Learning Process Interaction Through Simplex ISL Transducers

Chenli Wang and Adam Jardine
Rutgers University

Abstract

This paper investigates the learnability of interacting phonological processes by restricting the hypothesis space to a subregular class of functions. Interacting processes can be modeled as function composition, where the output of one function serves as the input to another. We focus specifically on interactions between two simplex Input Strictly Local (ISL_2) functions, a proper subclass of the ISL function class. We propose a decomposition algorithm that reconstructs both the individual component processes and their relative ordering by exploiting structural properties of simplex ISL_2 transducers and their compositions. This work provides an initial step toward understanding how learners can infer not only single phonological processes, but structured interactions between processes.

1 Introduction

This paper is motivated by a fundamental learning problem in natural language phonology: the acquisition of interacting functions. Previous work has established learning procedures for individual functions, including those characterized as subsequential functions (Oncina et al., 1993), and more efficient algorithms for subclasses of subsequential functions such as Input Strictly Local (ISL) and Output Strictly Local (OSL) functions (Chandlee, 2014; Chandlee et al., 2015; Jardine et al., 2014). Additional work has also developed algorithms that simultaneously infer both the mapping and the underlying representations from surface forms alone (Hua and Jardine, 2021).

However, many real-world phonological transformations are not realized by a single process. Phonological processes frequently interact when the application of one process affects the environment required by another, giving rise to both transparent and opaque interactions (Kiparsky, 1973; Baković, 2007). For example, in Votic (Ariste, 1968; Odden, 2005), raising and fronting of word-final mid vowels to [i] feeds palatalization of /k/.

(1) Feeding in Votic (Odden, 2005, p. 100)
a. /kurkə/ → kurki → [kurtʃi] ‘stork’
b. /əlʸkə/ → əlʸki → [əlʸtʃi] ‘straw’

In such cases, the observable mapping corresponds to a composed function: the output of the first function serves as the input to the second. The learners task, then, is to recover the component functions as well as their ordering.

In this work, we investigate the learnability of interacting functions that can be characterized as simplex ISL_2 functions, a subclass of the ISL function class, under subregular constraints. We frame the problem as one of function decomposition. Specifically, we assume that the learner has access to the output of OSTIA (Oncina et al., 1993), which is first used to learn a transducer representing the composed function. This assumption is justified because OSTIA can provably learn any subsequential function, and the class of subsequential functions is closed under composition. We then examine how to systematically decompose a function into its constituent functions.

Focusing on interactions between simplex ISL_2 functions, we introduce the Simplex ISL_2 Function Interaction Decomposition Algorithm (SI_2FIDA), which provably recovers both the individual functions and their relative ordering from the composed transducer. More broadly, this work contributes to our understanding of how locality constraints can be leveraged to learn structured compositions of functions. Indeed, this work generalizes the work of Hua and Jardine (2021) decomposing phonological functions. While Hua and Jardine show how to decompose the composition of a homomorphism and a simplex ISL_2 function, the current work shows how decomposition can be accomplished when both functions are sim-

plex ISL_2 functions.

The paper is structured as follows. Section 2 introduces the relevant notation and background concepts. Section 3 defines the relevant function classes. Section 4 establishes key properties of compositions of two simplex ISL_2 functions. Section 5 formalizes the learning problem. Section 6 presents the algorithm, demonstrates its operation, and proves its correctness, along with its data and time complexity. Section 7 concludes and discusses directions for future work.

2 Preliminaries

Let Σ be a finite alphabet. Σ^* denotes the set of all finite strings over Σ. The empty string is denoted by λ. For strings w and v, both wv and $w \cdot v$ denotes their connatenation. If $w = uv$ then $u^{-1} \cdot w = v$ and $w \cdot v^{-1} = u$. The set of *prefixes* of a string w is defined as: $\text{pref}(w) = \{u \in \Sigma^* \mid (\exists v \in \Sigma^*)[uv = w]\}$. The set of *suffixes* is defined as $\text{suff}(w) = \{v \in \Sigma^* \mid (\exists u \in \Sigma^*)[uv = w]\}$. The *k-prefix* of w is $\text{pref}_k(w) = u$ (respectively $\text{suff}_k(w) = u$) such that u is a prefix (respectively suffix) of w and $|u| = k$ if $|w| > k$, otherwise $u = w$. The notion of prefixes and suffixes can be extended to sets of strings. For example, $\text{pref}(S) = \bigcup_{w \in S} \text{pref}(S)$.

For a pair of strings w and v, let $w \wedge v$ denote their *longest common prefix*, which is a unique u s.t. $u \in \text{prefs}(w) \cap \text{prefs}(v)$ and for all $x \in \text{prefs}(w) \cap \text{prefs}(v), |u| \geq |x|$.

The function $f : \Sigma^* \to \Sigma^*$ maps strings to strings. As we are interested in functions that may be composed in either order, we assume that the input alphabet is the same as the output alphabet. $\text{dom}(f)$ and $\text{ran}(f)$ denote the domain and range of f respectively.

For f, $\mathring{f}(w)$ denotes the common output of w, which is defined as $\mathring{f}(w) \stackrel{\text{def}}{=} \bigwedge_{wu \in \text{dom}(f)} f(wu)$. For any $w \in \Sigma^*$, f_w denotes the tail function of w with respect to f, which is defined as $f_w(u) \stackrel{\text{def}}{=} \mathring{f}(w)^{-1} \cdot f(wu)$. Intuitively, $f_w(u)$ returns what the output of u is after removing the output we can attribute to w. $\mathring{f}_w$ denotes the tail prefix function, which is defined as $\mathring{f}_w \stackrel{\text{def}}{=} \bigwedge_{wu \cdot v \in \text{dom}(f)} f_w(uv)$. Intuitively, $\mathring{f}_w$ returns the common output of wu minus the common output of w.

3 Function Classes

A function f is said to be *(left-)subsequential* iff the set of its unique tail functions, $\{f_w \mid w \in \Sigma^*\}$, is finite (Schützenberger, 1977; Sakarovitch, 2009). A bold capital letter (e.g., $\mathbf{F}$) denotes the set of tail functions of f. Intuitively, each tail function f_w represents the behavior of f after processing some prefix w.

Subsequential functions are exactly those described by *subsequential finite-state transducers* (SFSTs), which are deterministic finite-state transducers with outputs on the states. An SFST T is a tuple $\langle Q, q_0, Q_\ltimes, \delta, \omega, \iota \rangle$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $Q_\ltimes \subseteq Q$ is the set of final states, $\delta : Q \times \Sigma \to Q$ is the transition function, $\omega : Q \times \Sigma \to \Sigma^*$ is the output function, and $\iota : Q_\ltimes \to \Sigma^*$ is the final-state output function. To define the function described by an SFST, we extend δ and ω to functions δ^* and ω^* on $Q \times \Sigma^*$ as follows: for $w = \lambda$ and any $q \in Q$, $\delta^*(q, \lambda) = q$ and $\omega^*(q, \lambda) = \lambda$; otherwise, for any $\sigma \in \Sigma$, $\delta^*(q, w\sigma) = \delta(\delta^*(q, w), \sigma)$ and $\omega^*(q, w\sigma) = \omega^*(q, w)\omega(\delta^*(q, w), \sigma)$. Then, for an SFST T, the function $f(T) : \Sigma^* \to \Sigma^*$ is defined as follows: $f(w)$ is defined iff $\delta^*(q_0, w) \in Q_\ltimes$, and in that case, $f(w) = \omega^*(q_0, w)\iota(\delta^*(q_0, w))$.

The states in an SFST for f correspond to the tail functions of f, that is, for each $q \in Q$, $f_q = f_w$ for any prefix w that reaches q (i.e., $\delta^*(q_0, w) = q$). An SFST is *onward* if for any $q \in Q$ and $\sigma \in \Sigma$, $\omega(q, \sigma) = \mathring{f}_q(\sigma)$. The *canonical* SFST for f is the minimal onward transducer for f. We can effectively generate the canonical SFST for f from any arbitrary SFST for f (Sakarovitch, 2009).

A more restricted subclass of subsequential functions is known as *Input Strictly Local* (ISL) functions (Chandlee, 2014).

Definition 1 (ISL)**.** A function f is *input strictly k-local (ISL$_k$)* iff for any $u, v \in \Sigma^*$, $\text{suff}_{k-1}(u) = \text{suff}_{k-1}(v)$ implies that $f_u = f_v$.

In other words, two prefixes u and v that share the same last $k-1$ symbols must induce the same tail function. Intuitively, an ISL_k function is one whose computation depends only on the most recent $k-1$ input symbols.

Every ISL function can be represented by a subsequential finite-state transducer whose states correspond to possible $(k-1)$-length suffixes of input strings. Formally, for an ISL_k function f, the set of tail functions is $\mathbf{F} = \{f_w \mid w \in \text{suff}_{k-1}(\Sigma^*)\}$.

That is, each state in the corresponding SFST represents the continuation behavior of f from a given $(k-1)$-symbol input context.

In order to make the learning problem tractable, we focus on ISL_k functions for $k = 2$. Consequently, the behavior of the function following a prefix w depends only on the final symbol of w (or λ if $w = \lambda$).

To further simplify the learning problem, and to model the notion of a single phonological process, we further narrow the focus to a subclass of the ISL_2 functions which make a single change.

Definition 2 (Left-simplex ISL_2 function (Hua and Jardine, 2021))**.** An ISL_2 function f is left-simplex iff it is properly ISL_2 and there is exactly one $f_{env} \in \mathbf{F}$ and $\tau \in \Sigma$ such that $\mathring{f}_{env}(\tau) \neq \tau$.

This class captures functions whose output depends only on the immediately preceding input symbol and permits exactly one symbol to be changed after exactly one class of prefixes. Henceforth we will refer to left-simplex ISL_2 functions simply as simplex ISL_2 functions.

Let e denote the *environment*, or the 1-suffix after which the change in f occurs (i.e., $f_e = f_{\text{env}}$). Likewise, we refer to $\tau \in \Sigma$ such that $f_{\text{env}}(\tau) = w_\tau \neq \tau$ as the target of f, with $w_\tau \in \Sigma^*$ representing the output of τ. Finally, let f_{def} denote the default tail function, which maps every symbol to itself. An example of a transducer that captures a simplex ISL_2 function is given in Fig. 1.

Example 1. Suppose a simplex ISL_2 function f over $\Sigma = \{a, b, c\}$ that maps a to b when it follows c, while all other strings map to themselves. In this case, $e = c$, $\tau = a$, and $w_\tau = b$; i.e. $g_c(a) = b$. Accordingly, $f_{\text{env}} = f_c$, and $f_{\text{def}} = f_\lambda = f_a = f_b$, representing the tail function associated with all other 1-suffixes. Thus, a unique simplex ISL_2 function can be fully specified once e, τ, and w_τ are identified.

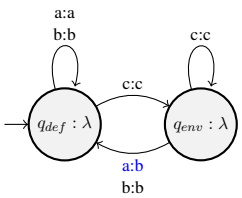


Figure 1: A simplex ISL_2 SFST for f from Ex. 1.

The following remark about the structure of SFSTs for simplex ISL_2 functions straightforwardly follows the from the definition of simplex ISL_2 functions. For a state $q \in Q$ of a transducer T, let its set of *input* 1-*suffixes*

$$\texttt{IS}_1(q) = \texttt{suff}_1(\{w \mid \delta^*(q_0, w) = q\})$$

be the set of 1-suffixes in the input seen before reaching that state.

Remark 1. For the canonical SFST for any simplex ISL_2 function:

1. There are exactly two states $\{q_{\text{def}}, q_{\text{env}}\}$.
2. $|\,\text{IS}_1(q_{\text{env}})| = 1$; if $\text{IS}_1(q_{\text{env}}) = \{\lambda\}$, then $q_{\text{env}} = q_0$ (q_{env} is the initial state). Otherwise, there is exactly one symbol σ such that $\delta(q, \sigma) = q_{\text{env}}$ for any $q \in \{q_{\text{def}}, q_{\text{env}}\}$. Thus $\text{IS}_1(q_{\text{env}}) \cap \text{IS}_1(q_{\text{def}}) = \emptyset$.
3. There is exactly one symbol τ for which $\omega(q_{\text{env}}, \tau) \neq \tau$. For any other state q and symbol σ, $\omega(q, \sigma) = \sigma$.

The SFST in Figure1 is a canonical SFST for the function f in Ex. 1.

4 Composition of input-strictly local functions

4.1 Interacting simplex ISL_2 functions

The primary interest of this paper is the composition of simplex ISL_2 functions. Two functions f and g are said to *interact* when $g(f(x)) \neq f(g(x))$ for at least one x. Below is an example of two interacting functions.

Example 2. Consider two simplex ISL_2 functions, f and g, with tail functions $\mathring{f}_c(a) = b$ and $\mathring{g}_b(d) = b$. These two functions interact because their order of application yields different results. If f is ordered first, it creates the environment for g to apply in the string *cad*, producing the derivation: $cad \xrightarrow{f} cbd \xrightarrow{g} cbb$. In contrast, when g applies first, the input *cad* is mapped to *cbd*.

When both f and g are restricted to simplex ISL_2 functions, their interaction occurs when one function either creates or removes the environment required by the other. This can be formalized in terms of tail functions as follows.

Remark 2. If $g \circ f \neq f \circ g$, then at least one of the following two situations holds:

1. There is some prefix w whose tail function $g_w \in \mathbf{G}$ of g is such that $g_{\mathring{f}(w)} \neq g_w$ or whose tail function $f_w \in \mathbf{F}$ of f is such that

$f_{\mathring{g}(w)} \neq f_w$. That is, either f affects some tail function of g or g affects some tail function of f.

2. There are some w and σ for whom $g_{\mathring{f}(w)}(f_w(\sigma)) \neq f_w(\sigma)$ or $f_{\mathring{g}(w)}(g_w(\sigma)) \neq g_w(\sigma)$. That is, either g 'obscures' an output of f or vice versa.

Example 3. Let f and g be as in Example 2. Because $g(cad) = cad$ (i.e., identity) but $g(f(cad)) = cbb$ (i.e., the change affected by g has occured), this means that the tail function of ca for g is distinct from the tail function of $\mathring{f}(ca) = cb$, as $g_{ca}(d) = d \neq g_{\mathring{f}(ca)}(d) = b$.

Because the present problem concerns learning from the composition of two functions f and g, we must limit ourselves to cases in which each the application of individual function can be observed from their composition. We call this *recoverable*.

Definition 3 (Recoverable)**.** For two functions f and g, we say that each is *recoverable* from $h = g \circ f$ iff there is some x and some y such that $f(x) = h(x)$ and $g(y) = h(y)$.

Non-recoverable cases are exactly those in which both f and g have the same environment and the change in f serves as the target of g, as shown in Example 4.

Example 4 (A non-recoverable function)**.** Consider the simplex ISL_2 functions f and g in which the environment tail function for f is $\mathring{f}_b(a) = c$ and $\mathring{g}_b(c) = d$. In this case, both f and g share the same environment (b) *and* the output of the change of f (c) is exactly the target of g. Thus, while $f(ba) = bc$, $g(f(ba)) = bd$. In fact, there is no string x such that $f(x) = g \circ f(x)$, as $g \circ f$ will change any a following a b directly into a d. Thus, in $g \circ f$ there is no way to observe that f changes a a into a c.

We henceforth are only concerned with interactions in which both f and g are recoverable from $g \circ f$ and $f \circ g$. We can make the following observation, which formalizes the above discussion about the nature of interaction.

Lemma 1. For two interacting simplex ISL_2 functions f and g, let e' and τ' be the environment and target of f, respectively, and let e'' and τ'' be the environment and target for g. Let $s' = \text{suff}_1(\mathring{f}(e'\tau'))$; that is, the 1-suffix after seeing the output of the change in f. Define s'' similarly for g.

If f and g are both recoverable from $g \circ f$ and $f \circ g$, then one of the following is true: 1) $e' = s''$; 2) $e'' = s'$; 3) $e' = \tau''$; or 4) $e'' = \tau'$.

Proof. From Remark 2 and the fact that both f and g are recoverable, if $g \circ f \neq f \circ g$ then it must be the case that there is some w such that either $g_w \neq g_{\mathring{f}(w)}$ or $f_w \neq f_{\mathring{g}(w)}$. (Remark 2-2 cannot be true because if f and g are recoverable, then one cannot obscure the output of the other.)

We focus on the case in which $g_w \neq g_{\mathring{f}(w)}$. Since g is simplex and thus only one non-identity change is allowed, then either $g_w = g_{e''}$ or $g_{\mathring{f}(w)} = g_{e''}$. That is, one must be the environment tail function, as one must have some non-identity output for some input symbol σ. Furthermore, they must disagree on the target τ''. Therefore either $g_w = g_{e''}$ and $g_{e''}(\tau'') \neq g_{\mathring{f}(w)}(\tau'')$ or $g_{\mathring{f}(w)} = g_{e''}$ and $g_w(\tau'') \neq g_{e''}(\tau'')$.

Since f itself is only allowed to change τ' to s' after seeing environment e', then if $w \neq \mathring{f}(w)$ then $e'\tau'$ must be a suffix of w and s' must be a suffix of $\mathring{f}(w)$. Thus if $g_w \neq g_{\mathring{f}(w)}$ either $\tau' = e''$ or $s' = e''$. This establishes cases (2) and (4) above. The same logic applied to the case in which $f_w \neq f_{\mathring{g}(w)}$ establishes cases (1) and (4). □

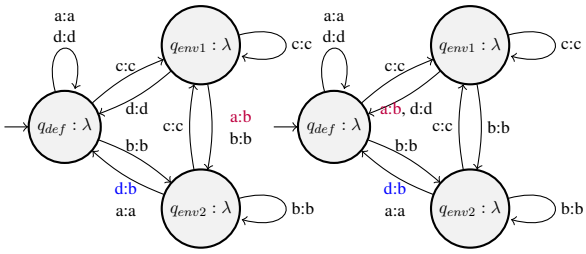
4.2 Interaction and composition of SFSTs

The composition of two subsequential functions can be viewed through the composition operation on their respective SFSTs. For two SFSTs T' and T'', their composition is given by the following from Hua and Jardine (2021), which in turn was adapted from Lothaire (2005).

Definition 4 (Composition of SFSTs)**.** For two SFSTs $T' = \langle Q', q'_0, Q'_\ltimes, \delta', \omega', \iota' \rangle$ and $T'' = \langle Q'', q''_0, Q''_\ltimes, \delta'', \omega'', \iota'' \rangle$, $T'' \circ T' := \langle Q' \times Q'', (q'_0, q''_0), Q'_\ltimes \times Q''_\ltimes, \delta, \omega, \iota \rangle$ where

$$\delta((q', q''), \sigma) := (\delta'(q', \sigma), \delta''^*(q'', \omega'(q', \sigma)));$$
$$\omega((q', q''), \sigma) := \omega''^*(q'', \omega'(q', \sigma)); \text{ and}$$
$$\iota((q'_\ltimes, q''_\ltimes)) := \omega''^*(q''_\ltimes, \iota'(q'_\ltimes)).$$

The compositions $g \circ f$ and $f \circ g$ from Example 3 are shown in Figure 2 as three-state onward SFSTs. Although the construction in Definition 4 yields four states, the fourth state is unreachable from the initial state and is therefore omitted.



(a. $T_{g \circ f}$**)** **(b.** $T_{f \circ g}$**)**

Figure 2: Composition

The following states some important properties of the canonical SFST representing the composition of any two left-simplex ISL_2 functions. Note that any ISL_k class is not closed under composition (Chandlee, 2014) and therefore the composed transducer may not be ISL_2. However, this does not affect the success of our algorithm.

Lemma 2. Let f and g be two simplex ISL_2 functions. Let $\tau' \in \Sigma$ be the one symbol such that $f_{\text{env}}(\tau') \neq \tau'$ and similarly $\tau'' \in \Sigma$ be the one symbol such that $g_{\text{env}}(\tau'') \neq \tau''$. (It may be the case that $\tau' = \tau''$.) The following is true for the canonical SFST for $T_{g \circ f}$:

1. There is exactly one state $q_{\text{def}} \in Q$ for which $\omega(q_{\text{def}}, \sigma) = \sigma$ for all $\sigma \in \Sigma$.

2. Out of the remaining states, there are at most two states that are either the initial state or are reachable from q_{def} by a path one transition long. Call them q_{env1} and q_{env2}(it may be the case that $q_{\text{env1}} = q_{\text{env2}}$).

3. For one $q' \in \{q_{\text{env1}}, q_{\text{env2}}\}$, $\omega(q, \tau') = f_{env}(\tau')$ and similarly for one $q'' \in \{q_{\text{env1}}, q_{\text{env2}}\}$, $\omega(q, \tau') = g_{env}(\tau'')$.

Proof. Let T' be the canonical SFST for f and T'' be the canonical SFST for g. The set of states for T' and T'' are $\{q'_{\text{def}}, q'_{\text{env}}\}$ and $\{q''_{\text{def}}, q''_{\text{env}}\}$, respectively. Thus, by Definition 4 of SFST composition, the set of states for $T'' \circ T'$ is, maximally, $\{(q'_{\text{def}}, q''_{\text{def}}), (q'_{\text{def}}, q''_{\text{env}}), (q'_{\text{env}}, q''_{\text{def}}), (q'_{\text{env}}, q''_{\text{env}})\}$. Lemma 2-1 holds for $(q'_{\text{def}}, q''_{\text{def}})$ and only $(q'_{\text{def}}, q''_{\text{def}})$, as any state corresponding to q'_{env} or q''_{env} will have at least one outgoing non-identity transition.

Lemma 2-2 holds because each of q'_{env} and q''_{env} can be reached from q'_{def} and q''_{def}, respectively, by exactly one 1-suffix. For each of these states, either this 1-suffix is λ (and thus either q'_{env} or q''_{env} is the start state) or it is some σ such that $\delta'(q'_{\text{def}}, \sigma) = q'_{\text{env}}$ or $\delta''(q''_{\text{def}}, \sigma) = q''_{\text{env}}$. Thus, by the definition of composition, $q_{\text{env1}} = (q'_{\text{env}}, q''_{\text{def}})$ and $q_{\text{env2}} = (q'_{\text{def}}, q''_{\text{env}})$ are both reachable from q_{def} by a single transition.

In contrast, if a fourth state $q_{\text{extra}} = (q'_{\text{env}}, q''_{\text{env}})$ exists, the path from q_{def} to q_{extra} is at minimum two transitions. In other words, q_{extra} is only accessible from q_{env1} or q_{env2}. By the definition of simplex ISL_2, there is only one transition from q'_{def} to q'_{env}. Let σ be the single symbol in Σ such that $\delta'(q'_{\text{def}}, \sigma) = q'_{\text{env}}$. By the definition of simplex ISL_2, any transition out of q'_{def} must be an identity transition, so it is also the case that $\omega'(q'_{\text{def}}, \sigma) = \sigma$. Thus, by the definition of composition, $\delta(q_{\text{def}}, \sigma) = \delta((q'_{\text{def}}, q''_{\text{def}}), \sigma) = (\delta'(q'_{\text{def}}, \sigma), \delta''^{*}(q''_{\text{def}}, \sigma)) = (q'_{\text{env}}, q'')$ for some q'' such that either $q'' = q''_{\text{def}}$ or $q'' = q''_{\text{env}}$. If $q'' = q''_{\text{def}}$, then the only transition to $q_{\text{extra}} = (q'_{\text{env}}, q''_{\text{env}})$ must be $\delta((q'_{\text{env}}, q''_{\text{def}}), \sigma)$ and thus any string must first traverse $(q'_{\text{env}}, q''_{\text{def}})$ before reaching q_{env}.

In the other case that $q'' = q''_{\text{env}}$, then both $\delta'(q'_{\text{env}}, \sigma) = q'_{\text{env}}$ and $\delta''(q''_{\text{def}}, \sigma) = q''_{\text{env}}$. However, then for any distinct $\tau \neq \sigma \in \Sigma$, then $\delta'(q'_{\text{env}}, \tau) = q'_{\text{def}}$ and $\delta''(q''_{\text{env}}, \tau) = q''_{\text{def}}$, and so all other transitions in T lead back to q_{def}. Thus if $q'' = q''_{\text{env}}$, there cannot be four distinct states (in fact there can only be two). □

To clarify Lemma 2-3, q_{extra} is a distinct fourth state exactly in cases where both processes must apply, but the non-identity transition exiting the first environment state cannot directly enter the second environment state. In such configurations, the extra state mediates the interaction.

Lemma 3. Assume τ', τ'', T_f, and T_g as in Lemma 2. Let e' be the unique 1-suffix that reaches q'_{env} in T_f and e'' be the unique 1-suffix that reaches q''_{env} in T_g. Let $s' = \text{suff}_1(\omega'^{*}(q'_0, e'\tau'))$; that is, the 1-suffix seen in the output after seeing both the environment and target for T_f. Similarly, let and $s'' = \text{suff}_1(\omega''^{*}(q''_0, e'\tau''))$.

The following hold of s', e', and e''.

1. If $s' = e''$, then $\delta(q_{\text{env1}}, \tau') = q_{\text{env2}}$.

2. If $e' = \tau''$, then $\delta(q_{\text{env2}}, \tau'') = q_{\text{env1}}$.

3. If neither is true, then $f \circ g = g \circ f$.

Proof. Points (1) and (2) follow directly from Lemma 1 and the definition of composition. Point (3) follows from Lemma 1. □

The Algorithm in §6 uses the properties of Lemmas 2 and 3 to identify the individual functions f and g.

5 Learning Paradigm

Informally, the learning goal is to identify two functions, f and g, along with their ordering, from a transducer obtained via OSTIA that represents their composition, under the assumption that each function is a simplex ISL_2 function. Formally, we adopt de la Higuera (2010)'s identification in the limit from polynomial time and data. Under this paradigm, the goal is to identify an algorithm that, given a sufficiently large positive sample for a specified class of functions, can recover a representation of each target function and their ordering in polynomial time. Such a sufficient sample must be of polynomial size relative to the size of the functions representation.

Consider a class $\mathbb{T}$ of functions represented by a class $\mathbb{R}$ of finite representations, there exists a total, surjective mapping $N : \mathbb{R} \to \mathbb{T}$ from representations $r \in \mathbb{R}$ to functions in $\mathbb{T}$.

Definition 5 (Sample)**.** For a function $t \in \mathbb{T}$, a sample S of t is a finite set of pairs (w, v) such that $t(w) = v$ for all $(w, v) \in S$. The size of a sample S is defined as $|S| = \sum_{(w,v)\in S} (|w| + |v|)$.

Definition 6 ($(\mathbb{T}, \mathbb{R})$-Learning Algorithm)**.** A $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathcal{A}$ is a program that takes as input a sample S for some $t \in \mathbb{T}$ and returns a representation $r \in \mathbb{R}$.

Here the class $\mathbb{R}$ consists of pairs of Simplex ISL_2 transducers (T_f,T_g). The mapping N from representation to target functions is $N(T_f, T_g) \stackrel{\text{def}}{=} f(T_g) \circ f(T_f)$.

Definition 7 (Strong characteristic sample)**.** For a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathcal{A}$, a sample CS is a *strong characteristic sample* of a representation $r \in \mathbb{R}$ if for all samples S for $\mathcal{L}(r)$ such that $CS \subseteq S$, $\mathcal{A}$ returns r.

Definition 8 (Strong identification in polynomial time and data (de la Higuera, 2010))**.** A class $\mathbb{T}$ of functions is *strongly identifiable in polynomial time and data* if there exists a $(\mathbb{T}, \mathbb{R})$-learning algorithm $\mathcal{A}$ and two polynomials p and q such that, for any characteristic sample S of size m for $r \in \mathbb{R}$, $\mathcal{A}$ returns a hypothesis $r \in \mathbb{R}$ in $O(p(m))$ time, and for each representation $r \in \mathbb{R}$ of size k, there exists a strong characteristic sample of r for $\mathcal{A}$ of size at most $O(q(k))$.

6 Algorithm

In this section we give the Simplex ISL2 Function Interaction Decomposition Algorithm (SI_2FIDA). Since both f and g are subsequential functions, their composition $g \circ f$ is also subsequential. Consequently, $g \circ f$ can be learned in the limit from positive data in cubic time using the Onward Subsequential Transducer Inference Algorithm (OSTIA; (Oncina et al., 1993). The inference procedure therefore begins with the transducer returned by OSTIA, which is constructed from input-output pairs representing f, g, and their composition.

Algorithm 1: SI_2FIDA

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

$T_h := \text{OSTIA}(D) = \langle Q_h = \{q_1, ..., q_n\} = Q_{h\rtimes},\ q_{0_h},\ \delta_h,\ \omega_h,\ \iota_h\rangle$;

$IS(q) = \{suff_1(\rho \text{ s.t } \delta_h(q, \rho) = q)\}$;

$q_{def} \leftarrow q$ s.t. $\forall\sigma \in \Sigma, \omega(q, \sigma) = \sigma$;

$q_{extra} \leftarrow q$ s.t. $\exists(\sigma, \sigma' \in \Sigma, q' \neq q_{\text{def}} \neq q)$ s.t. $\delta(q_{\text{def}}, \sigma) = q'$ and $\delta(q', \sigma') = q$ and $\neg\exists\sigma$ s.t. $\delta(q_{\text{def}}, \sigma) = q$;

if $|(Q_h - \{q_{extra}\}| = 3$ **then**

  $q_{env_1} \leftarrow q \in Q_h - \{q_{extra}\}$ s.t $q \neq q_{def}$;

  $q_{env_2} \leftarrow q \in Q_h - \{q_{extra}\}$ s.t $q \neq q_{def} \wedge q \neq q_{env_1}$

if $|(Q_h - \{q_{extra}\}| = 2$ **then**

  $q_{env_1} \leftarrow q \in Q_h - \{q_{extra}\}$ s.t $q \neq q_{def}$;

  $q_{env_2} \leftarrow q \in Q_h - \{q_{extra}\}$ s.t $q \neq q_{def}$

$T_f, T_g := \texttt{construct_}T_fT_g(T_h)$

return $\texttt{order_}T_fT_g(T_h, T_f, T_g)$

Algorithm 1 takes as input a finite set of inputoutput pairs. The first step is to construct a transducer that represents the composition of two functions using OSTIA. The algorithm then identifies the state whose outgoing transitions are all identity transitions as the default state, denoted q_{def}. Next, it identifies q_{extra}, the state that is reachable from q_{def} only via a path of length two. If removing q_{extra} leaves three states in the transducer, the two remaining non-default states are denoted q_{env_1} and q_{env_2}, respectively. If removing q_{extra} leaves only two states, then both q_{env_1} and q_{env_2} refer to the unique non-default state in T_h. The

Algorithm 2: construct_$T_f T_g$

Data: $T_h := \text{OSTIA}(D) = \langle Q_h = Q_{h\rtimes},\ q_{0_h},\ \delta_h,\ \omega_h,\ \iota_h \rangle$

1 **if** $q_{0_h} = q_{env1}$ *or* $q_{0_h} = q_{env2}$ **then**
2 $\quad T_f := \langle Q' = \{q'_{def}, q'_{env}\} = Q'_{\rtimes},\ q'_0 = q_{0_h},\ \delta',\ \omega',\ \iota' \rangle$;
3 $\quad T_g := \langle Q'' = \{q''_{def}, q''_{env}\} = Q''_{\rtimes},\ q''_0 = q_{def},\ \delta'',\ \omega'',\ \iota'' \rangle$;
4 **else**
5 $\quad T_f := \langle Q' = \{q'_{def}, q'_{env}\} = Q'_{\rtimes},\ q'_0 = q_{0_h},\ \delta',\ \omega',\ \iota' \rangle$;
6 $\quad T_g := \langle Q'' = \{q''_{def}, q''_{env}\} = Q''_{\rtimes},\ q''_0 = q_{0_h},\ \delta'',\ \omega'',\ \iota'' \rangle$;
7 **for** $\sigma \in \Sigma$ **do**
8 $\quad$ // Establish transitions in T_f for σ ;
9 $\quad$ **if** $\delta_h(q_{def}, \sigma) = q_{env_1}$ **then** $\delta'(q'_{def}, \sigma) := q'_{env}$;
10 $\quad$ **else** $\delta'(q'_{def}, \sigma) := q'_{def}$;
11 $\quad$ **if** $\delta_h(q_{env_1}, \sigma) = q_{env_1}$ **then** $\delta'(q'_{env}, \sigma) := q'_{env}$;
12 $\quad$ **if** $\delta_h(q_{env_1}, \sigma) = q_{def}$ **then** $\delta'(q'_{env}, \sigma) := q'_{def}$;
13 $\quad$ **if** $\delta_h(q_{env_1}, \sigma) = q_{extra}$ **then**
14 $\quad\quad$ **if** $\sigma = IS(q_{env_1})$ **then** $\delta'(q'_{env}, \sigma) := q'_{env}$;
15 $\quad\quad$ **else** $\delta'(q'_{env}, \sigma) := q'_{def}$;

Algorithm 2: construct_$T_f T_g$ (continued)

16 **for** $\sigma \in \Sigma$ **do**
17 $\quad$ // Establish transitions in T_g for σ ;
18 $\quad$ **if** $\delta_h(q_{def}, \sigma) = q_{env_2}$ **then** $\delta''(q''_{def}, \sigma) := q''_{env}$;
19 $\quad$ **else** $\delta'(q''_{def}, \sigma) := q''_{def}$;
20 $\quad$ **if** $\delta_h(q_{env_2}, \sigma) = q_{env_2}$ **then** $\delta''(q''_{env}, \sigma) := q''_{env}$;
21 $\quad$ **if** $\delta''(q''_{env}, \sigma) := q''_{def}$ **then** $\delta''(q''_{env}, \sigma) := q''_{def}$;
22 $\quad$ **if** $\delta_h(q_{env_2}, \sigma) = q_{extra}$ **then**
23 $\quad\quad$ **if** $\sigma = IS(q_{env_2})$ **then** $\delta''(q''_{env}, \sigma) := q''_{env}$;
24 $\quad\quad$ **else** $\delta''(q''_{env}, \sigma) := q''_{def}$;
25 $change' = (\tau', s')$ s.t. $\omega_h(q_{env_1}, \tau') = s'$ and $\tau' \neq s'$;
26 $change'' = (\tau'', s'')$ s.t. $\omega_h(q_{env_2}, \tau'') = s''$ and $\tau'' \neq s''$;
27 **for** $\sigma \in \Sigma$ **do**
28 $\quad$ **for** $q \in Q'$ **do** $\omega'(q', \sigma) = \sigma$; $\omega'(q'_{env}, \tau') = s'$
29 $\quad$ **for** $q \in Q''$ **do** $\omega''(q'', \sigma) = \sigma$;
30 $\iota'(q'_{def}) = \iota'(q'_{env}) = \iota''(q''_{def}) = \iota''(q''_{env}) = \lambda$
31 **return** T_f, T_g

next step is to decompose this composed transducer into the two underlying component transducers and determine their relative ordering.

Algorithm 2 shows how two transducers, T_f and T_g, are constructed from the composed transducer T_h. The algorithm takes the OSTIA output as input and initializes two simplex ISL_2 transducers in Lines 1-7. Under the simplex ISL_2 assumption, each transducer has a canonical structure consisting of exactly two states: an environment state and a default state. q_{env1} and q_{env2} map to the environment state of each individual machine. If the initial state of the composed machine corresponds to one of the environment states, then the default state in the composed machine maps to the initial state of one individual machine. Otherwise, the initial state of the composed machine functions as the default state for both individual machines.

Lines 8–16 compute the transition function for T_f. Under the simplex ISL_2 assumption, the individual transducer contains two types of transitions: transitions exiting the default state and transitions exiting the environment state. All such transitions in the individual machine can be recovered from the transition behavior of the composed machine. For transitions exiting the default state, if reading σ in T_h causes a transition from the q_{def} to q_{env1}, then the corresponding transition in T_f also goes from the q'_{def} to q'_{env}. Otherwise, the transitions exiting the default state in T_f remain in the default state. Transitions exiting the environment state have two possible outcomes as well: they either loop on the environment state or transition to the default state. If reading σ in the composed machine leads to a transition that loops on q_{env1}, then the corresponding transition in T_f also loops on q'_{env1}. If the composed machine transitions from q_{env1} to q_{def}, then the corresponding transition in T_f also goes from q'_{env} to q'_{def}. Finally, if the composed machine transitions from q_{env1} to q_{extra}, then the transition in T_f loops on q'_{env} if σ matches the input suffix associated with

q_{env1}. All other transitions from q'_{env} transition to q'_{def}. Lines 17-25 then compute the transition function for T_g in the same way.

Lines 26–27 identify the sole non-identity transition in each machine. Under the simplex ISL_2 assumption, non-identity outputs occur only on transitions exiting the environment state. Lines 28–32 then construct the output functions for the two individual machines. By definition, aside from this single non-identity output, all other outputs are identity mappings.

Algorithm 3 determines the ordering once the two individual machines have been derived. Lines 1–6 identify the environment, trigger, and output associated with each machine. Lines 7–12 detect a potential feeding interaction, which is when the environment of one function corresponds to the output of the other. In these cases, if the transition from the environment state goes to the default state, the environment-affecting function is ordered second; otherwise, it is ordered first. Lines 13–15 detect a potential bleeding interaction, where the environment of one function corresponds to the input of the other. In these cases, if the transition from the environment state goes to the other environment state, the environment-affecting function is ordered second; otherwise, it is ordered first.

Algorithm 3: order_T_fT_g

Data: T_h, T_f, T_g

1 $env' \leftarrow$ the unique $e_1 \in \text{IS}_1(q'_{env})$
2 $env'' \leftarrow$ the unique $e_2 \in \text{IS}_1(q''_{env})$
3 $\tau' \leftarrow$ the unique symbol s.t. $\omega'(q'_{env}, \tau') = w'_\tau$, for $w'_\tau \neq \tau'$
4 $s' \leftarrow \text{suff}_1(env'w'_\tau)$
5 $\tau'' \leftarrow$ the unique symbol s.t. $\omega''(q''_{env}, \tau'') = w''_\tau$, for $w''_\tau \neq \tau''$
6 $s'' \leftarrow \text{suff}_1(env''w''_\tau)$
7 **if** $env'' = s'$ **then**
8 **if** $\delta_h(q_{env_1}, \tau') = q_{def}$ **then return** (T_g, T_f)
9 **else return** (T_f, T_g)
10 **if** $env' = s''$ **then**
11 **if** $\delta_h(q_{env_2}, \tau'') = q_{def}$ **then return** (T_f, T_g)
12 **else return** (T_g, T_f)
13 **if** $env'' = \tau'$ **then**
14 **if** $\delta_h(q_{env_1}, \tau') = q_{env_2}$ **then return** (T_g, T_f)
15 **else return** (T_f, T_g)
16 **if** $env' = \tau''$ **then**
17 **if** $\delta_h(q_{env_2}, \tau') = q_{env_1}$ **then return** (T_f, T_g)
18 **else return** (T_g, T_f)
19 **return** (T_f, T_g)

7 Correctness and Complexity

In the following, assume T_f and T_g are the canonical SFSTs for two simplex ISL_2 functions f and g, respectively. Clearly, then, (T_f, T_g) is a unique canonical grammar for $g \circ f$. The below shows how the above algorithm identifies (T_f, T_g) from a sample of $g \circ f$.

The CS for OSTIA for a transducer T is essentially any set of strings that 1) reach each state; 2) exercise each transition; and 3) provide enough information to correctly calculate the longest common prefix of the output on each transition. We refer the reader to Oncina et al. (1993) for details. What is important here is that any CS for OSTIA for T returns a canonical SFST for T.

Lemma 4. The CS for OSTIA for $T_{g\circ f}$ is a CS for (T_f, T_g).

Proof. The CS for OSTIA for $T_{g\circ f}$ generates the canonical transducer for $T_{g\circ f}$ (Oncina et al., 1993). The correctness of the present algorithm then follows from Lemmas 2 and 3. Algorithm 1 identifies the states of two machines T' and T'' in $T_{g\circ f}$ exactly based on the structure of $T_{g\circ f}$ identified in Lemma 2 and Algorithm 2 builds the states and transitions of T' and T'' accordingly. Algorithm 3 then establishes which of T' and T'' is T_f and which is T_g based on the information about environments and targets available in $T_{g\circ f}$ as established in (1) and (2) of Lemma 3. □

Lemma 5. The size of the CS for SI_2FIDA is linear in the size of the grammar.

Proof. The size of the CS for OSTIA for any T is linear in the size of T (Oncina et al., 1993). □

Lemma 6. SI_2FIDA runs in time cubic with respect to the input data.

Proof. OSTIA runs in cubic time (Oncina et al., 1993). The `for` loops Algorithm 3 loop over the symbols in Σ, thus the steps in Algorithms 1 through 3 are constant with respect to the data. □

Theorem 1. SI_2FIDA strongly identifies T_f T_g from the limit in polynomial time and data.

Proof. From Lemmas 4 through 6. □

8 Discussion and Conclusion

Given that process interaction is pervasive in natural language phonology, a child learner must be able to infer not only individual processes but also their interaction. The present work establishes an initial inference procedure for recovering two interacting simplex ISL_2 functions from their composition. This thus takes steps towards solving the heretofore unsolved problem of learning interacting processes from data.

The problem of learning interacting processes is particularly important because process interaction is pervasive in natural language phonology. Chinese tone sandhi systems, for example, exhibit abundant interactions between tonal processes. In the Changting dialect, there are five lexical toneshigh, mid, low, rising, and fallingand 15 of the 25 possible disyllabic tone combinations undergo tone sandhi. Because the tonal inventory is limited, the application of one process frequently affects the environment required by another, giving rise to abundant interactions. Changting exhibits both transparent and opaque interactions. Furthermore, each individual tone sandhi process can be characterized as a simplex ISL_2 ($sISL_2$) function, which is the type of interaction investigated in this study.

The present algorithm exploits the canonical two-state structure of simplex ISL_2 transducers and the constrained form of their composition. A natural open question is thus how to generalize this result. First, the inference procedure should be generalized beyond $k = 2$.

A second direction involves extending the analysis to any arbitrary number of functions. In natural language phonology, the observable mapping may reflect the serial application of multiple, potentially interacting processes. Although subsequential functions are closed under composition, the corresponding decomposition problem is likely to become increasingly complex as the number of component functions grows. However, the current work shows that a strong definition of what it means to be an 'individual process' gives the learner sufficient structure to disentangle function interactions.

Acknowledgements

The authors thank the helpful suggestions of three anonymous reviewers. Both authors were supported in this work by NSF grant #2416184.

References

Paul Ariste. 1968. *A Grammar of the Votic Language*. Bloomington: Indiana University Press.

Eric Baković. 2007. A revised typology of opaque generalisations. *Phonology*, 24(2):217259.

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

Jane Chandlee, Rémi Eyraund, and Jeffery Heinz. 2015. Output strictly local functions. *Proceedings of the 14th Meeting on the Mathematics of Language*.

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

Wenyue Hua and Adam Jardine. 2021. Learning input strictly local functions from their composition. In *ICGI 15*, volume 153, pages 47–65. PMLR.

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 12th International Conference on Grammatical Inference (ICGI 2014)*, JMLR Workshop Proceedings, pages 94–108.

Paul Kiparsky. 1973. Abstractness, opacity, and global rules. In Osamu Fujimura, editor, *Three dimensions in linguistic theory*, pages 57–86. Tokyo: TEC.

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

David Odden. 2005. *Introducing Phonology*. Cambridge University Press.

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

Jacques Sakarovitch. 2009. *Elements of Automata Theory*. Cambridge University Press.

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