Learning Input Strictly Local Functions From Their Composition

Wenyue Hua WENYUE.HUA@RUTGERS.EDU

Adam Jardine ADAM.JARDINE@RUTGERS.EDU

Editors: Jane Chandlee, Rémi Eyraud, Jeffrey Heinz, Adam Jardine, and Menno van Zaanen

Abstract

This paper studies the learning of two functions given positive samples of their composition, motivated by an empirical problem in natural language phonology. Empirically relevant conditions under which this is possible are identified and a provably correct algorithm is given that can *semi-strongly* identify the two functions in polynomial time and data. In order to clearly illustrate the learning problem and related concepts, we focus on a simple subset of *input strictly 2-local functions*. But we further argue that the general learning procedure we propose can be extended to more general classes of functions.

Keywords: subsequential functions, input strictly local functions, identification with polynomial bounds

1. Introduction

Inspired by a fundamental learning problem in natural language phonology, this paper studies the problem of learning a pair of subsequential functions from positive examples of their composition. The subsequential functions (Schützenberger, 1977; Mohri, 1997) are exactly those that have a finite set of *tail functions* (analagous to the notion of good tails for regular languages), which can be used to efficiently identify a subsequential function in the limit from positive data (Oncina et al., 1993). Thus, while it is obviously impossible to exactly identify any arbitrary two functions from their composition (consider cases in which the first function entirely obscures the inputs of the second), the structure of the subsequential class allows us to study more specific cases in which the tail functions of two subsequential functions can be teased apart from their composition.

In particular, we focus on the case in which the input to a *simplex input-strictly* 2*-local (simplex ISL$_2$) function* is obscured by a second letter-to-string homomorphism. The simplex ISL$_2$ functions are a novel restriction on the *input-strictly local* (ISL) functions (Vaysse, 1986; Chandlee, 2014), which are those in which any change to the input string is necessarily based on a sequence of symbols of bounded length preceding that change. Simplex ISL$_2$ functions are ISL$_2$ functions restricted to modifying only one input symbol in a particular context. While a restricted class, it has the interesting property that the tail functions of a simplex ISL$_2$ function, while are defined based on the input to the function, can be determined from the output of the function. We give an algorithm SI$_2$DLA[1] that, given positive examples of the composition of a letter-to-string homomorphism f and a

1. An implementation in Python along with the data sets in this paper can be found on https://github.com/rucll/ur_learning.

simplex ISL_2 function g, can strongly identify g and identify some homomorphism f' such that $g \circ f' = g \circ f$, as long as f sufficiently exemplifies the tail functions of g.

While this is a specific case, it is of interest for several reasons. From a theoretical standpoint, to our knowledge, this is the first work studying the induction of multiple functions from their composition. In doing so, we identify a technique which can be extended to other sub-classes of the subsequential functions. As we discuss in §5.2, work on other classes can proceed by extending any of the many restrictions on the classes discussed here. Furthermore, it allows for the introduction of an intermediate notion between weak and strong learning, which we name *semi-strong* learning.

From an empirical perspective, this result has direct application to natural language phonology. Generative phonology posits that the phonological content of morphemes—meaningful pieces of words—are stored with abstract mental representations. When pronounced, these representations are subject to phonological *processes* that change sounds based on their context. We can model these two parts of phonology as a homomorphism from morphemes to their abstract representations, and a function representing phonological processes from abstract representations to their pronunciations. How words are pronounced is thus the composition of these two functions. However, children acquiring their language only have access to data from this composition. A major learning problem in theoretical phonology, then, is how children learn these abstract representations *and* processes from their composition. Thus, as explicated in §5.1, this problem can be identified with the learning goal explored in this paper.

The paper is organized as follows. Section 2 introduces relevant notations and concepts. Section 3 establishes the learning problem mathematically and also proves useful properties of them. Section 4 presents the algorithm, demonstrates how it works, and proves the data complexity, time complexity and the correctness of the algorithm. Section 5 ends the paper and discusses directions of future work.

2. Preliminaries

2.1. Strings

Let an alphabet Σ be a finite set of symbols and Σ^* be all strings over Σ, including the empty string λ. In this paper, we assume that $|\Sigma| \geq 3$. We denote by $|w|$ the length of w; note that $|\lambda| = 0$. For strings w and v, both wv and $w \cdot v$ denote their concatenation. Also if $w = uv$ then we write $u^{-1} \cdot w = v$ and $w \cdot v^{-1} = u$. A string u is a *prefix* of w iff $w = uv$ for some third string v; likewise u is a *suffix* of w iff $w = vu$ for some third string v. For $w \in \Sigma^*$, let $\texttt{prefs}(w)$ denote the prefixes of w, *i.e.*, $\{u \in \Sigma^* \mid u \text{ is a prefix of } w\}$; and $\texttt{suffs}(w)$ the suffixes of w. The *k-prefix* of w is $\texttt{pref}_k(w) = u$ (respectively $\texttt{suff}_k(w) = u$) s.t. u is a prefix of w (resp. suffix of w) and $|u| = k$ if $|w| > k$, otherwise $u = w$. We also extend these to sets of strings; e.g. for $L \subseteq \Sigma^*$, $\texttt{suff}_k(L) = \cup_{w \in L}\{\texttt{suff}_k(w)\}$. Let $w \wedge v$ denote the *longest common prefix* (lcp) of w and v; *i.e.*, a unique $u \in \texttt{prefs}(w) \cap \texttt{prefs}(v)$ s.t. for any other $u' \in \texttt{prefs}(w) \cap \texttt{prefs}(v)$, $|u'| \leq |u|$.

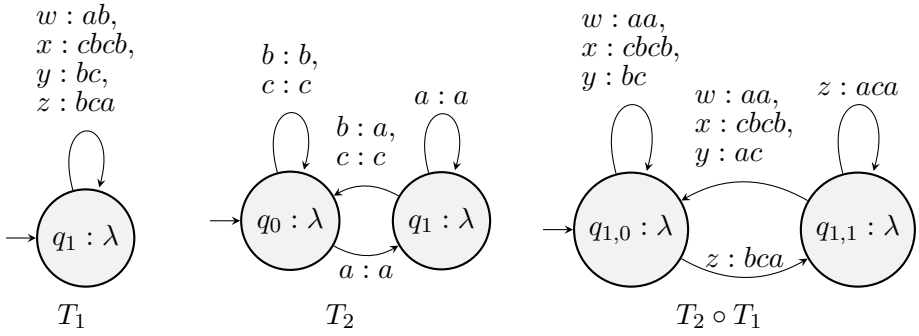

Figure 1: Two onward SFSTs and their (minimized) composition.

2.2. Functions on strings

We consider functions on strings of the form $f : \Sigma^* \to \Gamma^*$ from an input alphabet Σ to a (potentially distinct) output alphabet Γ. In this paper, we only consider total functions. Let $\texttt{dom}(f)$ and $\texttt{ran}(f)$ denote the domain and range of f, respectively. For f, let $f^p(w)$ denote the *common output* of w which is defined as: $f^p(w) \stackrel{\text{def}}{=} \bigwedge_{wu \in \texttt{dom}(f)} f(wu)$. For any $w \in \Sigma^*$, by $f_w : \Sigma^* \to \Gamma^*$ we denote the *tail function of* w, which is defined as $f_w(u) \stackrel{\text{def}}{=} f^p(w)^{-1} \cdot f(wu)$. That is, $f_w(u)$ returns the contribution of u to the output of f following w. Finally, based on the definition of f^p, define another function $f^p_w(u) \stackrel{\text{def}}{=} \bigwedge_{wu \cdot v \in \texttt{dom}(f)} (f_w(uv))$. Note this is just the common output of wu minus the common output of w.

Any function f is *(left-)subsequential* iff the set of unique tail functions—that is, $\{f_w \mid w \in \Sigma^*\}$—is finite (Schützenberger, 1977; Mohri, 1997). We often use a bold capital letter, e.g., $\mathbf{F}$ to denote the set of tail functions of f.

2.3. Subsequential Finite-State Transducers

Subsequential functions can be described by *subsequential finite-state transducers* (SFSTs) defined as deterministic FSTs with outputs on the states. An SFST T is a tuple $\langle Q, q_0, Q_\ltimes, \delta, \omega, \iota \rangle$ with $Q_\ltimes \subseteq Q$ being final states, $\delta : Q \times \Sigma \to Q$ the transition function, $\omega : Q \times \Sigma \to \Gamma^*$ the output function, and $\iota : Q_\ltimes \to \Sigma^*$ the state output function.

To define the function described by an SFST, we extend δ and ω to functions δ^* and ω^* on $Q \times \Sigma^*$ in the usual way as follows. For $w = \lambda$ and any state $q \in Q$, $\delta^*(q, \lambda) = q$; otherwise $\delta^*(q, w\sigma) = r$ iff $\delta^*(q, w) = q'$ and $\delta(q', \sigma) = r$. Similarly, for $w = \lambda$ and any $q \in Q$, $\omega^*(q, \lambda) = \lambda$; otherwise $\omega^*(q, w\sigma) = uv$ iff $\delta^*(q, w) = q'$, $\omega^*(q, w) = u$, and $\omega(q', \sigma) = v$. Then for an SFST T, the function $f(T) : \Sigma^* \to \Gamma^*$ is defined as follows: $f(w)$ is defined iff $\delta^*(q_0, w) \in Q_\ltimes$, and if $f(w)$ is defined then $f(w) = \omega^*(q_0, w) \cdot \iota(\delta^*(q_0, w))$.

The states in an SFST for f represent tail functions of f, *i.e.* for each $q \in Q$, $f_q = f_w$ for any prefix w that reaches q (*i.e.*, that $\delta^*(q_0, w) = q$). Thus, the cardinality of Q must at least equal the cardinality of $\mathbf{F}$. We say that an SFST for a subsequential function f is *onward* iff for any $q \in Q, \sigma \in \Sigma$, $\omega(q, \sigma) = f^p_q(\sigma)$. A *canonical* SFST T for f is the onward SFST with a one-to-one correspondence between states in T and tail functions in f.

Example 1 *Two onward SFSTs are given in the standard directed graph representation in T_1, T_2 of Fig. 1: T_1 describes a letter-to-string homomorphism $f = f(T_1)$ from $\{w, x, y, z\}^*$ to $\{a, b, c\}^*$ in which w, x, y, and z are mapped to ab, $cbcb$, bc, and bca. Thus, for example, $f(zyw) = bcabcab$. The reader can check that for all $\sigma \in \{w, x, y, z\}^*$, $\omega(q_1, \sigma) = f^p_\lambda(\sigma)$; Similarly, T_2 represents the onward SFST for a function over $\{a, b, c\}^*$ that maps any b immediately following an input a to a. That is, e.g., for $g = f(T_2)$, $g(bcabcab) = bcaacaa$.*

We can take the composition of two SFSTs using a construction based on Lothaire (2005); we give this in full in Appendix A.1.

Example 2 *The composition of T_2 and T_1 is given in the right figure in Fig. 1. Let $h = f(T_2 \circ T_1)$; for example, $h(zyw) = bcaacaa = g \circ f(zyw)$.*

In our figures and examples, we assume the composed SFSTs have also been minimized (see, e.g., (Sakarovitch, 2009) for minimization techniques). This construction also allows us to make the following statement about $g \circ f$.

Lemma 1 *Let f and g be two subsequential functions and $h = g \circ f$. Then for the set $\mathbf{H}$ of tail functions of h, $|\mathbf{H}| \leq |\mathbf{F} \times \mathbf{G}|$.*

This lemma follows immediately from the state structure of the construction of $T_f \circ T_g$, where T_f and T_g are the canonical SFSTs for f and g, respectively.

3. Learning paradigm

Informally, the learning problem is about inducing two functions f and g when the input-output pairs of their composition h are given. Formally, we pose the learning problem in terms of learning in the limit from positive data (Gold, 1967) in polynomial time and data (de la Higuera, 2010). The goal is to identify for a class of functions where there is an algorithm that returns a representation of that function in polynomial time, given a sufficient positive sample of any function in that class. Further, this sufficient positive sample must be in polynomial size of the given representation of the function.

We consider a class $\mathbb{H}$ of functions that is *represented* by a class of finite representations $\mathbb{R}$; that is, there is a total, surjective function $N : \mathbb{R} \to \mathbb{H}$ from representations in $\mathbb{R}$ to functions in $\mathbb{H}$. The notion of a sample and a learning algorithm for it is defined as follows.

Definition 2 (Sample) *For some function $h \in \mathbb{H}$, a* sample *D of h is a partial, finite function that is* consistent *with h; that is, whenever $D(w) = v$, $h(w) = v$. The* size *of D is the sum of the size of its composite pairs; that is, $|D| = \sum_{w \in \text{preim}(D)} |w| + |D(w)|$, where* preim$(D)$ *denotes the pre-image of D .*

Definition 3 ($(\mathbb{H}, \mathbb{R})$-learning algorithm) *A* $(\mathbb{H}, \mathbb{R})$-learning algorithm *is a program $\mathcal{A}$ that takes as its input a sample from a function $h \in \mathbb{H}$ and returns a representation $r \in \mathbb{R}$.*

In previous work on learning subsequential functions (Oncina et al., 1993; Jardine et al., 2014; Chandlee et al., 2015), the representations are single SFSTs, and so $N(r) \stackrel{\text{def}}{=} f(r)$.

Here, our class $\mathbb{R}$ consists of *pairs* of SFSTs (T_f, T_g), and our mapping N from representations to target functions is $N((T_f, T_g)) \stackrel{\text{def}}{=} f(T_g) \circ f(T_f)$. This adds additional logical possibilities to the usual notions of *weak* learning (*i.e.*, learning the target function) and *strong* learning (*i.e.*, learning the target representation) (Clark, 2014). The following intermediate notion applies specifically to the problem of decomposing functions.

Definition 4 (Semi-strong characteristic sample) *For an $\mathbb{R}$ comprised of pairs of subrepresentations, for a $(\mathbb{H}, \mathbb{R})$-learning algorithm $\mathcal{A}$, a sample C is a* semi-strong characteristic sample (CS) *of $(r_1, r_2) \in \mathbb{R}$ iff for all samples D for $N((r_1, r_2))$ s.t. $C \subseteq D$, $\mathcal{A}$ returns a representation (r_1', r_2') s.t. $N((r_1', r_2')) = N((r_1, r_2))$ and either $r_1' = r_1$ or $r_2' = r_2$.*

We can now define the learning goal.

Definition 5 (Semi-strong identification in the limit in polynomial time and data) *The class $\mathbb{H}$ is* semi-strongly identifiable in polynomial time and data (sSIPTD) *iff there exists an $(\mathbb{H}, \mathbb{R})$-learning algorithm $\mathcal{A}$ and two polynomials p and q such that for any semi-strong characteristic sample S for some $r \in \mathbb{R}$, $\mathcal{A}$ returns a hypothesis $r' \in \mathbb{R}$ in $\mathcal{O}(p(m))$ time and for each $r \in \mathbb{R}$ of size k, there exists a semi-strong CS of r for $\mathcal{A}$ of size at most $\mathcal{O}(q(k))$.*

The class $\mathbb{H}$ is sSIPTD indicates that for any function $h \in \mathbb{H}$ and an (r_1, r_2) such that $N((r_1, r_2)) = h$ (which is not necessarily unique), there is a polynomial-sized semi-strong CS such that a reasonably accurate representation (r_1', r_2') can be computed. Based on N, the function h can be inferred by $N((r_1', r_2'))$. The algorithm $\mathcal{A}$ generates the representation such that either $r_1' = r_1$ or $r_2' = r_2$. We do not aim at generating a representation that both $r_1' = r_1$ and $r_2' = r_2$ because such requirement imposes strong restrictions on the class $\mathbb{H}$.

3.1. Simplex Input Strictly Local functions

Clearly we cannot learn $g \circ f$ for any arbitrary functions f and g. The goal of this paper is to show that this is possible given a hypothesis space restricted to a class of functions.

3.1.1. Definition

We give such a hypothesis space based on the *input strictly local functions* (Vaysse, 1986; Chandlee, 2014). A function g is *input strictly k-local (ISL_k)* iff for any $w, v \in \Sigma^*$, $\texttt{suff}_{k-1}(w) = \texttt{suff}_{k-1}(v)$ implies that $g_w = g_v$ (Chandlee, 2014). A consequence of this definition is that for an ISL_k function g, $\mathbf{G} = \{g_w \mid w \in \texttt{suff}_{k-1}(\Sigma^*)\}$.

Example 3 *The function $g = f(T_2)$ in Fig. 1 is ISL_2. For this g, $g_a(b) = a$ for the 1-suffix a, whereas for any other 1-suffix u $g_u(b) = b$. Thus, there are only two distinct tail functions for g: g_a and g_u for all other 1-suffixes u.*

A useful property of the ISL class is that the canonical SFST for an ISL_k function has a predictable structure: each state represents a set of $k-1$ suffixes. This has been used in past learning algorithms for the ISL class (Chandlee et al., 2014; Jardine et al., 2014), and will play a role in the algorithm presented here.

In order to highlight the key insights of the algorithm, we simplify the learning problem by focusing on the following restriction to ISL_2 functions.

Definition 6 (Left-simplex ISL$_2$ function) *An ISL$_2$ function g is* left-simplex *iff it is properly ISL$_2$ and there is exactly one $g_{\text{env}} \in \mathbf{G}$ and $\tau \in \Sigma$ such that $g^p_{\text{env}}(\tau) \neq \tau$.*

We focus on properly ISL$_2$ functions as we are interested in functions with multiple tail functions (ISL$_1$ functions only have one tail function). Let e be the *environment*—that is, the 1-suffix after which the change in g occurs—for g. (That is, $g_e = g_{\text{env}}$.) Likewise, we refer to $\tau \in \Sigma$ s.t. $g_{\text{env}}(\tau) = w_\tau \neq \tau$ as the *target* of g and $w_\tau \in \Sigma^*$ as the *output* of τ. We can also refer to by g_{def} the "default" tail function which maps any symbol to itself.

Example 4 *The function $g = f(T_2)$ from Fig. 1 is also left-simplex; $e = a$, $\tau = b$ and $w_\tau = a$. That is, $g_a(b) = a$, so $g_a = g_{\text{env}}$. Conversely $g_{\text{def}} = g_{\lambda,b,c} = g_\lambda = g_b = g_c$; i.e., the tail function representing all other 1-suffixes. That is, $g_u(\sigma) = \sigma$ for any $\sigma \in \Sigma$ when $u = \lambda$, b, or c. Note that q_0 in T_2 corresponds to g_{def} and q_1 corresponds to g_{env}.*

3.1.2. Properties of simplex ISL$_2$ functions

As will be seen later on, in order to find g from the composition of $g \circ f$, it is necessary to determine the 1-suffixes in the *input* that correspond to differing tail functions of g from the 1-suffixes of the *output* prefixes for the differing tail functions of $g \circ f$. We thus define a function $IS : \mathbf{G} \to \texttt{suff}_1(\Sigma^*)$ that takes a tail function g_x of g and maps it to the *input suffixes* for g_x; that is, the 1-suffixes of the input of all strings whose tail function is g_x: $IS(g_x) = \{s \mid \exists w \text{ s.t. } g_w = g_x \text{ and } \texttt{suff}_1(w) = s\}$. Similarly, we define a function $OS : \mathbf{G} \to \texttt{suff}_1(\Sigma^*)$ that maps g_x to its *output suffixes*; that is, the 1-suffixes of the common outputs of all strings whose tail function is g_x: $OS(g_x) = \{s \mid \exists w \text{ s.t. } g_w = g_x \text{ and } \texttt{suff}_1(g^p(w)) = s\}$. Note that given a finite alphabet these sets will always be finite.

Example 5 *For $g = f(T_2)$ from Fig. 1, $IS(g_{\text{def}}) = \{\lambda, b, c\}$ and $IS(g_{\text{env}}) = \{a\}$. However, $OS(g_{\text{def}}) = \{\lambda, a, b, c\} \neq IS(g_{\text{def}})$ while $OS(g_{\text{env}}) = \{a\} = IS(g_{\text{env}})$.*

Let g be a left-simplex ISL$_2$ function. The following lemmas hold for $OS(g_{\text{env}})$. Lemma 8 shows that there are cases in which $IS(g_x) \subsetneq OS(g_x)$. Lemma 9 establishes a key fact for distinguishing g_{env} from g_{def}. Proofs for Lemmas 7 to 9 are given in Appendix A.2.

Lemma 7 *For $g_x \in \mathbf{G}$, $IS(g_x) \subseteq OS(g_x)$.*

Lemma 8 *For any left-simplex ISL$_2$ function g, $|OS(g_{\text{env}})| \leq 2$.*

Lemma 9 *For any simplex ISL$_2$ function g, $|OS(g_{\text{env}})| < |OS(g_{\text{def}})|$.*

3.2. Conditions on the two functions and their interaction

There must be conditions on f such that f and g can be discovered given some hypothesis for g. In particular, we cannot learn g unless its tail functions are detectable from the strings in the range of f. For the ISL$_2$ class, the below is sufficient. The proofs for the lemmas in this subsection are presented in Appendix A.3.

Lemma 10 (Distinguishing tail functions in ISL$_2$) *For any ISL$_2$ function g and all tail functions $g_\sigma, g_\varphi \in \mathbf{G}$, $g_\sigma \neq g_\varphi$ iff there exists some $\psi \in \Sigma \cup \{\lambda\}$ s.t. $g_\sigma(\psi) \neq g_\varphi(\psi)$.*

Definition 11 (Witnessing ISL$_2$) *A homomorphism $f : P^* \to \Sigma^*$* witnesses *the ISL$_2$ class iff $\forall \sigma \in \Sigma,\ \exists \rho \in P$ s.t. $\texttt{suff}_1(f(\rho)) = \sigma$ and $\forall \sigma \in \Sigma,\ \exists \rho \in P$ s.t. $\texttt{pref}_1(f(\rho)) = \sigma$.*

This is stronger than is strictly necessary for our learning algorithm, but it is simple to state and generalizes to the entire ISL$_2$ class. We now prove that given an f that witnesses the ISL$_2$ class, we can distinguish the tail functions of g from $g \circ f$.

Lemma 12 *Let f be a homomorphism that witnesses the ISL$_2$ class, let g be an ISL$_2$ function, and let $h = g \circ f$. h is a ISL$_2$ function and there is a bijection between $\mathbf{G}$ and $\mathbf{H}$ such that for every distinct tail function $g_\sigma \in \mathbf{G}$, there is a distinct tail function $h_\rho \in \mathbf{H}$ such that for any string $w \in P^*$ whose tail function is h_ρ, the tail function of $f(w)$ is g_σ.*

Thus, if g is a simplex ISL$_2$ function, then h has an environment tail function h_{env} that corresponds to g_{env} and a default tail function h_{def} that corresponds to g_{env}. This is why, for example, T_2 and $T_2 \circ T_1$ in Fig. 1 are isomorphic up to transitions.

However, in some cases $OS(h_x) \neq OS(g_x)$. In Ex. 5, $\texttt{OS}(g_{\text{def}}) = \{\lambda, a, b, c\}$ while it is possible that $\texttt{OS}(h_{\text{def}}) = \{\lambda, b, c\}$ when $w \notin \texttt{dom}(h)$. The mismatch is due to the fact that $g_w \neq g_{g^p(w)}$. We say that a string u is *opaque* iff $g_u \neq g_{g^p(u)}$, meaning that its input 1-suffix (and thus its tail function) is not visible from the output. We then amend the notion of witnessing to ensure that there are non-opaque strings that witness each 1-suffix.

Definition 13 (Surface-witnessing ISL$_2$ function) *A homomorphism $f : P^* \to \Sigma^*$* surface-witnesses *the ISL$_2$ function g iff there is a sub-function f' of f for whom $\texttt{ran}(f')$ includes no opaque strings respect to g and f' witnesses the ISL$_2$ class.*

Lemma 14 *For a homomorphism f that surface-witnesses a simplex ISL$_2$ function g, for $h = g \circ f$ it holds that for any g_x and h_x, $OS(h_x) \subseteq OS(g_x)$ and $IS(g_x) \subseteq OS(h_x)$.*

Lemma 15 $|OS(h_{\text{env}})| < |OS(h_{\text{def}})|$.

4. Algorithm

We now give an algorithm that learns two functions given the a sample of their composition.

4.1. Obtaining the composition transducer

Because both f and g are subsequential, $g \circ f$ is subsequential (Schützenberger, 1977). Thus, $g \circ f$ is strongly learnable in the limit from positive examples of $g \circ f$ in cubic time by the Onward Subsequential Transducer Inference Algorithm (OSTIA) (Oncina et al., 1993). The starting point of our algorithm is thus to learn the SFST for $g \circ f$ using OSTIA.

Briefly, OSTIA builds a prefix tree transducer using the input samples of pairs from $g \circ f$, and then employs a state merging strategy in which states with identical tail functions (with respect to the finite sample) are merged. Importantly, given a characteristic sample for $g \circ f$, OSTIA returns its canonical SFST.

4.2. Decomposition of composition transducer

The next step of the algorithm is to decompose this function into these constituent functions. Starting with the composition transducer T_f, decomposition aims to build T_g and turn T_f into a one-state machine representing a letter-to-string homomorphism. We use the running example from Fig. 1 to demonstrate the algorithm. The targets are $f = f(T_1)$ and $g = f(T_2)$ from Fig. 1. Let $D \subset g \circ f$ be the finite function represented by the set of pairs $\{(\lambda, \lambda), (w, aa), (x, cbcb), (y, bc), (z, bca), (ww, aaaa), (wx, aacbcb), (wy, aabc), (wz, aabca), (xw, cbcbaa), (xx, cbcbcbcb), (xy, cbcbbc), (xz, cbcbbca), (yw, bcaa), (yx, bccbcb), (yy, bcbc), (yz, bcbca), (zw, bcaaa), (zx, bcacbcb), (zy, bcaac), (zz, bcaaca)\}$. In Alg. 1, $\texttt{OSTIA}(D)$ generates a two-state composition transducer T_f, shown in Fig. 2, identical to $T_2 \circ T_1$ from Fig.1. Then the 1-suffixes of strings reaching the two states respectively are collected: $\texttt{OS}(q_1)$, which is $\{\lambda, a, b, c\}$, and $\texttt{OS}(q_2)$, which is $\{a\}$; these are underlined in Fig. 2.

Algorithm 1 The SI_2DLA

Data: A finite function $D : P^* \to \Sigma^*$

1 $T_f := \texttt{OSTIA}(D) = \langle Q_f = \{q_1, q_2\}, q_1 = q_{0f}, Q_{\ltimes f} = Q_f, \delta_f, \omega_f, \iota_f \rangle$

2 Define a mapping $\texttt{OS}$ as follows:

$\texttt{OS}(q_1) := \{\lambda\} \cup \{\texttt{suff}_1(\omega_f(q, \rho)) \mid \delta_f(q, \rho) = q_1\}$

$\texttt{OS}(q_2) := \{\texttt{suff}_1(\omega_f(q, \rho)) \mid \delta_f(q, \rho) = q_2\}$

3 $T_g, \texttt{corr}, \texttt{IS} := \texttt{construct\_}T_g(T_f, \texttt{OS})$

4 $T_f := \texttt{modify\_}T_f(T_f, T_g, \texttt{corr}, \texttt{IS})$

5 **return** (T_f, T_g)

Based on Lemma 12, Alg. 2 initializes T_g with two states q_{env} and q_{def}, corresponding to the environment tail function g_{env} and default tail function g_{def}, respectively, of g. Alg. 2 then defines $\texttt{corr}$ mapping states in T_g to states in T_f based on the size of the output suffix sets (based on Lemma 15). In this example, as $|\texttt{OS}(q_1)| = 4$ and $|\texttt{OS}(q_2)| = 1$, $\texttt{corr}(q_{\text{def}})$ is set to q_2 and $\texttt{corr}(q_{\text{env}})$ is set to q_1. This mapping is marked with dashed lines in Fig. 2.

Lines 3–5 in Alg. 2 then determine $\texttt{IS}$ for each state based on $\texttt{OS}$. First, $\texttt{IS}(q_x)$ is initialized to $\texttt{OS}(q_x)$. If $|\texttt{IS}(q_{\text{env}})| > 1$, then there are opaque strings and the environment e coincides with τ: $\texttt{OS}(q_{\text{env}}) = \{e, w_\tau\}$ (see the proof for Lemma 8). Thereby e is the element in $\texttt{IS}(q_{\text{env}}) - \texttt{IS}(q_{\text{def}})$. Otherwise $\texttt{IS}(q_{\text{env}}) = \texttt{OS}(q_{\text{env}})$ which is the case in the example: $\texttt{IS}(q_{\text{env}}) = \{a\}$. If $\texttt{IS}(q_{\text{env}}) \cap \texttt{IS}(q_{\text{def}})$ is still non-empty, this indicates that $e = \texttt{suff}_1(w_\tau) \in \texttt{OS}(q_{\text{def}})$ as shown in the example ($a \in \texttt{OS}(q_1)$ and $\texttt{OS}(q_2)$). In order to obtain the true $\texttt{IS}(q_{\text{def}})$, this overlapping element is deleted. $\texttt{IS}(q_{\text{env}})$ and $\texttt{IS}(q_{\text{def}})$ are now correctly calculated to be $\{a\}$ and $\{\lambda, b, c\}$, which corresponds to the input suffix sets for the function g (see Sec. 3.1.2).

Lines 6–9 of Alg. 2 compute the transition function for T_g. As T_g is ISL_2, the transition from each (q, σ) pair goes to q_{def} iff σ belongs to $\texttt{IS}(q_{\text{def}})$, otherwise it goes to q_{env}. In this example, $\delta_g(q_{\text{def}}, a) = q_{\text{env}}$ because $a \in \texttt{IS}(q_{\text{env}})$, $\delta_g(q_{\text{def}}, b) = q_{\text{def}}$ because $b \in \texttt{IS}(q_{\text{def}})$, etc. Lines 12–16 compute the output functions based on the correspondence between the composition transducer T_f and T_g. For all transitions from q_{def}, the output is identical to the input since no change should be expected after default state. For transitions from q_{env}, outputs are computed by comparing outputs coming out of $\texttt{corr}(q_{\text{def}})$ and $\texttt{corr}(q_{\text{env}})$; call these w_{def} and w_{env}. Since g is ISL_2, difference between w_{def} and w_{env} can only occur at their

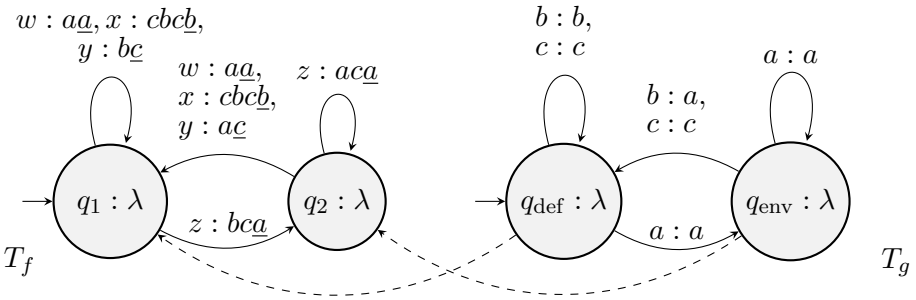

Figure 2: The initial state of T_f and its correspondence with T_g.

beginnings, i.e. w_{def} and w_{env} share the suffix $\texttt{rest}(w_{\text{def}})$ defined as $\texttt{first}(w_{\text{def}})^{-1} \cdot w_{\text{def}}$. Thereby $w_{\text{env}} \cdot \texttt{rest}(w_{\text{def}})^{-1}$ is the differing prefix between w_{env} and w_{def}, which is the output of change in g. For example, for $\delta_g(q_{\text{env}}, b)$, y is a $\rho \in P$ whose output starts with b. So $w_{\text{def}} = \omega_f(\texttt{corr}(q_{\text{def}}), y) = bc$ and $w_{\text{env}} = \omega_f(\texttt{corr}(q_{\text{env}}), y) = ac$. Thus $\omega_g(q_{\text{env}}, b) = w_{\text{env}} \cdot \texttt{rest}(w_{\text{def}})^{-1} = ac \cdot c^{-1} = a$.

Algorithm 3 computes T_f. All outgoing transitions from $\texttt{corr}(q_{\text{env}})$ are removed, as there are transitions whose output begins with w_τ instead of τ (i.e., $\omega(q_{\text{env}}, y) = ac$). The transitions from $\texttt{corr}(q_{\text{def}})$ are taken to be the 'true' outputs of f and thus become the basis of T_f. However, opaque strings need special treatment. For example, $f(w) = ab$ is opaque ($g_{ab} \neq g_{g^p(ab)} = g_{aa}$) and so the initial hypothesis for $f(w)$ is incorrectly $\omega_f(q_1, w) = aa$. However, Alg. 3 can detect this by checking if the 1-suffix of $\omega_f(q_1, w)$ is not in the input suffixes of $\delta_f(q_1, w) = q_1$ (ln 3). As it isn't ($a \notin \texttt{IS}(q_1)$), it must be the case (because g is left-simplex, and only one change can occur) that the end of the string is w_τ (=a, in this case), and it should be replaced with τ (b, in this case). Alg. thus 3 replaces a at the end of aa with b, and so $\omega_f(q_1, w)$ is correctly set to ab. In the end, states are merged for T_f. The reader can confirm that this correctly produces $T_f = T_1$ and $T_g = T_2$ from Fig. 1.

4.3. Sufficient Sample and Complexity

We now give the time and data complexity of the SI_2DLA, starting with the time complexity.

Lemma 16 *Given an input $D : P^* \to \Sigma^*$, let $n = \sum_{(x,y)\in D} |x|, m = max_{(x,y)\in D} |y|, k = |P|, l = |\Sigma|$. SI_2DLA runs in time $\mathcal{O}(n^3(m+k) + nmk + l)$.*

We now define a *sufficient sample* for SI_2DLA. Since it begins by running OSTIA, we give its characteristic sample in the Appendix A.4.

Definition 17 (Sufficient sample) *For a homomorphism f that surface-witnesses an ISL_2 function g, a finite subfunction H of $h = g \circ f$ is a* sufficient sample *iff $P^{\leq 2} \subseteq \texttt{dom}(H)$.*

Remark 18 *A sufficient sample for SI_2DLA is also a CS for OSTIA.*

The proofs for Lemma 16 and Remark 18 are presented in Appendix A.4. The lemma for the complexity of the sample below we give without proof.

Lemma 19 *The size of a sufficient sample for a pair of transtucers T_g, T_f is constant.*

Algorithm 2 The $\texttt{construct\_}T_g$ function. Here $\texttt{first}(w)$ returns the first symbol in w; and $\texttt{rest}(w)$ returns the remainder of w.

Data: An SFST T_f, a mapping $\texttt{OS}$

$Q_g = \{q_{\text{env}}, q_{\text{def}}\}$
Define a bijective mapping $\texttt{corr}$ from Q_g to Q_f:
  $\texttt{corr}(q_{\text{def}}) := q_i \in Q_f$ where $\texttt{OS}(q_i)$ is the larger of $\texttt{OS}(q_1), \texttt{OS}(q_2)$
  $\texttt{corr}(q_{\text{env}}) := q_j \in Q_f$ where $\texttt{OS}(q_i)$ is the smaller of $\texttt{OS}(q_1), \texttt{OS}(q_2)$
Define a mapping $\texttt{IS}$:
  $\texttt{IS}(q_{\text{env}}) := \texttt{OS}(\texttt{corr}(q_{\text{env}}))$
  $\texttt{IS}(q_{\text{def}}) := \texttt{OS}(\texttt{corr}(q_{\text{def}}))$
if $|\texttt{IS}(q_{\text{env}})| > 1$ **then** $\texttt{IS}(q_{\text{env}}) := \texttt{IS}(q_{\text{env}}) - (\texttt{IS}(q_{\text{env}}) \cap \texttt{IS}(q_{\text{def}}))$;
$\texttt{IS}(q_{\text{def}}) := \texttt{IS}(q_{\text{def}}) - (\texttt{IS}(q_{\text{env}}) \cap \texttt{IS}(q_{\text{def}}))$
for $q_i \in Q_g$ **do**
  for $\sigma \in \Sigma$ **do**
    Let q_j be the state for which $\sigma \in \texttt{IS}(q_j)$
    Set $\delta_g(q_i, \sigma) = q_j$
  end
end
for $(q_{\text{def}}, \sigma) \in \texttt{dom}(\delta_g)$ **do** $\omega_g(q_{\text{def}}, \sigma) := \sigma$;
for $(q_{\text{env}}, \sigma) \in \texttt{dom}(\delta_g)$ **do**
  Choose some $\rho \in P$ s.t. $\texttt{first}(\omega_f(\texttt{corr}(q_{\text{def}}), \rho)) = \sigma$
  Let $w_{\text{def}} = \omega_f(\texttt{corr}(q_{\text{def}}), \rho)$, $w_{\text{env}} = \omega_f(\texttt{corr}(q_{\text{env}}), \rho)$
  $\omega_g(q_{\text{env}}, \sigma) := w_{\text{env}} \cdot \texttt{rest}(w_{\text{def}})^{-1}$
end
$q_{0g} := \texttt{corr}^{-1}(q_{0f})$
$Q_{\ltimes g} := Q_g$
$\iota_g := \iota_f$
return T_g, $\texttt{corr}$, $\texttt{IS}$

Algorithm 3 The $\texttt{modify\_}T_f$ algorithm

Data: SFSTs T_f, T_g, functions $\texttt{corr}$,$\texttt{IS}$

Delete each $(\texttt{corr}(q_{\text{env}}), \rho)$ from δ_f, ω_f
$\tau := \sigma$ such that $\omega_g(q_{\text{env}}, \sigma) \neq \sigma$
$w_\tau := \omega_g(q_{\text{env}}, \sigma)$
for $(\texttt{corr}(q_{\text{def}}), \rho) \in \texttt{dom}(\omega_f)$ *such that* $\texttt{suff}_1(\omega_f(\texttt{corr}(q_{\text{def}}), \rho)) \notin \texttt{IS}(\texttt{corr}^{-1}(\delta_f(\texttt{corr}(q_{\text{def}}), \rho)))$ **do**
  // w_τ is a suffix of $\omega_f(\texttt{corr}(q_{\text{def}}), \rho)$; replace w_τ with τ
  $\omega_f(\texttt{corr}(q_{\text{def}}), \rho) := (\omega_f(\texttt{corr}(q_{\text{def}}), \rho) \cdot w_\tau^{-1}) \cdot \tau$
end
Merge the two states in Q_f
return T_f

4.4. Complexity and correctness proofs

Let $\hat{f} : P^* \to \Sigma^*$ and $\hat{g} : \Sigma^* \to \Sigma^*$ be the target functions, with $\hat{f}$ being a letter-to-string homomorphism and $\hat{g}$ a left-simplex ISL_2 function, and let $\hat{h} = \hat{g} \circ \hat{f}$ be their composition. Let $T_{\hat{f}} = T(\hat{f})$ and $T_{\hat{g}} = T(\hat{g})$. The learning target is then $\langle T_{\hat{f}}, T_{\hat{g}} \rangle$. We give sketches of the proofs here, for the full details see Appendix A.5.

Lemma 20 *If $\hat{f}$ surface-witnesses $\hat{g}$, then given a sufficient sample of $\hat{g} \circ \hat{f}$, SI$_2$DLA returns $\langle T_f, T_g \rangle$ such that $T_g = T_{\hat{g}}$.*

Proof (sketch) Let $\langle T_f, T_g \rangle = \mathrm{SI}_2\mathrm{DLA}(D)$ for a sample D of $\hat{h}$ containing a sufficient sample. As $\hat{g}$ is a simplex ISL_2 function, it has two tail functions $\hat{g}_{\mathrm{env}}$ and $\hat{g}_{\mathrm{def}}$, and as $\hat{f}$ witnesses the ISL_2 class, by Lemma 12, $\hat{h}$ has exactly two corresponding tail functions $\hat{h}_{\mathrm{env}}$ and $\hat{h}_{\mathrm{def}}$. Thus $\mathrm{OSTIA}(D) = T_f$ is a two-state machine whose states represent these two tail functions. Because $\hat{f}$ surface-witnesses $\hat{g}$, and because the sufficient sample contains all two-symbol combinations of P, when Alg. 1 calculates the output suffixes `OS` for these two states, it is equal to the output suffix function OS for the two tail functions of $\hat{h}$.

Then, Alg. 2 creates a new SFST with two states, q_{env} and q_{def}, and following Lemmas 9 and 15 associates the latter with the state in T_f with the larger set of output suffixes. As such, it correctly associates q_{env} with $\hat{h}_{\mathrm{env}}$—and thus $\hat{g}_{\mathrm{env}}$—and likewise q_{def} with $\hat{h}_{\mathrm{def}}$ and $\hat{g}_{\mathrm{def}}$. As $IS(\hat{g}_x) \subseteq OS(\hat{h}_x)$ (Lemma 14), `OS` discovers the input suffixes `IS` for q_{env} and q_{def}. To do this, lines 4 and 5 remove any opaque suffixes based on the proof of Lemma 8.

The δ_g function is predetermined by the structure of ISL_2 SFSTs—any transition on $\sigma \in \Sigma$ goes to the state to which σ is a 1-prefix. The ω_g function for q_{def} is predetermined $\hat{g}$ being simplex—all outputs are set equal to their inputs. For q_{env}, ω_g is calculated by comparing the outputs from the $\hat{h}_{\mathrm{def}}$ and $\hat{h}_{\mathrm{env}}$ states in T_f. As shown in the above explanation, this correctly sets the target τ to its output w_τ and any other σ to itself. ■

Lemma 21 *If $\hat{f}$ surface-witnesses $\hat{g}$, then given a sufficient sample D of $\hat{g} \circ \hat{f}$, SI$_2$DLA returns $\langle T_f, T_g \rangle$ such that $f(T_g \circ T_f) = \hat{h} = f(T_{\hat{g}} \circ T_{\hat{f}})$.*

Proof (sketch) Essentially, the goal is to produce a homomorphism as close to $\hat{f}$ as possible by using the outputs of the transitions of the state in T_f that is associated with $\hat{h}_{\mathrm{def}}$ (*i.e.*, $\mathtt{corr}(q_{\mathrm{def}})$). The outputs from the state associated with $\hat{h}_{\mathrm{env}}$ are discarded, as they represent changes that are dependent on the tail functions of $\hat{g}$. Out of the remaining outputs, there may be outputs that are the result of an opaque string. Line 6 detects these by finding mismatches in the 1-suffixes of the outputs with the input 1-suffixes of q_{def} in T_g. As $\hat{g}$ is left-simplex ISL_2, this is exactly the case when $w = ue\tau$ for some u—i.e. that $\hat{g}(w) = u'w_\tau$ for some u'—and $\hat{g}_\tau \neq \hat{g}_{w_\tau}$. Line 6 thus changes this to $u'\tau$, which is because $\hat{g}$ is ISL_2 with a tail function $\hat{g}_\tau$. Thus, when in the last step the two states of T_f are merged, this creates an SFST describing a homomorphism such that $f(T_g \circ T_f) = \hat{h} = f(T_{\hat{g}} \circ T_{\hat{f}})$. ■

We now give the main theorem of the paper.

Theorem 22 *For the class of functions $h \in \mathbb{H}$ described by pairs $\langle T_f, T_g \rangle \in \mathbb{R}$ of transducers where $f(T_g)$ is a left-simplex ISL_2 function and $f(T_f)$ is a homomorphism that surface witnesses $f(T_g)$, $\mathbb{H}$ is semi-strongly learnable in polynomial time and data by the SI_2DLA.*

Proof From lemmas 16, 19, 20, and 21. ■

5. Discussion and conclusion

We now situate this result both in its empirical applications and how it can be generalized to future theoretical work.

5.1. Empirical motivation

Generative phonology posits that systematic variations in the pronunciations of morphemes are best explained by mapping an abstract *underlying representation* (UR) of the phonological content of a morpheme to one or more concrete *surface representations* (SRs) using a phonological grammar (Chomsky and Halle, 1968). For example, the pronunciation of the English past tense suffix varies depending on its phonological context. In 'wagged' it is pronounced as a voiced [d] (brackets [ ] indicating an SR), whereas in 'napped' it is pronounced as a voiceless [t]. An analysis based on the principles of generative phonology posits that the morpheme PAST in English has a single abstract underlying representation /d/ (slashes / / indicating a UR) that is mapped by the phonology to [t] after voiceless sounds (like the /p/ in 'napped') and [d] elsewhere (e.g., the /g/ in 'wagged', which is voiced).

We can thus model the URs of morphemes with a homomorphism f: $f(\text{WAG}) = \text{wæg}$, $f(\text{NAP}) = \text{næp}$, $f(\text{NAP+PAST}) = \text{næpd}$, etc., and the phonology of English as an ISL_2 function g that changes /d/ to [t] following voiced sounds: $g(\text{wægd}) = \text{wægd}$ and $g(\text{næpd}) = \text{næpt}$. The surface pronunciation of the past tense of 'nap' is thus $g(f(\text{NAP+PAST})) = \text{næpt}$.

A major learning problem in phonological theory is how children acquire both of these functions. URs, being abstract mental representations, are not directly present in the child's linguistic input. They must then be inferred, along with the phonology function, from the surface pronunciations of words—in other words, from the composition of the two functions. That is, the child learner must somehow infer that, because WAG+PAST is pronounced [wægd] and NAP+PAST is pronounced [næpt], that the UR of PAST is /d/ and the phonology function that maps /d/ to [t] after voiceless sounds.

A toy example in Fig. 3 illustrates how this problem maps onto the decomposition problem solved by the SI_2DLA. Fig. 3 gives a morphology-to-UR function mapping a morpheme 1 to *tat*, 2 to *ada*, and 3 to *d* (T_f), a phonology function mapping *d* to *t* after a *t* (T_g), and a composition machine that shows how sequences of morphemes are mapped to their SRs ($T_g \circ T_f$). Note that g is simplex ISL_2 and that f surface-witnesses g, the two can be learned from a sample of their composition by the SI_2DLA.

This is a simplification: phonological processes target groups of sounds instead of single sounds. However, there are many ways in which this captures basic concepts of the problem. First, Chandlee (2014) showed that 94% of the processes in the P-Base database (Mielke, 2004) are ISL_k for some k, with the $k = 2$ for a great number of them. While phonological processes target more than one sound, usually (but not always) target a single *natural*

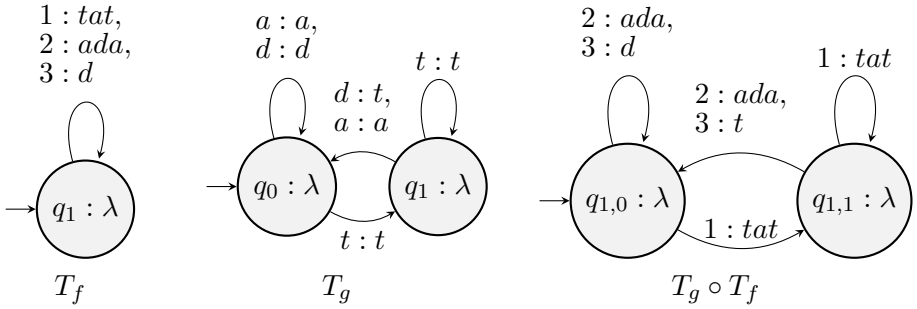

Figure 3: A toy phonological example.

class of sounds (Chomsky and Halle, 1968), and so are simplex given the right kind of representation. Thus, this is a limited example, but we expect future work to build on this result to increase its empirical coverage. We turn to these possibilites now.

5.2. Future work

The core insight of the current result is determining the tail functions of the second function g from its output strings. The fact that the first function f had a single tail function—and thus this did not factor into the learning problem—can also be adjusted to study the interaction of the tail functions of two different functions. However, the most immediate way to generalize the present result is to also capture *right*-simplex ISL_2 functions, which can be defined as the reversal of a left-simplex function (analagous to the right-subsequential functions). Both the left and right-simplex ISL_2 functions can be learned with the same procedure, but we have omitted this for concerns of length of the paper.

Similarly, extending this result to simplex ISL_k functions for k in general is straightforward, with more details to deal with multiple tail functions and cases of opacity. However, interestingly, in this case there will be $k-2$ 'default' tail functions, each keeping track of a prefix of the 'environment' $k-1$ suffix. This will likely not complicate the learning problem, however, as the algorithm still can focus on finding the single 'environment' tail function.

Finally, a prime target for future work is the *output*-strictly local functions (Chandlee et al., 2015), whose tail functions are defined based on their output and are also empirically relevant to phonology (Chandlee, 2014). Thus, this specific result opens the door to studying the general problem of simultaneous learning of multiple functions for sub-classes of subsequential functions.

Acknowledgements

The authors would like to thank Huteng Dai and other members of the Rutgers Mathematical Linguistics research group, the audience at the 2019 Stonybrook Workshop on Computational Phonology, and anonymous reviewers for ICGI and TACL for their helpful comments and suggestions.

References

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

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 Andras Kornai and Marco Kuhlmann, editors, *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 14)*, pages 52–63, Chicago, IL, July 2015.

Noam Chomsky and Morris Halle. *The Sound Pattern of English.* Harper & Row, 1968.

Alexander Clark. Learning trees from strings: A strong learning algorithm for some context-free grammars. *Journal of Machine Learning Research*, 14:3537–3559, 2014. URL papers/clark13a.pdf.

Colin de la Higuera. *Grammatical Inference: Learning Automata Grammars.* Cambridge University Press, 2010.

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

Adam Jardine, Jane Chandlee, Rémi Eyraud, and Jeffrey Heinz. Very efficient learning of structured classes of subsequential functions from positive data. In *Proceedings of the 12th International Conference on Grammatical Inference (ICGI 2014)*, JMLR Workshop Proceedings, pages 94–108, 2014.

M. Lothaire. *Applied Combinatorics on Words.* Cambridge University Press, 2005.

Jeff Mielke. P-Base 1.95. http://137.122.133.199/jeff/pbase, 2004.

Mehryar Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.

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

Jacques Sakarovitch. *Elements of Automata Theory.* Cambridge, 2009.

Marcel Paul Schützenberger. Sur une variante des fonctions séquentielles. *Theoretical Computer Science*, 4:47–57, 1977.

O. Vaysse. Addition molle et fonctions p-locales. *Semigroup Forum*, 34:157–175, 1986.

Appendix

A.1. Composition of Subsequential Functions

To represent the composition of subsequential functions, we adapt the composition of two SFSTs by Lothaire (2005). For two SFSTs $T' = \langle Q', q'_0, Q'_\ltimes, \delta', \omega', \iota' \rangle$ and $T'' = \langle Q'', q''_0, Q''_\ltimes, \delta'', \omega'', \iota'' \rangle$, $T'' \circ T' \stackrel{\text{def}}{=} \langle Q' \times Q'', (q'_0, q''_0), Q'_\ltimes \times Q''_\ltimes, \delta, \omega, \iota \rangle$ where

$$\omega((q', q''), \sigma) \stackrel{\text{def}}{=} w, \delta((q', q''), \sigma) \stackrel{\text{def}}{=} (r', r''), \iota((q'_\ltimes, q''_\ltimes)) \stackrel{\text{def}}{=} \omega''^*(q''_\ltimes, \iota'(q'_\ltimes))$$
$$\text{where } \omega'(q', \sigma) = v, \omega''^*(q'', v) = w \text{ for some } v \in \Sigma^*, \delta'(q', \sigma) = r' \text{ and } \delta''^*(q'', v) = r''$$

A.2. Proofs from Section 3.1

Lemma 7 *For $g_x \in \mathbf{G}$, $IS(g_x) \subseteq OS(g_x)$*

Proof Since $g_{\text{def}}(\sigma) = \sigma$ for every $\sigma \in \Sigma$, for some arbitrary $s \in IS(g_x)$, by taking some $u \in \Sigma^*$ and $d \in IS(g_{\text{def}})$, we can ensure that $g^p_{ud}(s) = s$. Thus by choosing uds as w from the definition of OS, we can ensure that $s = \texttt{suff}_1(g^p(w))$ and thus that $s \in OS(g_x)$. ■

The following lemma essentially shows that there are cases in which $IS(g_x) \subsetneq OS(g_x)$.

Lemma 8 *For any left-simplex ISL$_2$ function g, $|OS(g_{\text{env}})| \leq 2$.*

Proof First, by definition of left-simplex ISL$_2$, $|IS(g_{\text{env}})| = |\{e\}| = 1$, so $|OS(g_{\text{env}})| \geq 1$. However, in the case that $e = \tau$ and $\texttt{suff}_1(w_\tau) \neq \tau$, then there can be strings of the form $u\tau\tau$ whose input 1-suffix is $\tau = e$ but whose output 1-suffix is $\texttt{suff}_1(w_\tau)$. To give an example, let g be a left-simplex ISL$_2$ function such that $g^p_a(a) = b$. Then $OS(g_{\text{env}}) = \{a, b\}$ because when $w = a$, $g^p_w(a) = b$ where $a = \texttt{suff}_1(g(w)) = \texttt{suff}_1(g(a)) = \texttt{suff}_1(a)$; when $w = aa$, $g^p_w(a) = b$ where $b = \texttt{suff}_1(g(w)) = \texttt{suff}_1(g(aa)) = \texttt{suff}_1(ab)$.

However, as g is left-simplex, $g_\tau(\tau) = w_\tau$ is the only change that can be present, so at most $OS(g_{\text{env}}) = \{e, \texttt{suff}_1(w_\tau)\}$, whose cardinality is 2. ■

Lemma 9 *For any simplex ISL$_2$ function g, $|OS(g_{\text{env}})| < |OS(g_{\text{def}})|$.*

Proof As we assume $|\Sigma| \geq 3$ (see §2.1), $|\texttt{suff}_1(\Sigma^*)| = |\{\lambda\} \cup \Sigma| = 1 + |\Sigma| \geq 4$. As g is simplex and thus there are only two tail functions, $|IS(g_{\text{def}})| = |\texttt{suff}_1(\Sigma^*)| - |IS(g_{\text{env}})| = |\texttt{suff}_1(\Sigma^*)| - 1 \geq 3$. Then by Lemma 7, $|OS(g_{\text{def}})| \geq 3$, and as by Lemma 8, $|OS(g_{\text{env}})| \leq 2$, then the inequality $|OS(g_{\text{def}})| > |OS(g_{\text{env}})|$ always holds. ■

A.3. Proofs from Section 3.2

Lemma 10 (Distinguishing tail functions in ISL$_2$) *For any ISL$_2$ function g, for all tail funtions $g_\sigma, g_\varphi \in \mathbf{G}$, if $g_\sigma \neq g_\varphi$ there exists some $\psi \in \Sigma \cup \{\lambda\}$ such that $g_\sigma(\psi) \neq g_\varphi(\psi)$.*

Proof By contradiction. Say that $g_\sigma \neq g_\varphi$ but $g_\sigma(\lambda) = g_\varphi(\lambda)$ and for all $\psi \in \Sigma$, $g_\sigma(\psi) = g_\varphi(\psi)$. So there must be some ψ and u such that $g_\sigma(\psi u) \neq g_\varphi(\psi u)$; in other words, then $g_{\sigma\psi}(u) \neq g_{\varphi\psi}(u)$. But because g is ISL$_2$, then $g_{\sigma\psi}(u) = g_\psi(u) = g_{\varphi\psi}(u)$; contradiction. ■

Lemma 12 *Let f be a homomorphism that witnesses the ISL$_2$ class, let g be an ISL$_2$ function, and let $h = g \circ f$ be their composition. Then h is a ISL$_2$ function and there is a bijection between $\mathbf{G}$ and $\mathbf{H}$ such that for every distinct tail function $g_\sigma \in \mathbf{G}$, there is a distinct tail function $h_\rho \in \mathbf{H}$ such that for any string $w \in P^*$ whose tail function is h_ρ, the tail function of $f(w)$ is g_σ.*

Proof As per Lemma 10, if two tail functions $g_\sigma, g_\varphi \in \mathbf{G}$ are distinct then there is some third $\varphi \in \Sigma$ such that $g_\sigma(\psi) \neq g_\varphi(\psi)$. As per Def. 11, if f witnesses the ISL$_2$ class then there are $\rho_1, \rho_2 \in P$ such that their 1-suffixes under f are σ and ψ, respectively, and there is a ρ_3 whose 1-prefix is ψ. Thus for some $w \in \Sigma^*$, $g_{f(\rho_1)}(f(\rho_3)) = g_\sigma(\psi w) \neq g_{f(\rho_2)}(f(\rho_3)) = g_\varphi(\psi w)$, and so $h_{\rho_1}(\rho_3) \neq h_{\rho_2}(\rho_3)$. So g_σ corresponds to a distinct tail function h_{ρ_1} and g_φ corresponds to a distinct tail function h_{ρ_2}.

Furthermore, as f is a homomorphism, $|\mathbf{F}| = 1$ and thus $|\mathbf{F} \times \mathbf{G}| = |\mathbf{G}|$, so by Lemma 1, $|\mathbf{H}| \leq |\mathbf{G}|$. Thus, there can be no other distinct tail functions in $\mathbf{H}$ beyond those that correspond to tail functions in $\mathbf{G}$. ■

Lemma 14 *For a homomorphism f that surface-witnesses a simplex ISL$_2$ function g, for their composition $h = g \circ f$ it holds that for any tail function 1) $OS(h_x) \subseteq OS(g_x)$; and 2) $IS(g_x) \subseteq OS(h_x)$*

Proof As f witnesses the ISL$_2$ class, its range includes strings in the tails of g_x, so for any $s \in OS(h_x)$, it must also be in $s \in OS(g_x)$. Item (2) follows from the fact that f *surface*-witnesses g, as for any $s \in IS(g_x)$, there is a string w in the range of f whose 1-suffix is s and, because w is not opaque, the 1-suffix of $g^p(w)$ is also s. ■

Lemma 15 $|OS(h_{\text{env}})| < |OS(h_{\text{def}})|$.

Proof This follows directly from Lemmas 14 and 9. ■

A.4. Characteristic Sample for OSTIA

The characteristic sample of OSTIA is defined based on two definitions: *short prefix* and *kernel*.

Definition (short prefixes) *Let $t : \Sigma^* \to \Gamma^*$ be a subsequential function. The string u is a* short prefix *of t iff $u \in \texttt{prefs}(\texttt{dom}(t))$ and $\forall v \in \Sigma^*, t_u(w) = t_u(v)$ implies $v \geq w$. Let $SP(t)$ denote the short prefixes of t.*

Definition (kernel) *Let $t : \Sigma^* \to \Gamma^*$ be a subsequential function. The* kernel *of t is the set $K(t) = \big((SP(t) \cdot \Sigma) \cap \texttt{prefs}(\texttt{dom}(t))\big) \cup \{\lambda\}$.*

That is, the short prefixes of t is the set of shortest prefixes belonging to each tail function of t. The kernel of t is then each short prefix extended with each symbol in Σ. Having these in the characteristic example ensures that the algorithm sees examples of each tail function of t.

Definition (characteristic sample for OSTIA) *Given a total subsequential function $t : \Sigma^* \to \Gamma^*$, a strong characteristic sample T for t for OSTIA is as follows:*

1. $\forall u \in K(t), \exists v \in \texttt{dom}(T)$, $u \in \texttt{prefs}(v)$
2. $\forall u \in SP(t), \forall v \in K(t)$, *if* $t_u \neq t_v$ *then there exist* $uw, vw \in \texttt{dom}(T)$ *such that* $T(uw) = t^p(u)w'$, $T(vw) = t^p(v)w''$ *and* $w' \neq w''$
3. $\forall u \in K(t), \exists uv, uw \in \texttt{dom}(T)$, $T(uv) = t^p(u)v'$, $T(uw) = t^p(u)w'$ *where* $v' \wedge w' = \lambda$

That is, each string in the kernel of t is a prefix in the domain of T (1); for each short prefix u, for each kernel string v with a different tail function then there are examples in T such that $T_u \neq T_v$ (2); and for each kernel string u there are two extensions of u such that $t^p(u) = T^p(u)$ (3).

Lemma 16 *Given an input $D : P^* \to \Sigma^*$, let $n = \sum_{(x,y)\in D} |x|, m = max_{(x,y)\in D} |y|, k = |P|, l = |\Sigma|$. SI$_2$DLA runs in time $\mathcal{O}(n^3(m+k) + nmk + l)$.*

Proof Based on Oncina et al. (1993), the time complexity of OSTIA is $\mathcal{O}(n^3(m+k)+nmk)$. For the complexity of the decomposition algorithm, notice that the complexity of a for-loop on $\texttt{dom}(\delta_f)$ is linear with respect to n, a for-loop on $\texttt{dom}(\delta_g)$ is linear with respect to l, a for-loop on $\texttt{dom}(\delta_g)$ starting from q_{def} is sub-linear to l while a for-loop on $\texttt{dom}(\delta_g)$ starting from q_{env} is a constant based on Lemma 9. Thereby algorithm 2 has complexity $\mathcal{O}(2*l+l+n)$ and algorithm 3 has complexity $\mathcal{O}(l+n)$. Therefore the decomposition algorithm has complexity of $\mathcal{O}(n + l)$ and complexity of SI$_2$DLA has complexity $\mathcal{O}(n^3(m + k) + nmk + l)$. ■

Remark 18 *A sufficient sample for SI$_2$DLA is also a CS for OSTIA.*

Proof There are two tail functions h_{env} and h_{def}, one of which is equal to h_λ for the short prefix λ and one of which is equal to h_ρ for some short prefix $\rho \in P$. (The other short prefix is of length 1 because it is ISL$_2$.) The kernel is thus $\{\lambda\} \cup (\lambda \cdot P) \cup (\rho \cdot P)$. It is easy to then verify that H satisfies all the requirements in the definition of the OSTIA CS. ■

A.5. Correctness proofs

Lemma 20 *If $\hat{f}$ surface-witnesses $\hat{g}$, then given a sufficient sample of $\hat{g}\circ\hat{f}$, SI$_2$DLA returns $\langle T_f, T_g\rangle$ such that $T_g = T_{\hat{g}}$.*

Proof Let D be a sample that contains a sufficient sample, and let $\langle T_f, T_g\rangle$ =SI$_2$DLA(D).

By Lemma 12, as $\hat{f}$ witnesses the ISL$_2$ class, there is a bijection between the tail functions of $\hat{g}$ and $\hat{h}$. As $\hat{g}$ is a simplex ISL$_2$ function, it has exactly two tail functions $\hat{g}^p_{\text{env}}$ and $\hat{g}^p_{\text{def}}$. Thus $\hat{h}$ has exactly two tail functions, and so the output of OSTIA—i.e., the initial hypothesis for T_f—will be a two-state machine with each state representing these two tail functions.

These states are q_1 and q_2 in T_f, where q_1 is the initial state. In line 2, algorithm 1 computes the output suffixes for q_1 and q_2 from D. Let $\hat{h}_1, \hat{h}_2$ be the tail functions

represented by q_1 and q_2, respectively. For each $i \in \{1, 2\}$, $\texttt{OS}(q_i) = OS(\hat{h}_i)$, because by the definition of surface-witnessing $\hat{g}$, there is some $\hat{f}(\rho)$ for every 1-suffix associated with $\hat{h}_i$, and because the sample D is sufficient its domain contains ρ as a prefix. So for every $s \in OS(\hat{h}_i)$, there is a transition (q, ρ) to q_i such that $\texttt{suff}_1(\omega(q, \rho)) = s$ and thus $s \in \texttt{OS}(q_i)$. (The other direction of the inclusion follows from D being a sample of $\hat{h}$.)

Algorithm 2 then creates a new SFST T_g by first creating two states, q_{env} and q_{def} (Alg. 2, ln 1). It then creates a function $\texttt{corr}$ such that $\texttt{OS}(\texttt{corr}(q_{\text{env}}))$ is smaller than than $\texttt{OS}(\texttt{corr}(q_{\text{def}}))$ (ln 2). As from Lemma 15 we know that $|OS(\hat{h}^p_{\text{env}})| < |OS(\hat{h}^p_{\text{def}})|$, then this correctly associates q_{env} with $\hat{h}^p_{\text{env}}$ and q_{def} and $\hat{h}^p_{\text{def}}$.

Alg. 2 then begins determining the input 1-suffix e associated with g_{env}. Lines 3 through 5 do this by calculating the input suffixes for q_{env} and q_{def} from the output suffixes of $\texttt{corr}(q_{\text{env}})$ and $\texttt{corr}(q_{\text{def}})$. This is possible because as by Lemma 14, $IS(\hat{g}_x) \subseteq OS(\hat{h}_x)$. All that remains is to deal with opaque cases, which is first done by removing from $\texttt{IS}(q_{\text{env}})$ any suffixes shared with $\texttt{IS}(q_{\text{def}})$ if the cardinality of $\texttt{IS}(q_{\text{env}})$ is greater than 1 (ln 4). (This is the case in which there is some $\hat{f}(\rho)$ that ends in $e\tau$ and $\texttt{suff}_1(w_\tau) \in IS(g_{\text{def}})$.) The same is then done for $\texttt{IS}(q_{\text{def}})$ (in case $e \neq \tau$ and there is some $\hat{f}(\rho)$ that ends in $e\tau$ and $\texttt{suff}_1(w_\tau) = e$). As because the only suffixes in $OS(\hat{h}_x)$ not in $IS(\hat{g}_x)$ are due to these opaque cases, then $\texttt{IS}(q_{\text{def}}) = IS(\hat{g}^p_{\text{def}})$ and $\texttt{IS}(q_{\text{env}}) = IS(\hat{g}^p_{\text{env}}) = \{e\}$, and e has correctly been found.

The algorithm then constructs the transitions for T_g based on the basic structure of the ISL_2 class: $\delta(q_i, \sigma)$ is set to whichever state has σ in its input suffixes. As this is identical for $T_{\hat{g}}$ has the same structure (see the definition of ISL_k functions), the transition functions for $T_{\hat{g}}$ and T_g are identical. Additionally, as the states in $T_{\hat{g}}$ correspond to $\hat{g}^p_{\text{env}}$ and $\hat{g}^p_{\text{def}}$, then the states of T_g correspond to the states in $T_{\hat{g}}$.

Finally, the algorithm identifies the target τ of $\hat{g}$ and its output w_τ by comparing outputs from state q_{def} with those of q_{env}. This takes place in the loop starting on ln 12. As $\hat{f}$ surface-witnesses $\hat{g}$, there is at least one $\rho \in P$ such that $\tau = \texttt{pref}_1(\hat{f}(\rho))$. As D is a sufficient sample of $\hat{g} \circ \hat{f}$, for any such ρ there is some $\rho'\rho \in \texttt{prefs}(\texttt{dom}(D))$ such that $\hat{h}_{\rho'} = \hat{h}^p_{\text{env}}$ as well as some $\rho''\rho \in \texttt{prefs}(\texttt{dom}(D))$ such that $\hat{h}_{\rho''} = \hat{h}^p_{\text{def}}$. (Note that either ρ' or ρ'' could potentially be λ.) Thus, in T_f there are transitions on ρ from $\texttt{corr}(q_{\text{def}})$ and $\texttt{corr}(q_{\text{env}})$ such that $\texttt{first}(\omega_f(\texttt{corr}(q_{\text{def}}), \rho)) = \tau$ (as $\texttt{first}(\hat{h}^p_{\text{def}}(\rho)) = \tau$) and w_τ is a prefix of $\omega_f(\texttt{corr}(q_{\text{env}}), \rho)$ (because w_τ is a prefix of $\hat{h}^p_{\text{def}}(\rho)$).

As such, Alg. 2 finds some such ρ (ln. 14). Setting the outputs of $\omega_f(\texttt{corr}(q_{\text{def}}), \rho)$ and $\omega_f(\texttt{corr}(q_{\text{def}}), \rho)$ to w_{def} and w_{env}, respectively, it finds w_τ by taking $w_{\text{env}} \cdot \texttt{rest}(w_{\text{def}})^{-1}$. Note that if $\hat{f}(\rho) = \tau u$ for some $u \in \Sigma^*$, then $\hat{h}^p_{\text{def}}(\rho) = \tau \cdot \hat{g}_\tau(u)$ and $\hat{h}^p_{\text{env}}(\rho) = w_\tau \cdot \hat{g}_\tau(u)$. Thus $w_{\text{env}} \cdot \texttt{rest}(w_{\text{def}})^{-1} = (w_\tau \cdot (\hat{g}_\tau(u))) \cdot \hat{g}_\tau(u)^{-1} = w_\tau$.

For all other $\sigma \in \Sigma$, it is straightforward to verify that this same loop will set the output to σ. Similarly, the for loop in the preceding line will set all outputs from the q_{def} to σ. As $\hat{g}$ is left-simplex, this is also true for the corresponding transitions in $T_{\hat{g}}$. Thus $\omega_g = \omega_{\hat{g}}$, where $\omega_{\hat{g}}$ is the transition output function in $T_{\hat{g}}$.

The remainder of the algorithm sets the initial state, final states, and state output functions to T_f. It is straightforward to verify that these are the same as $T_{\hat{g}}$. Thus, SI_2DLA constructs T_g so that all of its elements are identical to those of $T_{\hat{g}}$, so $T_g = T_{\hat{g}}$. ■

Lemma 21 *If $\hat{f}$ surface-witnesses $\hat{g}$, then given a sufficient sample D of $\hat{g} \circ \hat{f}$, SI$_2$DLA returns $\langle T_f, T_g \rangle$ such that $f(T_g \circ T_f) = \hat{h} = f(T_{\hat{g}} \circ T_{\hat{f}})$.*

Proof First, for any homomorphism f', $\hat{g} \circ f' = \hat{h}$ iff for all $\rho \in P$, 1) $\hat{g}^p_{\text{def}}(f'(\rho)) = \hat{h}^p_{\text{def}}(\rho)$; 2) $\hat{g}^p_{\text{env}}(f'(\rho)) = \hat{h}^p_{\text{env}}(\rho)$; and 3) $\hat{h}_{f'(\rho)} = \hat{h}_{f(\rho)}$. This can be shown straightforwardly by induction on the length of $w \in P^*$: if $w = u\sigma$, then $\hat{h}(w) = \hat{g}_{\hat{h}(u)}(\hat{f}(\rho)) = \hat{g}_{\hat{h}(u)}(f'(\rho))$.

Alg. 3 thus modifies T_f into a one-state machine with a single state q such that for all $\rho \in P$, $\omega_f(q, \rho) = w$ such that $\hat{g}^p_{\text{def}}(w) = h_{\text{def}}(\rho)$, $\hat{g}^p_{\text{env}}(w) = h_{\text{env}}(\rho)$, and $\hat{g}_w = \hat{g}_{\hat{f}(\rho)}$. All outgoing transitions from $\texttt{corr}(q_{\text{env}})$ are removed, as there are transitions whose output begins with w_τ instead of τ and so $\hat{g}^p_{\text{def}}(\omega_f(\texttt{corr}(q_{\text{env}}), \rho)) \neq \hat{h}^p_{\text{def}}(\rho)$. This leaves the outgoing transitions from $\texttt{corr}(q_{\text{def}})$, which, because D is a sufficient sample of $\hat{h}$, has a transition for each $\rho \in P$ such that $\hat{g}^p_{\text{def}}(\omega_f(\texttt{corr}(q_{\text{def}}), \rho)) = \hat{h}^p_{\text{def}}(\rho)$ and thus also that $\hat{g}^p_{\text{def}}(\omega_f(\texttt{corr}(q_{\text{def}}), \rho)) = \hat{h}^p_{\text{def}}(\rho)$. This makes it such that $f(T_f)$ will satisfy conditions (1) and (2) above.

However, there may be a ρ for whom $\hat{f}(\rho) = w$ is opaque under $\hat{g}$; that is, $\hat{g}_w \neq \hat{g}_{\hat{g}^p(w)}$. Alg. 3 can detect this by checking if the 1-suffix of w is not in the input suffixes of the state in T_g corresponding to the state in T_f reached by $\delta_f(q_{\text{def}}, \rho)$ (ln 3). As $\hat{g}$ is left-simplex ISL$_2$, this is exactly the case when $w = ue\tau$ for some u—i.e. that $\hat{g}(w) = u'w_\tau$ for some u'—and $\hat{g}_\tau \neq \hat{g}_{w_\tau}$. As the transition outputs of T_f are built from samples of $\hat{h}$, $\omega_f(\texttt{corr}(q_{\text{def}}), \rho) = u'w_\tau$. Line 6 thus changes this to $u'\tau$, which because $\hat{g}$ is ISL$_2$ has a tail function of $\hat{g}_\tau$. As in all other cases the tail function of $\omega_f(\texttt{corr}(q_{\text{def}}), \rho)$ is the same as its output under $\hat{g}$. Thus, $f(T_f)$ will satisfy condition (3) above.

Finally, the two states of T_f are merged, making $f(T_f)$ a homomorphism. Thus, $f(T_f)$ satisfies all conditions for $\hat{g} \circ f(T_f) = \hat{h}$, and because $T_g = T_{\hat{g}}$, so $f(T_g \circ T_g) = \hat{h}$. ■