Efficient Learning of Output Tier-Based Strictly 2-Local Functions

Phillip Burness
University of Ottawa
pburn036@uottawa.ca

Kevin McMullin
University of Ottawa
kevin.mcmullin@uottawa.ca

Abstract

This paper characterizes the Output Tier-based Strictly k-Local ($OTSL_k$) class of string-to-string functions, which are relevant for modeling long-distance phonological processes as input-output maps. After showing that any $OTSL_k$ function can be learned when k and the tier are given, we present a new algorithm that induces the tier itself when $k = 2$ and provably learns any total $OTSL_2$ function in polynomial time and data—the first such learner for any class of tier-based functions.

1 Introduction

In this paper, we investigate the class of Output Tier-based Strictly k-Local ($OTSL_k$) functions. In terms of finite state transducers, $OTSL_k$ functions are those for which the output symbol(s) to be written at each timestep depends on the $k-1$ most recent symbols on the output tape that belong to the relevant 'tier' (a subset of the output alphabet; Heinz et al., 2011), without regard for any non-tier symbols that might have been written between them or after them. We show that they are learnable when the contents of the tier are provided as input to the learner, and introduce an algorithm that provably and efficiently learns any total OTSL function when $k = 2$.

Recent research investigating the computational properties of phonological patterns observed in natural language has shown that many attested processes can be characterized as Strictly Local (SL) functions (Chandlee, 2014; Chandlee et al., 2014, 2015). That is, the output at any given timestep is dependent on the previous $k-1$ symbols from either the input string (Input Strictly k-Local; ISL_k) or the output string (Output Strictly k-Local; OSL_k). Multiple characterizations of these classes exist and their properties are well-understood. One important distinction between the two is that non-iterative processes are ISL, whereas processes that apply iteratively to multiple targets are OSL. Moreover, efficient learning algorithms exist for both the ISL and OSL functions. The OSL Function Inference Algorithm (OSLFIA; Chandlee et al., 2015) is of particular importance to this paper, as we will show that many of their theoretical results can be generalized to OTSL functions in a natural way.

Long-distance phonological processes, for which a potentially unbounded number of segments may intervene between the trigger and target without being affected in any way, are neither ISL nor OSL for any value of k. For example, Samala has a long-distance process of sibilant harmony in which an underlying /s/ surfaces as [ʃ] if another [ʃ] appears anywhere later in the word. This is seen when, e.g., the perfective suffix /-waʃ/ is added to a root containing /s/, as in /ha-s-xintila-waʃ/→[haʃxintilawaʃ] 'his former gentile name' (Applegate, 1972). This process can be understood as applying iteratively to multiple targets, as in /s-lu-sisin-waʃ/→[ʃluʃiʃinwaʃ] 'It is all grown awry'. Indeed, it seems that the vast majority of attested long-distance processes are enforced iteratively (Kaplan, 2008; Hansson, 2010). As such, we focus this paper on $OTSL_k$ functions in particular, which generalize the OSL_k class in a way that allows us to model these kinds of long-distance processes. We note that $ITSL_k$ functions can be characterized in a similar way and that the learning strategy outlined below could likely be extended to total $ITSL_2$ functions.

While the notion of a tier has long been incorporated into phonological theory (e.g., Clements, 1980; Goldsmith, 1990; Odden, 1994; Heinz et al., 2011; McMullin, 2016), the range of possible tiers is typically assumed to be available to the learner *a priori*. Each possible tier could, for example, be defined in terms of feature specifications or natural classes of segments (e.g., Hayes and Wilson, 2008). Though algorithms have been developed for inducing a relevant tier from a sample of positive training, their success is limited to phonotactic co-occurrence restrictions. This is true both for constraint-based maximum entropy learners (Gouskova and Gallagher, 2019) as well as for algorithms that learn grammars for Tier-based Strictly Local formal languages (Jardine and Heinz, 2016; Jardine and McMullin, 2017). To our knowledge, the algorithm presented below, which we call the Output Tier-based Strictly 2-Local Function Inference Algorithm (OTSL2FIA), is the first algorithm which learns the relevant tier for transformations of underlying representations (strings of input segments) to surface forms (strings of output segments).

The remainder of this paper is organized as follows. Notation and relevant concepts are presented in Section 2. In Section 3, we define the OTSL functions and characterize them in terms of finite state transducers. In Section 4, we highlight several important properties of OTSL$_2$ functions in particular that can be taken advantage of during learning. All aspects of the learning algorithm, along with the theoretical learning results, are described in Section 5. Section 6 discusses how OTSL$_2$ functions can model various phonological processes and identifies several avenues for future research. Section 7 concludes.

2 Preliminaries

2.1 Strings and sets

Given a set S, we write $\texttt{card}(S)$ to denote its cardinality. For a string w made of symbols from some alphabet Σ, $|w|$ denotes the length of the string. We write Σ^* to denote all possible strings made from the alphabet Σ, while Σ^n denotes all possible strings made from that alphabet with a length of n, and $\Sigma^{\leq n}$ denotes all such strings with a length up to n. The unique string of length 0 (the empty string) is written as λ. Given two strings u and v, we write $u \cdot v$ to denote their concatenation, but often shorten this to uv when context permits. We write $\texttt{fac}_k(w)$ to denote all the contiguous substrings of length k (the k-factors) contained in a string w.

We assume a fixed but arbitrary total order $\prec$ over the letters of Σ, an order which we extend to all strings in Σ^* by defining the *length-lexicographical order* (Oncina et al., 1993; Chandlee et al., 2015) as follows. String w_1 occurs length-lexicographically before w_2 (written as $w_1 \triangleleft w_2$) when $|w_1| < |w_2|$ or, if $|w_1| = |w_2|$, when $a_i \prec b_i$ where a_i is the i^{th} letter in w_1, b_i is the i^{th} letter in w_2, and i is the first position on which w_1 and w_2 differ. For example, given $\Sigma = \{a, b\}$ where $a \prec b$, we have $\lambda \triangleleft a \triangleleft b \triangleleft aa \triangleleft ab \triangleleft ba \triangleleft bb \triangleleft aaa$ and so on.

A prefix of some string w is any string u such that $w = ux$ and $x \in \Sigma^*$. Similarly, a suffix of some string w is any string u such that $w = xu$ and $x \in \Sigma^*$. Note that any string is a prefix and suffix of itself, and that λ is a prefix and suffix of every string. When $|w| \geq n$, $\texttt{Pref}^n(w)$ and $\texttt{Suff}^n(w)$ denote the unique prefix and suffix of w with a length of n; when $|w| < n$, they simply denote w itself. We write $\texttt{Pref}^*(w)$ to denote the set of all prefixes of w. Also, $\texttt{Suff}^n(\texttt{Suff}^n(w_1)w_2)) = \texttt{Suff}^n(w_1 w_2)$. Given a string w, one of its prefixes p, and one of its prefixes s, we write $p^{-1} \cdot w$ to represent the string w without that prefix p and write $w \cdot s^{-1}$ to represent the string w without that suffix s. For example, $a^{-1} \cdot aba = ba$ and $aba \cdot a^{-1} = ab$. Finally, given a set of strings S, we write $\texttt{lcp}(S)$ to denote the *longest common prefix* of S, which is the string u such that u is a prefix of every $w \in S$, and there exists no other string v such that $|v| > |u|$ and v is also a prefix of every $w \in S$.

2.2 Functions and transducers

This paper deals exclusively with string-to-string functions, relations that pair every $w \in \Sigma^*$ with at most one $y \in \Delta^*$, where Σ and Δ are the input alphabet and output alphabet respectively. The input language and output language of such a function are $\texttt{pre\_image}(f) = \{x \mid (\exists y)[x \mapsto_f y]\}$ and $\texttt{image}(f) = \{y \mid (\exists x)[x \mapsto_f y]\}$, respectively. An important concept is that of the *tails* of an input string w with respect to a function f.

Definition 1. (Tails) *Given a function f and an input $w \in \Sigma^*$, $\texttt{tails}_f(w) = \{(y, v) \mid f(wy) = uv \wedge u = \texttt{lcp}(f(w\Sigma^*))\}$.*

In words, $\texttt{tails}_f(w)$ pairs every possible string $y \in \Sigma^*$ with the portion of $f(wy)$ that is directly attributable to y. That is, it describes the effect that w has on the output of any subsequent string of input symbols. When $\texttt{tails}_f(w_1) = \texttt{tails}_f(w_2)$ we say that w_1 and w_2 are *tail-equivalent* with respect to f.

A related concept to tails and tail-equivalency is the *contribution* of a symbol $a \in \Sigma$ relative to a string $w \in \Sigma^*$ with respect to a function f.

Definition 2. (Contribution) *Given a function f, some $a \in \Sigma$, and some $w \in \Sigma^*$, $\texttt{cont}_f(a, w) = \texttt{lcp}(f(w\Sigma^*))^{-1} \cdot \texttt{lcp}(f(wa\Sigma^*))$.*

In words, for an input string x that has the prefix wa, the contribution of the a in wa is the portion of $f(x)$ that is uniquely and directly attributable to that instance of a.

The Output Tier-based Strictly Local functions that will be introduced below are a proper subclass of the subsequential functions. Oncina and García (1991) show that when a function is subsequential, tail-equivalency will partition Σ^* into finitely many blocks, allowing us to construct a finite-state transducer that computes f. In this paper we use delimited subsequential finite state transducers (DSFSTs; see Jardine et al., 2014), to characterize the class of Output Strictly Local (OSL) functions. The following definition is drawn directly from Chandlee et al. (2015).

Definition 3. *A delimited subsequential finite state trasnducer (DSFST) is a 6-tuple $\langle Q, q_0, q_f, \Sigma, \Delta, \delta\rangle$ where Q is a finite set of states, $q_0 \in Q$ is the unique initial state, $q_f \in Q$ is the unique final state, Σ is the finite input alphabet, Δ is the finite output alphabet, and $\delta \subseteq Q \times (\Sigma \cup \{\rtimes, \ltimes\}) \times \Delta^* \times Q$ is the transition function (where $\rtimes \notin \Sigma$ indicates the start of the input and $\ltimes \notin \Sigma$ indicates the end of the input), and the following hold:*

1. *if $(q, a, u, q') \in \delta$ then $q \neq q_f$ and $q' \neq q_0$*
2. *if $(q, a, u, q_f) \in \delta$ then $a = \ltimes$ and $q \neq q_0$*
3. *if $(q_0, a, u, q') \in \delta$ then $a = \rtimes$ and if $(q, \rtimes, u, q') \in \delta$ then $q = q_0$*
4. *if $(q, a, u, q'), (q, a, u', q'') \in \delta$ then $q' = q''$ and $u = u'$*

Each transition $(q, a, u, q') \in \delta$ can be seen as an instruction to append u to the end of the output tape and to move to state q' upon reading a while in state q. This transition function may be partial, and its recursive extension δ^* is the smallest set containing δ closed under the following conditions: $(q, \lambda, \lambda, q) \in \delta^*$, and $(q, w, u, q'), (q', a, v, q'') \in \delta^* \Rightarrow (q, wa, uv, q'') \in \delta^*$. The initial state of a DSFST has no incoming transitions and has exactly one outgoing transition, which will be for the input $\rtimes$ and does not land in the final state. Furthermore, the final state of a DSFST has no outgoing transitions, and every transition into the final state is for the input $\ltimes$. DSFSTs are also deterministic on the input, such that each state has at most one outgoing transition per input symbol.

The size of a DSFST $\mathcal{T} = \langle Q, q_0, q_f, \Sigma, \Delta, \delta\rangle$ is $|\mathcal{T}| = \texttt{card}(Q) + \texttt{card}(\delta) + \sum_{(q,a,u,q')\in\delta} |u|$, and the relation defined by a DSFST is $\mathcal{R}(\mathcal{T}) = \{(x, y) \in \Sigma^* \times \Delta^* \mid (q_0, \rtimes x \ltimes, y, q_f) \in \delta^*\}$.

The DSFSTs we will use below have a special property known as *onwardness*, which informally means that the writing of the output is never delayed. The following formal definition of onwardness and a related lemma are borrowed from Chandlee et al. (2015).

Definition 4. (Onwardness) *A DSFST is* onward *if for every $w \in \Sigma^*$ and $u \in \Delta^*$, $(q_0, \rtimes w, u, q) \in \delta^* \Longleftrightarrow u = \texttt{lcp}(f(w\Sigma^*))$.*

Lemma 1. *Let the outputs of the edges out of state q be $\texttt{Outputs}(q) = \{u \mid (\exists a \in \Sigma \cup \{\rtimes, \ltimes\})\ (\exists q' \in Q)\ [(q, a, u, q') \in \delta]\}$. If $\mathcal{T}$ is an onward DSFST and recognizes f, then $\forall q \neq q_0$, $\texttt{lcp}(\texttt{Outputs}(q)) = \lambda$ and $\texttt{lcp}(\texttt{Outputs}(q_0)) = \texttt{lcp}(f(\Sigma^*))$.*

Below we will frequently make reference to the length-lexicographically earliest input string that can lead to a state q in a given transducer $\mathcal{T}$, which we will denote as w_q. A formal definition is provided here for reference.

Definition 5. (Earliest string) *Given a transducer $\mathcal{T} = \langle Q, q_0, q_f, \Sigma, \Delta, \delta\rangle$, the earliest string that leads to $q \in Q$ is $w_q = min_{\triangleleft}\{w \in \Sigma^* \mid \exists u, (q_0, \rtimes w, u, q) \in \delta^*\}$.*

A distinction that will be important throughout the rest of this paper is that between the writing that occurs in a DSFST as it is reading letters from Σ, and the writing that occurs at the very end (when the DSFST reads $\ltimes$). To make this distinction, Chandlee et al. (2015) defined the prefix function f^p associated with a subsequential function f as follows.

Definition 6. (Prefix function) *Given a subsequential function f, its associated prefix function f^p is such that $f^p(w) = \texttt{lcp}(f(w\Sigma^*))$.*

Remark 1. *Given a subsequential function f, some $a \in \Sigma$, and some input string $w \in \Sigma^*$, $\texttt{cont}_f(a, w) = f^p(w)^{-1} \cdot f^p(wa)$ and $\texttt{cont}_f(\ltimes, w) = f^p(w)^{-1} \cdot f(w)$.*

2.3 Strict locality and tiers

Chandlee (2014) and Chandlee et al. (2014) originally introduced the Input Strictly Local (ISL) and Output Strictly Local (OSL) functions, both of which generalize Strictly Local (SL) stringsets to functions based on one of the defining properties of SL languages, the Suffix Substitution Closure (Rogers and Pullum, 2011). The definitions of the ISL and OSL functions exploit a corollary of this defining property, which Chandlee et al. (2015) call Suffix-defined Residuals. For reasons of space, we only discuss the OSL functions below.

Theorem 1. (Suffix Substitution Closure) *A language L is SL if for all strings u_1, v_1, u_2, v_2 there exists a natural number k such that for any string x of length $k-1$, if $u_1xv_1, u_2xv_2 \in L$, then $u_1xv_2 \in L$.*

Corollary 1. (Suffix-defined Residuals) *A language L is SL if for all $w_1, w_2 \in \Sigma^*$, there exists a natural number k such that if $\mathtt{Suff}^{k-1}(w_1) = \mathtt{Suff}^{k-1}(w_2)$ then $\{v \mid w_1v \in L\} = \{v \mid w_2v \in L\}$, that is w_1 and w_2 have the same* residuals *(tails) with respect to L.*

Definition 7. (Output Strictly Local functions) *A function f is OSL$_k$ if for all $w_1, w_2 \in \Sigma^*$, $\mathtt{Suff}^{k-1}(f^p(w_1)) = \mathtt{Suff}^{k-1}(f^p(w_2)) \Rightarrow \mathtt{tails}_f(w_1) = \mathtt{tails}_f(w_2)$.*

Chandlee (2014) and Chandlee et al. (2014, 2015) show that most iterative phonological processes can be modelled with an OSL function, with an important exception being long-distance iterative processes like consonant harmony. This is parallel to the fact that long-distance phonotactics cannot be represented with an SL stringset, which motivated Heinz et al. (2011) to define the Tier-based Strictly Local (TSL) languages—stringsets that are SL after an erasure function has applied, masking all symbols that are irrelevant to the restrictions that the language places on its strings.

Definition 8. (Erasure function) *Given an alphabet Σ, a tier $\Theta \subseteq \Sigma$, and a string $w = a_1...a_n$, $\mathtt{Erase}_\Theta(w) = b_1...b_n$ where for all $i \leq n$, $b_i = a_i$ if $a_i \in \Theta$, else $b_i = \lambda$.*

Informally, $\mathtt{Erase}_\Theta(w)$ returns the string w with all non-tier elements removed. For convenience, we will write $\mathtt{Suff}^n_\Theta(w)$ to mean $\mathtt{Suff}^n(\mathtt{Erase}_\Theta(w))$ in what follows.

Definition 9. (Tier-based Strictly Local languages) *A language L is Tier-based Strictly k-Local (TSL$_k$) if there is a tier $\Theta \subseteq \Sigma$ and a subset $S \subseteq \mathtt{fac}^k(\rtimes\Theta^*\ltimes)$ such that:*

$$L = \{w \in \Sigma^* \mid \mathtt{fac}^k(\rtimes \mathtt{Erase}_\Theta(w)\ltimes) \subseteq S\}$$

3 Output Tier-based Strictly Local functions and transducers

In this section, we define the OTSL functions, which generalize the TSL stringsets to functions in the same way that the OSL functions generalize SL stringsets to functions (see Chandlee, 2014; Chandlee et al., 2015).

Definition 10. (Output Tier-based Strictly Local functions) *A function f is OTSL$_k$ if there is a tier $\Theta \subseteq \Delta$ such that for all $w_1, w_2 \in \Sigma^*$, $\mathtt{Suff}^{k-1}_\Theta(f^p(w_1)) = \mathtt{Suff}^{k-1}_\Theta(f^p(w_2)) \Rightarrow \mathtt{tails}_f(w_1) = \mathtt{tails}_f(w_2)$.*

The OTSL class properly contains the OSL functions, since every OSL$_k$ function can be described as an OTSL$_k$ function whose tier is equal to the entire output alphabet. Note that it is possible for a single OTSL function to be described with more than one tier. For example, the identity function (where $\Sigma = \Delta$ and $f(w) = w$) can be described with any subset of Δ as its tier. We use the term k-tier to describe a tier Θ for which f is OTSL$_k$.

Like the OSL$_k$ functions, the OTSL$_k$ functions can be characterized in automata-theoretic terms. First, we define OTSL$_k$ finite state transducers as follows.

Definition 11. *An onward DSFST $\mathcal{T} = \langle Q, q_0, q_f, \Sigma, \Delta, \delta\rangle$ is OTSL$_k$ for the tier $\Theta \subseteq \Delta$ if:*

1. *$Q = S \cup \{q_0, q_f\}$ with $S \subseteq \Theta^{\leq k-1}$*
2. $(\forall u \in \Delta^*)$
   $[(q_0, \rtimes, u, q') \in \delta \Rightarrow q' = \mathtt{Suff}^{k-1}_\Theta(u)]$
3. $(\forall q \in Q - \{q_0\}, \forall a \in \Sigma, \forall u \in \Delta^*)$
   $[(q, a, u, q') \in \delta \Rightarrow q' = \mathtt{Suff}^{k-1}_\Theta(qu)]$.

Lemmas 2 and 3, together with Theorem 2, show that the OTSL$_k$ functions and the functions represented by OTSL$_k$ transducers exactly correspond.

Lemma 2. *Let $\mathcal{T} = \langle Q, q_0, q_f, \Sigma, \Delta, \delta\rangle$ be an OTSL$_k$ transducer for the tier Θ. The following holds: $(q_0, \rtimes w, u, q) \in \delta^* \Rightarrow q = \mathtt{Suff}^{k-1}_\Theta(u)$.*

Lemma 3. *Any OTSL$_k$ transducer corresponds to an OTSL$_k$ function.*

Theorem 2. *Given an OTSL$_k$ function f and one of its k-tiers Θ, the DSFST $\mathcal{T} = \langle Q, q_0, q_f, \Sigma, \Delta, \delta\rangle$ defined as follows computes f:*

1. $Q = S \cup \{q_0, q_f\}$ *with* $S \subseteq \Theta^{\leq k-1}$
2. $(q_0, \rtimes, u, \mathtt{Suff}^{k-1}_{\Theta}(u)) \in \delta \Longleftrightarrow u = f^p(\lambda)$
3. $a \in \Sigma, (q, a, u, \mathtt{Suff}^{1}_{\Theta}(qu)) \in \delta \Longleftrightarrow$ $(\exists w)\ [f^p(w) = vr \wedge \mathtt{Suff}^{k-1}_{\Theta}(vr) = q \wedge f^p(wa) = vru]$ where $r = t_1x_1t_2x_2...t_{k-1}x_{k-1}$, $t_i \in \Theta, x_i \in (\Delta - \Theta)^*$ and $v = f^p(w) \cdot r^{-1}$
4. $(q, \ltimes, u, q_f) \in \delta \Longleftrightarrow u = f^p(w_q)^{-1} \cdot f(w_q)$

We note that these are trivial extensions of Lemmas 3, 4, and Theorem 2 in Chandlee et al. (2015). Indeed, only two minor changes are necessary for this generalization to OTSL$_k$ functions. First, each instance of $\mathtt{Suff}^{k-1}$ must be replaced with $\mathtt{Suff}^{k-1}_{\Theta}$. Second, in order to account for the fact that non-tier elements may come between relevant tier elements, certain references to a string $q = t_1t_2...t_{k-1}$ must be rewritten as $r = t_1x_1t_2x_2...t_{k-1}x_{k-1}$, where $t_i \in \Theta$ and $x_i \in (\Delta - \Theta)^*$. As the proofs are otherwise identical in structure to those found in Chandlee et al. (2015), we do not provide them here.

It is therefore the case that any OTSL$_k$ function can be represented by an OTSL$_k$ transducer. Informally, this will be an onward DSFST in which the non-initial and non-final states represent the most recent $k-1$ tier symbols written thus far, meaning that this is the only information that will dictate what the DSFST writes upon reading the next input symbol.

As an example, Figure 1 presents an OTSL$_2$ transducer that models the unbounded sibilant harmony in Samala from Section 1. Note that in order to achieve the regressive directionality of the process, we assume that this transducer reads input strings from right-to-left (following, e.g., Heinz and Lai, 2013; Chandlee et al., 2015). Directionality will be further discussed in Section 6.

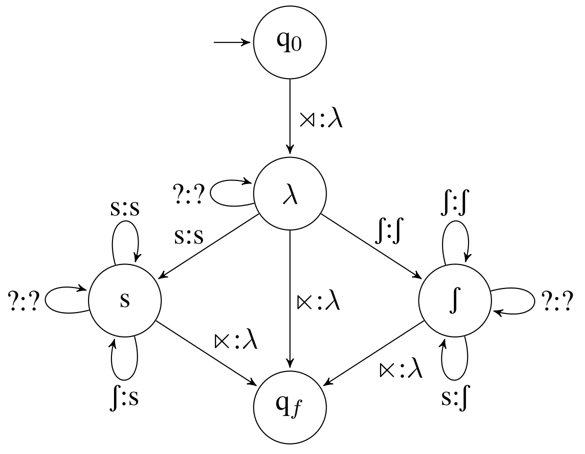

Figure 1: An OTSL$_2$ transducer that models unbounded sibilant harmony, where ? represents any symbol that is not s or ʃ

4 Useful properties of OTSL$_2$ functions

The main goal of this paper is to demonstrate how OTSL functions can be learned from positive data, even without prior knowledge of the tier itself. We note that the tier-induction strategy adopted below relies on certain properties that hold when $k = 2$, but not necessarily for greater values of k. These are outlined below.

First, when an OTSL$_2$ function f can be described with more than one 2-tier, the union of any two or more such 2-tiers is also a 2-tier for f.

Lemma 4. *Given an OTSL$_2$ function f, if $\Theta_1 \subseteq \Delta$ and $\Theta_2 \subseteq \Delta$ are both 2-tiers for f, then $\Omega = \Theta_1 \cup \Theta_2$ is also a 2-tier for f.*

Proof. If $\mathtt{Suff}^1_{\Omega}(f^p(w_1)) = \mathtt{Suff}^1_{\Omega}(f^p(w_2)) = t$, then $t \in \Theta_1$ or $t \in \Theta_2$. If $t \in \Theta_1$, then $\mathtt{Suff}^1_{\Theta_1}(f^p(w_1)) = \mathtt{Suff}^1_{\Theta_1}(f^p(w_2)) = t \Rightarrow \mathtt{tails}_f(w_1) = \mathtt{tails}_f(w_2)$. If $t \in \Theta_2$, then $\mathtt{Suff}^1_{\Theta_2}(f^p(w_1)) = \mathtt{Suff}^1_{\Theta_2}(f^p(w_2)) = t \Rightarrow \mathtt{tails}_f(w_1) = \mathtt{tails}_f(w_2)$. Therefore, $\mathtt{Suff}^1_{\Omega}(f^p(w_1)) = \mathtt{Suff}^1_{\Omega}(f^p(w_2)) = t \Rightarrow \mathtt{tails}_f(w_1) = \mathtt{tails}_f(w_2)$ □

It is this property that allows us to identify a unique *target* tier for an OTSL$_2$ function, which the algorithm can find by flagging and removing elements of Δ from its hypothesis when evidence is found that they cannot be on a relevant tier. We define this canonical 2-tier as follows.

Definition 12. (Canonical 2-tier) *Given an OTSL$_2$ function f, Θ is the canonical 2-tier for f iff there is no other 2-tier $\Omega \subseteq \Delta$ for f such that $\mathtt{card}(\Omega) \geq \mathtt{card}(\Theta)$.*

Remark 2. *Given an OTSL$_2$ function f, its canonical 2-tier Θ is a superset of any 2-tier for f. (This follows immediately from Lemma 4.)*

There is therefore a unique canonical 2-tier (i.e., the largest one) for each OTSL$_2$ function. Interestingly, this can be exploited during the learning process, since it leads to the following useful property of OTSL$_2$ functions.

Lemma 5. *Let $f : \Sigma^* \to \Delta^*$ be an OTSL$_2$ function, let Θ be its canonical 2-tier, and let Ω be such that $\Theta \subset \Omega \subseteq \Delta$. We have $\exists a \in (\Omega - \Theta)$, $\exists w_1, w_2 \in \Sigma^*$, and $\exists x \in \Sigma \cup \{\ltimes\}$ such that $\mathtt{Suff}^1_\Omega(f^p(w_1)) = \mathtt{Suff}^1_\Omega(f^p(w_2)) = a$ and $\mathtt{cont}_f(x, w_1) \neq \mathtt{cont}_f(x, w_2)$.*

Proof. By contradiction. Suppose that the lemma is false. This means that $\forall a \in (\Omega - \Theta)$, $\forall w_1, w_2 \in \Sigma^*$, and $\forall x \in \Sigma \cup \{\ltimes\}$, we have $\mathtt{Suff}^1_\Omega(f^p(w_1)) = \mathtt{Suff}^1_\Omega(f^p(w_2)) = a \Rightarrow \mathtt{cont}_f(x, w_1) = \mathtt{cont}_f(x, w_2)$. Now, since Θ is a 2-tier for f, it is also the case that $\forall b \in \Theta$, $\forall w_1, w_2 \in \Sigma^*$, and $\forall x \in \Sigma \cup \{\ltimes\}$, we have $\mathtt{Suff}^1_\Omega(f^p(w_1)) = \mathtt{Suff}^1_\Omega(f^p(w_2)) = b \Rightarrow \mathtt{cont}_f(x, w_1) = \mathtt{cont}_f(x, w_2)$. Together these imply that $\forall c \in \Omega$, $\forall w_1, w_2 \in \Sigma^*$, and $\forall x \in \Sigma \cup \{\ltimes\}$, we have $\mathtt{Suff}^1_\Omega(f^p(w_1)) = \mathtt{Suff}^1_\Omega(f^p(w_2)) = c \Rightarrow \mathtt{cont}_f(x, w_1) = \mathtt{cont}_f(x, w_2)$.

Since $[\mathtt{Suff}^1_\Omega(f^p(w_1)) = \mathtt{Suff}^1_\Omega(f^p(w_2))$ and $\mathtt{cont}_f(x, w_1) = \mathtt{cont}_f(x, w_2)] \Rightarrow \mathtt{Suff}^1_\Omega(f^p(w_1x)) = \mathtt{Suff}^1_\Omega(f^p(w_2x))$, we also have $\mathtt{cont}_f(y, w_1x) = \mathtt{cont}_f(y, w_2x)$ for all $y \in \Sigma \cup \{\ltimes\}$. This applies recursively, giving us $\mathtt{Suff}^1_\Omega(f^p(w_1)) = \mathtt{Suff}^1_\Omega(f^p(w_2)) \Rightarrow \mathtt{tails}_f(w_1) = \mathtt{tails}_f(w_2)$, which means that Ω is a 2-tier for f. However, $\mathtt{card}(\Omega) > \mathtt{card}(\Theta)$, contradicting the fact that Θ is the canonical 2-tier for f. $\square$

Importantly, it follows from Lemma 5 that for any set Ω which is a strict superset of Θ (the canonical 2-tier), we will always be able to find evidence that some member of Ω could not be a member of any 2-tier for f. It is this property of OTSL$_2$ functions that our algorithm makes use of to determine which output symbols are in Θ. Once again, when $k > 2$, this property does not necessarily hold.[1] Accordingly, we restrict ourselves to $k = 2$ when discussing the learning of OTSL functions without prior knowledge of the tier. While OTSL$_2$ functions seem sufficient for modelling a wide range of long-distance phonological processes, we discuss certain exceptions in Section 6.

[1]For example, if $\Delta = \{a, b, c\}$, there could be an OTSL$_3$ function for which $\Theta_1 = \{a, b\}$ and $\Theta_2 = \{a, c\}$ are both 3-tiers, but $\Omega = \{a, b, c\}$ is not.

5 Learning OTSL functions

5.1 Learning paradigm

We adopt the criterion for successful learning that requires exact identification in the limit from positive data (Gold, 1967), with polynomial bounds on time and data (de la Higuera, 1997). We first define what it means for a class of functions to be represented by a class of representations.

Definition 13. *A class $\mathbb{T}$ of functions is represented by a class $\mathbb{R}$ of representations if every $r \in \mathbb{R}$ is of finite size and there is a total and surjective naming function $\mathcal{L} : \mathbb{R} \to \mathbb{T}$ such that $\mathcal{L}(r) = t$ if and only if for all $w \in \mathtt{pre\_image}(t), r(w) = t(w)$, where $r(w)$ is the output produced by r given the input w.*

The notions of a sample and a learning algorithm are defined as follows.

Definition 14. (Sample) *A sample S for a function $t \in \mathbb{T}$ is a finite set of data consistent with t, that is to say $(w, u) \in S$ iff $t(w) = v$. The size of a sample is the sum of the length of the strings it is composed of: $|S| = \sum_{(w,u) \in S} |w| + |u|$.*

Definition 15. (Learning algorithm) *A $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$ is a program that takes as input a sample for a function $t \in \mathbb{T}$ and outputs a representation from $\mathbb{R}$.*

The paradigm relies on the notion of a characteristic sample, adapted here for functions as in Chandlee et al. (2015).

Definition 16. (Characteristic sample) *For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a characteristic sample of a function $t \in \mathbb{T}$ if for all samples $S \supseteq CS$, $\mathfrak{A}$ returns a representation r such that $\mathcal{L}(r) = t$.*

The learning paradigm can now be defined as follows.

Definition 17. (Identification in polynomial time and data) *A class $\mathbb{T}$ of functions is identifiable in polynomial time and data if there exists a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathfrak{A}$ and two polynomial equations $p()$ and $q()$ such that:*

1. *For any sample S of size m for $t \in \mathbb{T}$, $\mathfrak{A}$ returns a hypothesis $r \in \mathbb{R}$ in $\mathcal{O}(p(m))$ time.*

2. *For each representation $r \in \mathbb{R}$ of size n, with $t = \mathcal{L}(r)$, there exists a characteristic sample of t for $\mathfrak{A}$ of size at most $\mathcal{O}(q(n))$.*

5.2 Learning when the tier is given

Prior to describing the approach we take to inducing the contents of a tier when $k = 2$, we note that learning any OTSL$_k$ function from positive data is relatively straightforward if the value of k and the tier Θ are known beforehand. In particular, although the OSLFIA presented in Chandlee et al. (2015) was designed only to learn OSL functions, it turns out that a minor modification allows us to extend their result to OTSL functions, so long as k and Θ are known beforehand. We summarize how this can be done below.

In its original form, the OSLFIA inevitably fails to learn any OTSL function that is not itself OSL (i.e., where $\Theta \neq \Delta$). Specifically, since the algorithm labels each landing state of a transition with the $k - 1$ suffix of its associated output, it will always incorrectly determine the landing state of one of more transitions when there is a long-distance dependency. Moreover, the exact way in which the resulting OSL transducer differs from the target OTSL transducer is somewhat unpredictable. As such, there does not seem to be a general approach for transforming the OSLFIA's output into an appropriate OTSL$_k$ transducer.

In cases where Θ is known beforehand, however, we can circumvent this issue by simply specifying that non-tier elements should be skipped over when labelling a state. In doing so, the algorithm will be able to find all of the necessary states as well as the correct landing state for each transition in the target OTSL$_k$ transducer. This modification of the OSLFIA is incorporated into the function `build_fst`, which is detailed in Algorithm 1. While this constitutes one important aspect of learning OTSL$_k$ functions, it is nonetheless a major challenge to determine the actual contents of Θ without *a priori* knowledge. Although inducing a k-tier for any value of k remains as an open problem, in the following section we describe how his can be done when $k = 2$.

5.3 Learning the contents of a 2-tier

Having shown that the OSLFIA can be modified to learn an OTSL$_k$ function f once Θ (the tier) is known, we now describe our approach to inducing Θ itself when $k = 2$. After this is done, Θ can simply be fed into the `build_fst` function in order to produce an OTSL$_2$ transducer that represents f.

The first step toward learning the contents of a 2-tier is to gain as much information as possible

```
Function build_fst (S, Θ, k):
    $C \leftarrow \{q_0, q_f\}$ with $q_0, q_f \notin \Theta^{\leq k-1}$;
    $s \leftarrow \texttt{lcp}(\{y | (x, y) \in S\})$;
    $q \leftarrow \texttt{Suff}_{\Theta}^{k-1}(s)$;
    $\texttt{Earliest}(q) \leftarrow \rtimes$;
    $\texttt{Out}(q) \leftarrow s$;
    $\delta \leftarrow \{(q_0, \rtimes, s, q)\}$;
    $R \leftarrow \{q\}$;
    while $R \neq \emptyset$ do
        $q \leftarrow \texttt{First}(R)$;
        $s \leftarrow \texttt{Earliest}(q)$;
        for all $a \in \Sigma$ in alphabetical order do
            if $\exists (w, u) \in S, x \in \Sigma^*$ s.t. $w = sax$ then
                $v \leftarrow \texttt{lcp}(\{y | \exists x, (sax, y) \in S\})$;
                $r \leftarrow \texttt{Suff}_{\Theta}^{k-1}(v)$;
                $\delta \leftarrow \delta \cup \{(q, a, \texttt{Out}(q)^{-1} \cdot v, r)\}$;
                if $r \notin R \cup C$ then
                    $R \leftarrow R \cup \{r\}$;
                    $\texttt{Earliest}(r) \leftarrow sa$;
                    $\texttt{Out}(r) \leftarrow v$;
        if $\exists u$ s.t. $(s, u) \in S$ then
            $\delta \leftarrow \delta \cup \{(q, \ltimes, \texttt{Out}(q)^{-1} \cdot u, q_f)\}$
        $R \leftarrow R - \{q\}$;
        $C \leftarrow C \cup \{q\}$;
    return $\langle C, q_0, q_f, \Sigma, \Delta, \delta \rangle$
```

Algorithm 1: Building an OTSL$_k$ transducer when given Θ

about the prefix function f^p corresponding to f, based only on the evidence provided in the training sample. To do this, the function `estimate_fp`, shown in Algorithm 2, goes through every string x that is the prefix of at least one input string in the training data, and for every $a \in \Sigma$, it checks whether xa is also a prefix of some input string. If this is the case, there is enough information to determine $f^p(x)$. The function `estimate_fp` will then add the pair (x, z) to the set P, where z is the longest common prefix of $f(w)$ for all $(w, f(w)) \in S$ such that x is a prefix of w. We note that this z will be equal to $f^p(x)$ provided that the training data come from a subsequential function, and so this technique may be useful for learning other types of functions as well.

We further note, however, that by using this strategy, `estimate_fp` is only guaranteed to produce all the pairs $(x, f^p(x))$ necessary to discover the tier for total functions.

```
Function estimate_fp(S):
    $P \leftarrow \emptyset$;
    $X \leftarrow \{x \mid x \in \texttt{Pref}^*(w)$, where $(w,u) \in S\}$;
    $Y \leftarrow \{x \in X \mid (\forall a \in \Sigma)[xa \in X]\}$;
    for each $y \in Y$ do
        $z \leftarrow \texttt{lcp}(\{u \mid (w,u) \in S$, where $y \in \texttt{Pref}^*(w)\})$;
        $P \leftarrow P \cup \{(y,z)\}$
    return $P$;
```

Algorithm 2: Prefix function estimation

This is because, when there is no pair with the shape $(\rtimes pax\ltimes, f(pax))$ in the training data, `estimate_fp` does not know whether this is accidental (i.e., due to the finite nature of the training data) or because the function is undefined for all inputs of the shape $\rtimes pax\ltimes$. While the ability to accommodate partial functions would have practical applications for learning from natural language data, at present we leave the task of extending `estimate_fp` in this way to future research.

The full learning algorithm, which we call the OTSL$_2$ Function Inference Algorithm (OTSL2FIA) is shown in Algorithm 3. We assume that Σ and Δ are fixed and not part of the input to the learning problem (and that $k = 2$). Given a finite sample of training data, it first estimates the relevant prefix function with the set P, as described above, and begins with the hypothesis that $\Theta = \Delta$ (i.e., that all members of the output alphabet are on the target tier). Then, for each $a \in \Theta$, it looks through P for any evidence that a needs to be removed from Θ. To do this, it builds an auxiliary set `Match` that contains every $(p, f^p) \in P$ for which $\texttt{Suff}^1_\Theta(f^p(p)) = a$ under the current hypothesis for Θ. For each $x \in \Sigma \cup \{\ltimes\}$, it then checks whether $\texttt{cont}_f(x,p)$ is the same for all $(p,q) \in \texttt{Match}$. If this is the case, a is added to the set `Keep`. However, if there is more than one value found for the contribution of some $x \in \Sigma \cup \{\ltimes\}$, it will instead remove a from Θ, since it cannot possibly be a member of the target 2-tier. If at any point some symbol gets removed from Θ, the set `Keep` is immediately emptied. This portion of the algorithm will run until every a in the current hypothesis for Θ gets added to the set `Keep`, in which case it knows it has found the canonical 2-tier of the target function.

Once the OTSL2FIA converges on the canonical 2-tier Θ, the final step is simply to feed Θ and the sample S into the function `build_fst` shown above in Algorithm 1 (further specifying that $k = 2$). Under the assumption that the training sample contains the appropriate evidence, as described in the following section, this will produce an OTSL$_2$ transducer which represents the target OTSL$_2$ function.

```
Data: Sample $S \subset \{\rtimes\}\Sigma^*\{\ltimes\} \times \Delta^*$
Result: An OTSL$_2$ transducer
        $\mathcal{T} = \langle C, q_0, q_f, \Sigma, \Delta, \delta\rangle$
$P \leftarrow$ estimate_fp($S$);
$\Theta \leftarrow \Delta$;
Keep $\leftarrow \emptyset$;
while Keep $\neq \Theta$ do
    for each $a \in \Theta$ do
        Match $\leftarrow \{(p,q) \in P \mid \texttt{Suff}^1_\Theta(q) = a\}$;
        for each $\sigma \in \Sigma$ do
            $C_{\sigma,a} \leftarrow \{q^{-1} \cdot y \mid (p,q) \in \texttt{Match} \wedge (p\sigma, y) \in P\}$;
            if $card(C_{\sigma,a}) > 1$ then
                $\Theta \leftarrow \Theta - \{a\}$;
                Keep $\leftarrow \emptyset$;
        $C_{\ltimes,a} \leftarrow \{q^{-1} \cdot y \mid (p,q) \in \texttt{Match} \wedge (p,y) \in S\}$;
        if $card(C_{\ltimes,a}) > 1$ then
            $\Theta \leftarrow \Theta - \{a\}$;
            Keep $\leftarrow \emptyset$;
        if $a \in \Theta$ then
            Keep $\leftarrow$ Keep $\cup \{a\}$
$\mathcal{T} \leftarrow$ build_fst($S$, $\Theta$, 2);
return $\mathcal{T}$
```

Algorithm 3: OTSL2FIA

5.4 Theoretical results

Here we establish several theoretical results, which culminate in the theorem that the OTSL2FIA identifies the class of total OTSL$_2$ functions in polynomial time and data.

In what follows, we let f be the target OTSL$_2$ function, $\Theta_\diamond$ be its canonical 2-tier, and $\mathcal{T}^\diamond = \langle Q_\diamond, q_0, q_f, \Sigma, \Delta, \delta_\diamond\rangle$ be its target transducer as defined by Theorem 2. We furthermore let Θ be the OTSL2FIA's final tier hypothesis, and $\mathcal{T} = \langle Q, q_0, q_f, \Sigma, \Delta, \delta\rangle$ be the transducer that is constructed on the input.

Lemma 6. (Polynomial time) *For any input sample S, the OTSL2FIA produces $\mathcal{T}$ in time polynomial in the size of S.*

Proof. Let $n = \sum_{(w,u)\in S} |w|$, $m = max\{|u| : (w,u) \in S\}$, $p = max\{|w| : (w,u) \in S\}$, and $s = \texttt{card}(S)$. We note that these are all linear in the size of the sample.

The OTSL2FIA starts by calling `estimate_fp`. This function first determines all of the input prefixes present in S, which takes n steps. Then `estimate_fp` checks, for each prefix x and all $a \in \Sigma$, whether xa is also an input prefix in S. There are at most sm prefixes in S, so this takes at most $\texttt{card}(\Sigma) \cdot (sm)n$ steps. Finally, for a subset of the input prefixes, `estimate_fp` determines $\texttt{lcp}(\{u \mid (w,u) \text{ s.t. } x \in \texttt{Pref}^*(w)\})$, which with an appropriate data structure (for instance a prefix tree) can be done in nm steps. The overall computation time of `estimate_fp` is thus $\mathcal{O}(n + (sm)n + (sm)(nm))$, which is quartic in the size of the learning sample.

The portion of the OTSL2FIA that determines the tier is now run. After i elements have been removed from Θ, the combined *while/for* loop can run up to $|\Delta| - i$ times, and can only remove up to $|\Delta|$ items, so the loop will be used fewer than $|\Delta|^2$ times, which is a constant. This main loop first gathers all $(w,u) \in P$ that meet a certain criterion into the set `Match`, which can be done in $\texttt{card}(P)m = (sm)m = sm^2$ steps. Next, the main loop enters a *for* loop that is used $\texttt{card}(\Sigma)$ times (a constant) and which attempts to calculate the contribution of $\sigma \in \Sigma$ using each $(w,u) \in \texttt{Match}$ if it can find $(w\sigma, v) \in P$. We note that $\texttt{card}(\texttt{Match})$ will be at most sm, that finding $(w\sigma, v) \in P$ takes at most smp steps, and that calculating the contribution takes at most m steps. The main loop then attempts to calculate the contribution of $\ltimes$ using each $(w,u) \in \texttt{Match}$ if it can find $(w,v) \in S$. We note that finding $(w,v) \in S$ takes at most n steps, and that calculating the contribution takes at most m steps. The overall computation time of this portion of the algorithm is thus $\mathcal{O}(sm^2 + sm(smp+m) + sm(n+m))$, which is quintic in the size of the learning sample.

Finally, the OTSL2FIA feeds Θ and S to the function `build_fst`. As noted above, this fuction incorporates a simple modification to the state-labelling process in Chandlee et al.'s (2015) OSLFIA. While this change allows it to build an OTSL transducer once the tier is known, it does not affect computation time. This final step of the OTSL2FIA therefore runs in time quadratic in the size of the learning sample (for OSLFIA time complexity proofs, see Chandlee et al., 2015). Since each portion of the OTSL2FIA runs in time polynomial in the size of the sample, with the highest complexity being quintic, the overall computation time of the algorithm is therefore polynomial in the size of the learning sample.

□

The remaining lemmas of this section will show that for each total OTSL$_2$ function f, there is a finite kernel of data consistent with f that is a characteristic sample for OTSL2FIA, which we call an OTSL2FIA seed.

Definition 18. (Seed) *Given $\mathcal{T}^\diamond$, a sample S contains a seed if:*

1. *For all $q \in Q_\diamond$, $(\rtimes w_q \ltimes, f(w_q)) \in S$.*
2. *For all $(q, a, u, q') \in \delta_\diamond$ such that $q' \neq q_f$ and $a \in \{\rtimes\} \cup \Sigma$, and for all pairs $b, c \in \Sigma$:*
   (a) $(\rtimes w_q a \ltimes, f(w_q a)) \in S$
   (b) $(\rtimes w_q ab \ltimes, f(w_q ab)) \in S$
   (c) $(\rtimes w_q abcx \ltimes, f(w_q abcx)) \in S$*, where* $x \in \Sigma^*$

Lemma 7. *If a learning sample S contains a seed, then the OTSL2FIA can determine $\texttt{cont}_f(x, w)$ for all $w \in \Sigma^*$ and all $x \in \Sigma \cup \{\ltimes\}$.*

Proof. Let us start with some string $\rtimes wby \ltimes$ such that $b \in \Sigma$ and $w, y \in \Sigma^*$. Since $\mathcal{T}^\diamond$ is OTSL$_2$, it will be in the state corresponding to $\texttt{Suff}^1_{\Theta_\diamond}(f^p(w))$ immediately prior to reading b when processing $\rtimes wby\ltimes$. Let us call this state q'. The most recent transition that $\mathcal{T}^\diamond$ will have traversed is (q, a, u, q'). The target function is OTSL$_2$, and so it is the case that either $w = w_q a$ or else can be replaced thereby since $\texttt{Suff}^1_{\Theta_\diamond}(f^p(w)) = \texttt{Suff}^1_{\Theta_\diamond}(f^p(w_q a))$ and therefore $\texttt{cont}_f(b, w) = \texttt{cont}_f(b, w_q a)$.

Now let us start with some string $\rtimes w \ltimes$ such that $w \in \Sigma^*$. When $\mathcal{T}^\diamond$ reads $\rtimes w\ltimes$, it will be in the state corresponding to $\texttt{Suff}^1_{\Theta_\diamond}(f^p(w))$ immediately prior to reading $\ltimes$. Let us call this state q'. The most recent transition that $\mathcal{T}^\diamond$ will have traversed is (q, a, u, q'). The target function is OTSL$_2$, and so it is the case that either $w = w_q a$ or else can be replaced thereby since $\texttt{Suff}^1_{\Theta_\diamond}(f^p(w)) = \texttt{Suff}^1_{\Theta_\diamond}(f^p(w_q a))$ and therefore $\texttt{cont}_f(\ltimes, w) = \texttt{cont}_f(\ltimes, w_q a)$.

Now recall that $f^p(w) = \texttt{lcp}(\{u \mid u = f(wx) \wedge x \in \Sigma^*\})$. We do not actually need the entirety of this infinite set to determine $f^p(w)$, it is sufficient to use a set containing $f(w)$ and one $f(wax)$ for each $a \in \Sigma$ where $x \in \Sigma^*$ because every $x \in \Sigma^*$ is either λ or begins with some $a \in \Sigma$. Let us call such a set a *support* for determining $f^p(w)$. The function estimate_fp takes every prefix p present in S and checks whether a support for determining $f^p(p)$ exists in S. Then, if a support exists, estimate_fp adds (p, q) to the set P, where $q = \texttt{lcp}(\{u \mid (w, u) \in S \wedge p \in \texttt{Pref}^*\})$, that is $q = f^p(p)$.

By the definition of the seed, for every transition $(q, a, u, q') \in \delta_\diamond$ such that $q' \neq q_f$, the learner will see $\rtimes w_q a \ltimes$, $\rtimes w_q ab \ltimes$ for all $b \in \Sigma$, and at least one input string $\rtimes w_q abcx \ltimes$ for all pairs $b, c \in \Sigma$. We therefore know that for any pair of input strings $\rtimes w \ltimes$ and $\rtimes wby \ltimes$ in the domain of f such that $b \in \Sigma$ and $x \in \Sigma^*$, the seed will contain all the input strings necessary to build supports for determining $f^p(w_q a)$ and $f^p(w_q ab)$ such that $\texttt{tails}_f(w_q a) = \texttt{tails}_f(w)$.

By Remark 1, we know that $\texttt{cont}_f(b, w) = f^p(w)^{-1} \cdot f^p(wb)$ for all $b \in \Sigma$ and $w \in \Sigma^*$. It is therefore the case that for every $\rtimes wby \ltimes$, the algorithm can determine $\texttt{cont}_f(b, w_q a) = f^p(w_q a)^{-1} \cdot f^p(w_q ab) = \texttt{cont}_f(b, w)$. Also by Remark 1, we know that $\texttt{cont}_f(\ltimes, w) = f^p(w)^{-1} \cdot f(w)$ for all $w \in \Sigma^*$. It is therefore the case that for every $\rtimes w \ltimes$, the algorithm can determine $\texttt{cont}_f(\ltimes, w_q a) = f^p(w_q a)^{-1} \cdot f(w_q a) = \texttt{cont}_f(\ltimes, w)$. $\square$

Lemma 8. (Tier convergence) *If a learning sample S contains a seed, then $\Theta = \Theta_\diamond$.*

Proof. The OTSL2FIA starts with $\Theta = \Delta$, and so either $\Theta = \Theta_\diamond$ already, or else $\Theta \supset \Theta_\diamond$.

We know from Lemma 5 that if $\Theta \supset \Theta_\diamond$, there will exist a pair of input strings w_1 and w_2 in the domain of f such that $\texttt{Suff}^1_\Theta(f^p(w_1)) = \texttt{Suff}^1_\Theta(f^p(w_2)) = a$ for some $a \in (\Theta - \Theta_\diamond)$ and $\texttt{cont}_f(x, w_1) \neq \texttt{cont}_f(x, w_2)$ for some $x \in \Sigma \cup \{\ltimes\}$. We know from Lemma 7 that every $\rtimes w \ltimes$ in the domain of f has at least one corresponding $(w', f^p(w')) \in P$ and at least one corresponding $(w'a, f^p(w'a)) \in P$ for each $a \in \Sigma$, where $\texttt{Suff}^1_\Theta(f^p(w)) = \texttt{Suff}^1_\Theta(f^p(w'))$ and so $\texttt{cont}_f(x, w) = \texttt{cont}_f(x, w')$ for all $x \in \Sigma \cup \{\ltimes\}$.The algorithm will thus be able to calculate and check all the relevant contributions necessary to flag and remove at least one $a \in (\Theta - \Theta^\diamond)$ when $\Theta \supset \Theta^\diamond$.

Conversely, there will be no pair of input strings w_3 and w_4 in the domain of f such that $\texttt{cont}_f(x, w_3) \neq \texttt{cont}_f(x, w_4)$ for some $x \in \Sigma \cup \{\ltimes\}$ when $\texttt{Suff}^1_\Theta(f^p(w_3)) = \texttt{Suff}^1_\Theta(f^p(w_3)) = c$ for some $c \in \Theta^\diamond$. When $\Theta = \Theta_\diamond$, then, the algorithm will add all $a \in \Theta$ to Keep and pass Keep $= \Theta = \Theta_\diamond$ to the build_fst function. $\square$

Lemma 9. (Transducer convergence) *If a learning sample S contains a seed then* $(q_0, \rtimes w, u, r) \in \delta^* \Longleftrightarrow (q_0, \rtimes w, u, r) \in \delta^*_\diamond$.

Lemma 10. (Characteristic Sample) *Any learning sample containing a seed is a characteristic sample for the OTSL2FIA.*

We do not include the proofs of Lemmas 9 and 10 here, as they are a trivial extension of analogous proofs in Chandlee et al. (2015, Lemmas 7 and 8). Again, the generalization requires only that each instance of $\texttt{Suff}^{k-1}$ be replaced by $\texttt{Suff}^{k-1}_\Theta$ in order for the proofs hold for any OTSL$_k$ transducer, provided that the target tier is passed to build_fst. We further note that when the target transducer is one that computes a total OTSL$_2$ function with its canonical tier, a seed (as defined in Definition 18 above) is a superset of that required by Chandlee et al. (2015, Definition 11).

Lemma 11. (Polynomial data) *Given an OTSL$_2$ transducer $\mathcal{T}^\diamond$, there exists a seed for the OTSL2FIA that is of size polynomial in the size of $\mathcal{T}^\diamond$.*

Proof. Let $\mathcal{T}^\diamond = \langle Q_\diamond, q_0, q_f, \Sigma, \Delta, \delta_\diamond \rangle$. For item 1 in Definition 18 there are $\texttt{card}(Q_\diamond)$ corresponding input-output pairs $(w_q, f(w_q))$ in a seed. For each of these pairs, it is the case that $| \rtimes w_q \ltimes | \leq \texttt{card}(Q_\diamond)$ and it is the case that $|f(w_q)| \leq \sum_{(q,a,u,q') \in \delta_\diamond} |u|$. We denote the latter quantity with $x_\diamond = \sum_{(q,a,u,q') \in \delta_\diamond} |u|$ and note that $x_\diamond = \mathcal{O}(|\mathcal{T}^\diamond|)$. The overall length of the inputs in the portion of the seed contributed by item 1 is thus in $\mathcal{O}(\texttt{card}(Q_\diamond)^2)$. The overall length of the outputs in the portion of the seed contributed by item 1 is thus in $\mathcal{O}(\texttt{card}(Q_\diamond) \cdot x_\diamond)$. We note that both of these are quadratic in the size of $\mathcal{T}^\diamond$.

For items 2a, 2b, and 2c in Definition 18, there are respectively 1, $\texttt{card}(\Sigma)$, and $\texttt{card}(\Sigma)^2$ corresponding input-output pairs per transition $(q, a, u, q') \in \delta_\diamond$. Factoring out the constant $\texttt{card}(\Sigma)$ gives us $3 \cdot \texttt{card}(\delta_\diamond)$ pairs. For the pairs contributed by item 2c, we can restrict ourselves to pairs $(\rtimes w_q abc \ltimes, f(w_q abc))$, since f is a total function. For each pair, we have $| \rtimes w \ltimes | \leq \texttt{card}(Q_\diamond) + 3$ and $|f(w)| \leq \sum_{(q,a,u,q') \in \delta_\diamond} |u| + 3m$, where $m = max\{|u| : (q,a,u,q') \in \delta_\diamond\}$. With this last quantity denoted $y_\diamond$, we note that $y_\diamond = \mathcal{O}(|\mathcal{T}^\diamond|)$. The overall length of the inputs in the portion of the seed contributed by item 2 is therefore in $\mathcal{O}((3 \cdot \texttt{card}(\delta_\diamond))(\texttt{card}(Q_\diamond) + 3) = \mathcal{O}(\texttt{card}(\delta_\diamond) \cdot \texttt{card}(Q_\diamond) + \texttt{card}(\delta_\diamond))$, and the overall length of the outputs in the portion of the seed contributed by item 2 is in $\mathcal{O}(3 \cdot \texttt{card}(\delta_\diamond) \cdot y_\diamond)$. Both of these are quadratic in the size of $\mathcal{T}^\diamond$.

Altogether, then, the size of the seed is quadratic in the size of the target transducer. □

Theorem 3. *The OTSL2FIA identifies the OTSL*$_2$ *functions in polynomial time and data.*

Proof. Immediate from Lemmas 6, 8, 9, 10, and 11. □

6 Discussion

The OTSL functions introduced in this paper are capable of modelling many of the attested long-distance phonological processes. These processes can be assimilatory like sibilant harmony in Samala (see Section 1), but can also be dissimilatory. For example, Georgian exhibits a pattern of liquid dissimilation, in which /r/ surfaces as [l] when preceded at any distance by another [r] (e.g., /aprik'-uri/ → [aprik'uli] 'African'; Odden, 1994). Interestingly, the dissimilation does not occur if there is an intervening [l] (e.g., /kartl-uri/ → [kartluri] 'Kartvelian'). The OTSL functions are fully capable of representing such *blocking* effects, as shown in Figure 2. To avoid cluttering the figure, we omit the final state and all of its incoming transitions (which would be labelled $\ltimes$:λ).

It is worth pointing out that the processes in Samala and Georgian apply in opposite directions. In Samala, the trigger is the *rightmost* sibilant, whereas in Georgian it is the *leftmost* liquid. This distinction can be captured by assuming that input strings are read from left-to-right in the Georgian case (i.e., the process is progressive), but from right-to-left in the Samala case (i.e., the process is regressive). The direction of reading, then, divides the OTSL functions into two overlapping but distinct classes which we call L-OTSL (which read from the left) and R-OTSL (which read from the right), following Heinz and Lai (2013) and Chandlee et al. (2015) who make the same distinction

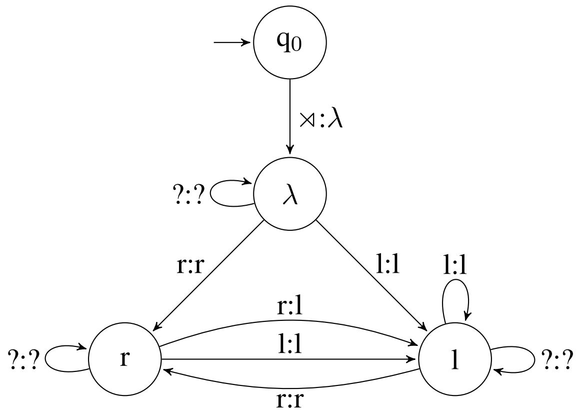

Figure 2: An OTSL$_2$ transducer that models unbounded liquid dissimilation with blocking, where ? represents any symbol that is not [l] or [r].

for the subsequential and OSL functions, respectively.

As mentioned above, the OTSL2FIA outlined in Section 5 only succeeds in learning total functions and is designed specifically to learn OTSL$_2$ functions. The algorithm exploits the fact that the largest possible 2-tier for an OTSL$_2$ function f is a superset of every other 2-tier for f, and will accordingly never run the risk of removing an element that would need to be subsequently re-added to the tier. However, it is not clear that this strategy will succeed for higher values k, which may be needed to model certain types of patterns. For example, a reviewer raises the complex case of retroflexion harmony targetting /n/ in Sanskrit (also known as *nati*) as one such pattern. A formal analysis provided by Graf and Mayer (2018) uses a class of stringsets that they call Input-Output Tier-based Strictly Local (IO-TSL). IO-TSL formal languages are like TSL languages except that input symbols are projected onto the tier based on (i) the surrounding context of input symbols and (ii) the symbols that precede it on the tier that has been projected so far. Under this analysis, Sanskrit n-retroflexion requires $k = 3$ on the projected tier.

Finally, while the OTSL class can model long-distance processes, it can only do so when no more than a single tier is required. That is, a language that simultaneously exhibits patterns of, e.g., sibilant harmony and liquid dissimilation would not be OTSL for any value of k. Further exploration of these issues will allow us to better understand the computational properties of phonological transformations and to establish a boundary of complexity that is both necessary and sufficient for capturing the full range of possible phonological systems.

7 Conclusion

This paper has provided both a language-theoretic and an automata-theoretic characterization of the OTSL class of functions, which is relevant for modelling long-distance phonological processes as string-to-string transformations. We further demonstrated that by generalizing previous research on OSL functions to the OTSL class, any OTSL_k function can be learned once the tier is known. Finally, we introduced an algorithm for efficiently learning any total OTSL_2 function from positive data, even when a relevant tier is not given *a priori*. To our knowledge, this is the first algorithm to accomplish this for input-output mappings rather than phonotactics. In future research, we aim to extend this result in multiple ways: to partial functions, to any value of k, and to processes requiring multiple tiers.

Acknowledgements

Special thanks to Jane Chandlee, participants of the Third Subregular Workshop at Stony Brook University, and three anonymous reviewers. This research was supported by the Social Sciences and Humanities Research Council of Canada.

References

Richard B. Applegate. 1972. *Ineseño Chumash grammar*. Doctoral dissertation, University of California, Berkeley.

Jane Chandlee. 2014. *Strictly Local phonological processes*. Ph.D. thesis, University of Delaware.

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*.

George N. Clements. 1980. *Vowel harmony in nonlinear generative phonology: an autosegmental model*. Indiana University Linguistics Club, Bloomington, IN.

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

E. Mark Gold. 1967. Language identification in the limit. *Information and Control*, 10:447–474.

John A. Goldsmith. 1990. *Autosegmental and metrical phonology*. Blackwell, Oxford.

Maria Gouskova and Gillian Gallagher. 2019. Inducing nonlocal constraints from baseline phonotactics. *Natural Language and Linguistic Theory*.

Thomas Graf and Connor Mayer. 2018. Sanskrit n-retroflexion is input-output tier-based strictly local. In *Proceedings of SIGMORPHON 2018*, pages 151–160.

Gunnar Ólafur Hansson. 2010. *Consonant harmony: long-distance interaction in phonology*. University of California Press, Berkeley, CA.

Bruce Hayes and Colin Wilson. 2008. A maximum entropy model of phonotactics and phonotactic learning. *Linguistic Inquiry*, 39:379–440.

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

Jeffrey Heinz, Chetan Rawal, and Herbert G. Tanner. 2011. Tier-based strictly local constraints for phonology. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 58–64, Portland, OR. Association for Computational Linguistics.

Adam Jardine, Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. 2014. Very efficient learning of structured classes of subsequential functions from positive data. In *Proceedings of the Twelfth International Conference on Grammatical Inference*, volume 34, pages 94–108.

Adam Jardine and Jeffrey Heinz. 2016. Learning tier-based strictly 2-local languages. *Transactions of the Association for Computational Linguistics*, 4:87–98.

Adam Jardine and Kevin McMullin. 2017. Efficient learning of Tier-based Strictly k-Local languages. In *Proceedings of Language and Automata Theory and Applications, 11th International Conference*, Lecture Notes in Computer Science. Springer.

Aaron Kaplan. 2008. *Noniterativity is an emergent property of grammar*. Ph.D. thesis, University of California Santa Cruz.

Kevin McMullin. 2016. *Tier-based locality in long-distance phonotactics: learnability and typology*. Ph.D. thesis, University of British Columbia.

David Odden. 1994. Adjacency parameters in phonology. *Language*, 70:289–330.

José Oncina and Pedro García. 1991. Inductive learning of subsequential functions. Techical Report DSIC II-34, University Politećnia de Valencia.

José Oncina, Pedro García, and Enrique Vidal. 1993. Learning subsequential transducers for pattern recognition tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:448–458.

James Rogers and Geoffrey K. Pullum. 2011. Aural pattern recognition experiments and the subregular hierarchy. *Journal of Logic, Language and Information*, 20(3):329–342.