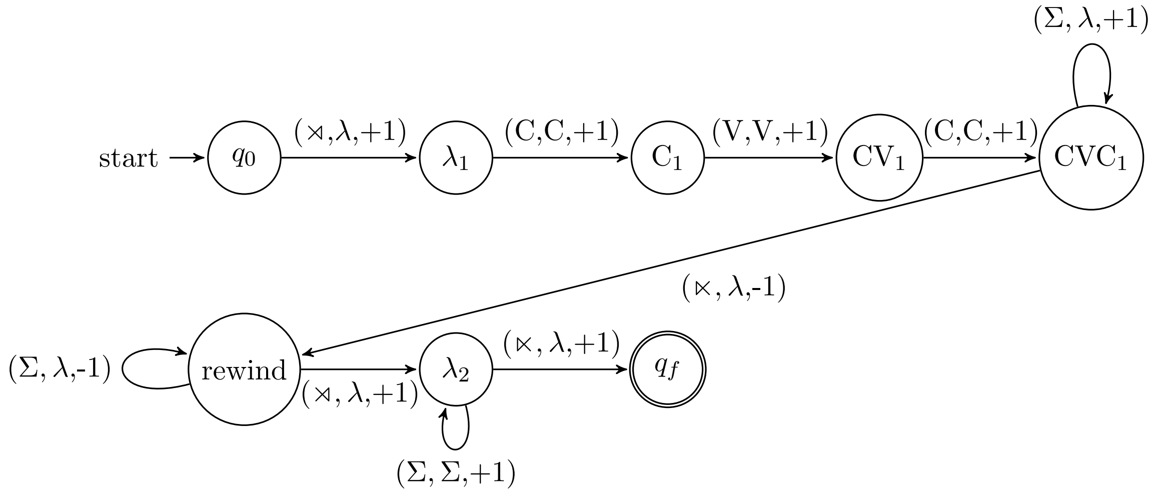


Figure 4: C-OSL 2-way FST for initial-CVC partial reduplication

The first function, here labeled $Trunc(x)$, 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*.

The truncation function $Trunc(x)$ is a 3-OSL function because it outputs a truncation of the input to just the first CVC (similar to English nickname formation). The identity function $ID(x)$ is 1-OSL. Each of the two functions, $Trunc(x)$ and $ID(x)$, are L-OSL functions that can be modeled with 1-way FSTs. Figure 4 illustrates a 2-way FST for initial-CVC reduplication which is formulated as a concatenation of two OSL functions.

In our survey of reduplication described in Section 3, ***most*** forms of total and partial reduplication are C-OSL functions. As mentioned in Section 3, we have developed our own database of reduplicative patterns, RedTyp, which consists of 138 reduplicative patterns from 91 languages. Of these 138 reduplicative processes, 121 (87%) are C-OSL functions.

What is the significance of the 13% non-C-OSL patterns? First, this does not mean that we estimate that 13% of the cross-linguistic typology of reduplicative processes is not C-OSL. This is because RedTyp under-represents cases of opacity wherein phonological processes will exceptionally apply either across both or neither copies because of a drive to maintain identity between the two copies (McCarthy and Prince, 1995). Only 5% of RedTyp displays opacity. RedTyp under-represents cases of opacity in reduplication because our main source, Moravcsik (1978), did not list opaque cases. Depending on the specific process, opacity may necessitate using 2-way FSTs that are more powerful than C-OSL, specifically compositions of C-OSL functions. We are still in the process of incorporating more cases of opacity from McCarthy and Prince (1995). Second, some reduplicative processes are sensitive to syllable count and foot structure. We have treated these processes as non-C-OSL because we assume that the input is a linear string of symbols. If our input representation were enriched with prosodic or autosegmental information (Jardine, 2016b; Strother-Garcia, 2018), then these may arguably be C-OSL.

5. Learning C-OSL functions

Given a computational representation of reduplication (C-OSL 2-way FSTs), learning reduplication is now reducible to inducing those representations from examples. Here, we provide a learning algorithm for C-OSL 2-way FSTs. We call this learner the Boundary-Based C-OSL inference algorithm (BB-COSLIA) because it makes crucial use of the boundary symbol being present in the data. Pseudo-code for BB-COSLIA is in Algorithm 1.

BB-COSLIA is dependent on having an independent OSL learner such as OSLFIA (Chandlee et al., 2015). We find this dependence an advantage because it builds on preexisting results in grammatical inference and modularizes the learning task for reduplication.

To simplify matters, the BB-COSLIA in Algorithm 1 is written for the usual case where the 2-way FSTs are equivalent to the concatenation of two Left k-OSL functions. For any given natural numbers k and n, BB-COSLIA can be easily adapted to learn C-k-OSL functions that are equivalent to the concatenation of n k-OSL functions, as long as n, k, and and the direction of the OSL functions are given as *a priori*.

Algorithm 1: BB-COSLIA Algorithm

```
Input: A finite sample $S \subset \{\rtimes\}\Sigma^*\{\ltimes\} \times \Delta^*\{\sim\}\Delta^*$ and $k \in \mathbb{N}$
Output: C-$k$-OSL 2-way FST
$H1 \leftarrow \{\}$;
$H2 \leftarrow \{\}$;
for $(u, l \sim r) \in S$ do
    $H1 \leftarrow H1 \cup (u, l)$;
    $H2 \leftarrow H2 \cup (u, r)$;
end
$T_{H1} \leftarrow$ OSLFIA($k$, H1);
$T_{H2} \leftarrow$ OSLFIA($k$, H2);
return $T_{H1} \cdot T_{\texttt{Rev}} \cdot T_{H2}$ ;
```

To learn a function $f : \Sigma^* \to \Delta^*$, BB-COSLIA takes as input the *boundary-enriched* data sample S and a positive integer k. The sample S includes pairs of input and output strings exemplifying the function f. The set S is boundary-enriched if for each pair $(u, v) \in S$, the string v includes a reduplicant boundary symbol $\sim$ which marks the boundary between the two copies in the output string.

We illustrate BB-COSLIA with an example. To learn an initial-CVC reduplicative process as in Agta (10), BB-COSLIA takes as input a natural number $k = 3$ and a boundary-enriched sample S which consists of the input-output pairs in (11).

(11) Example Data $= \{$(cat, cat$\sim$cat), (bird, bir$\sim$bird), (music, mus$\sim$music) $\ldots\}$

Since initial-CVC reduplication is a C-OSL function, the output of BB-COSLIA will be a C-OSL 2-way FST which models initial-CVC reduplication as in Figure 4. To generate this 2-way FST, BB-COSLIA learns the corresponding C-OSL function by first learning the individual OSL functions which comprise the C-OSL function: the truncation function and the identity function. It does so by separating the sample set S into two data sets: one set $H1$ for learning the first half of the transduction, the truncation function, and set

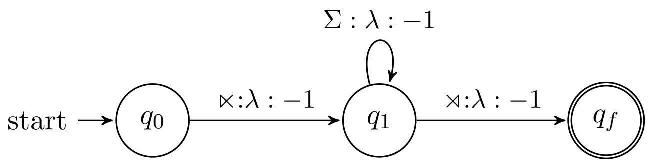


Figure 5: 2-way FST $T_{\texttt{Rev}}$ which reads the input right-to-left and outputs the empty string.

$H2$ for learning the second half, the identity function. The two data sets are constructed by using the reduplicative boundary symbol from our initial data set S. The sets $H1, H2$ are submitted to a k-OSL learner such as OSLFIA (Chandlee et al., 2015) which returns two corresponding 1-way FSTs T_{H1} and T_{H2} for the data sets $H1$ and $H2$ respectively. BB-COSLIA also uses the 2-way FST $T_{\texttt{Rev}}$, which reads the input string from right to left and outputs the empty string. The learner concatenates $T_{H1}, T_{\texttt{Rev}}$, and T_{H2} to form a single 2-way FST and returns it.

BB-COSLIA provably returns the target function if the set $H1$ is a characteristic sample for the OSL function modeled by the first function T_{H1}, and the set $H2$ is a characteristic sample for the OSL function modeled by the second function T_{H2}.[9] The key to the BB-COSLIA learner is that the input was representationally enriched with the reduplication boundary symbol $\sim$. It used this boundary to determine which parts of substrings in the output corresponded to the different OSL functions whose concatenation defines the reduplication process. Without the reduplicative boundary $\sim$, the learning task is more difficult. The problem of learning reduplicative processes in the absence of such a marker, with the result here, reduces to the task of morpheme segmentation, which is an open problem (Goldsmith et al., 2017).

6. Conclusion

In this paper, we have shown that natural language reduplication can be modeled using 2-way FSTs. Specifically, a large proportion of the typology on reduplication can be modeled with a subclass of 2-way FSTs which we call C-OSL functions. These functions are an extension of the OSL class of functions discovered for 1-way FSTs. We sketched a provably-correct learning algorithm for the C-OSL class of functions by extending the OSL learner used by Chandlee et al. (2015) and by enriching the learner's sample data with boundary symbols that mark the juncture between the reduplicated copies in the output.

Many questions are left unanswered. First, as stated in Section 4.2, 13% of the surveyed reduplicative processes require functions that are more powerful than C-OSL. Second, our learner requires that the sample data be enriched with a special boundary symbols. Future work will focus on better describing the computational nature of these non-C-OSL functions and on developing learners without boundary-enriched samples. Third, partial reduplication can be modeled with either 1-way or 2-way FSTs with different intensional descriptions; it is an open question if there is a (severe) learnability trade-off between these two models.

9. Criteria on what constitutes a characteristic sample for learning OSL functions with OSLFIA can be found in Chandlee et al. (2015).

Acknowledgments

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

References

Daniel M. Albro. *Studies in computational Optimality Theory, with special reference to the phonological system of Malagasy.* PhD thesis, University of California, Los Angeles, 2005.

Rajeev Alur. Expressiveness of streaming string transducers. In *Proceedings of the 30th Annual Conference on Foundations of Software Technology and Theoretical Computer Science,*, volume 8, pages 1–12, 2010.

Rajeev Alur, Adam Freilich, and Mukund Raghothaman. 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, 2014.

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

Mikołaj Bojańczyk. Transducers with origin information. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 26–37, Berlin, Heidelberg, 2014. Springer.

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

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

Jane Chandlee and Jeffrey Heinz. 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, June 2012. Association for Computational Linguistics.

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

Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. Output strictly local functions. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*, pages 112–125, Chicago, USA, July 2015.

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

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

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

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

Colin de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning*, 27(2):125–138, 1997.

Hossep Dolatian and Jeffrey Heinz. Modeling reduplication with 2-way finite-state transducers. In *Proceedings of the 15th SIGMORPHON Workshop on Computational Research in Phonetics, Phonology, and Morphology*, Brussells, Belgium, October 2018. Association for Computational Linguistics.

Joost Engelfriet and Hendrik Jan Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Logic*, 2(2), April 2001.

Rémi Eyraud, Jeffrey Heinz, and Ryo Yoshinaka. Efficiency in the identification in the limit learning paradigm. In Jeffrey Heinz and José M. Sempere, editors, *Topics in Grammatical Inference*, pages 25–46. Springer, Berlin, Heidelberg, 2016.

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

Pedro Garcia, Enrique Vidal, and José Oncina. Learning locally testable languages in the strict sense. In *Proceedings of the Workshop on Algorithmic Learning Theory*, pages 325–338, 1990.

John A Goldsmith, Jackson L Lee, and Aris Xanthos. Computational learning of morphology. *Annual Review of Linguistics*, 3:85–106, 2017.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 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)*, pages 1631–1640, Berlin, Germany, August 2016. Association for Computational Linguistics.

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

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

Jeffrey Heinz and William Idsardi. What complexity differences reveal about domains in language. *Topics in Cognitive Science*, 5(1):111–131, 2013.

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

Mans Hulden. *Finite-state machine construction methods and algorithms for phonology and morphology.* PhD thesis, The University of Arizona, Tucson, AZ, 2009.

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

International Phonetic Association. *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet.* Cambridge University Press, 1999.

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

Adam Jardine. *Locality and non-linear representations in tonal phonology.* PhD thesis, University of Delaware, Newark, DE, 2016b.

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

John J McCarthy and Alan Prince. Faithfulness and reduplicative identity. In Jill N. Beckman, Laura Walsh Dickey, and Suzanne Urbanczyk, editors, *Papers in Optimality Theory.* GLSA, University of Massachusetts, Amherst, MA, 1995.

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

Edith Moravcsik. Reduplicative constructions. In Joseph Greenberg, editor, *Universals of Human Language*, volume 1, pages 297–334. Stanford University Press, Stanford, 1978.

Andrew Nevins. 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*, pages 113–126. MITWPL, Cambridge,MA, 2004.

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

James Rogers and Geoffrey Pullum. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20:329–342, 2011.

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

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

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

Kristina Strother-Garcia. Imdlawn tashlhiyt berber syllabification is quantifier-free. In *Proceedings of the Society for Computation in Linguistics (SCiL) 2018*, volume 1, pages 145–153, 2018. doi: 10.7275/R5J67F4D.