Learning Tier-based Strictly 2-Local languages
(draft, July 2, 2015)

Adam Jardine and **Jeffrey Heinz**
University of Delaware
{ajardine,heinz}@udel.edu

Abstract

The Tier-based Strictly 2-Local (TSL_2) languages are a class of formal languages which have been shown to model long-distance phonotactic generalizations in natural language (Heinz et al., 2011). This paper introduces the Tier-based Strictly 2-Local Inference Algorithm (2TSLIA), the first non-enumerative learner for the TSL_2 languages. We prove the 2TSLIA is guaranteed to converge in polynomial time on a data sample whose size is bounded by a constant.

1 Introduction

This work presents the Tier-based Strictly 2-Local Inference Algorithm (2TSLIA), an efficient learning algorithm for a class of *Tier-based Strictly Local* (TSL) formal languages (Heinz et al., 2011). A TSL class is determined by two parameters: the *tier*, or subset of the alphabet, and the *permissible tier k-factors*, which are the legal sequences of length k allowed in the string, once all non-tier symbols have been removed. The Tier-based Strictly 2-Local (TSL_2) languages are those in which $k = 2$.

As will be discussed below, the TSL languages are of interest to phonology because they can model a wide variety of long-distance phonotactic patterns found in natural language (Heinz et al., 2011; McMullin and Hansson, to appear). One example is derived from Latin liquid dissimilation, in which two *l*s cannot appear in a word unless there is an *r* intervening, regardless of distance. For example, *floralis* 'floral' is well-formed but not **militalis* (cf. *militaris* 'military'). As explained in sections 2 and 4, this can be modeled with permissible 2-factors over a tier consisting of the liquids $\{l, r\}$.

For long-distance phonotactics, k can be fixed to 2, but it is does not appear that T can be fixed since languages employ a variety of different tiers. This presents an interesting learning problem: Given a fixed k, how can an algorithm induce both a tier and a set of permissible tier k-factors from positive data?

There is some related work which addresses this question. Goldsmith and Riggle (2012), building on work by Goldsmith and Xanthos (2009), present a method based on mutual information for learning tiers and subsequently learning harmony patterns. This paper differs in that its methods are rooted firmly in grammatical inference and formal language theory (de la Higuera, 2010). For instance, in contrast to the results presented there, we prove the kinds of patterns 2TSLIA succeeds on and the kind of data sufficient for it to do so.

Nonetheless, there is relevant work in computational learning theory: Gold (1967) proved that any finite class of languages is identifiable in the limit via an enumeration method. Given a fixed alphabet and a fixed k, the number of possible tiers and permissible tier k-factors is finite, and thus learnable in this way. However, such learners are grossly inefficient. No provably-correct, non-enumerative, efficient learner for both the tier and permissible tier k-factor parameters has previously been proposed. This work fills this gap with an algorithm which learns these parameters when $k = 2$ from positive data in time polynomial in the size of the data.

Finally, Jardine (to appear) presents a simplified version of 2TSLIA and reports the results of some simulations. Unlike that paper, the present work provides the full mathematical details and proofs. The simulations are discussed in the discussion section.

This paper is structured as follows. §2 motivates the work with examples from natural language phonology. §3 outlines the basic concepts and definitions to be used throughout the paper. §4 defines the TSL languages and discusses their properties. §5 details the 2TSLIA and proves it learns the TSL_2 class in polynomial time and data. §6 discusses future work, and §7 concludes.

2 Linguistic motivation

The primary motivation for studying the TSL languages and their learnability comes from their relevance to phonotactic (word well-formedness) patterns in natural language phonology. Many phonotactic patterns belong to either the Strictly Local (SL) languages (McNaughton and Papert, 1971) or the Strictly Piecewise (SP) languages (Rogers et al., 2010).[1] An example of phonotactic knowledge which is SL is Chomsky and Halle's (Chomsky and Halle, 1965) observation that English speakers will classify *blick* as a possible word of English while rejecting *bnick* as impossible. This is SL because it can be described as **bn* being unacceptable as a word-initial sequence. SP languages can describe long-distance dependencies based on precedence relationships between sounds (such as consonant harmony in Sarcee, in which *s* may not follow a *ʃ* anywhere in a word, but may precede one (Cook, 1984)) (Heinz, 2010a). Also, the SL and SP languages are efficiently learnable (García et al., 1990; Heinz, 2010a; Heinz, 2010b; Heinz and Rogers, 2013).

However, there are some long-distance patterns which cannot be described purely with precedence relationships. One examples is a pattern from Latin, in which in certain cases an *l* cannot follow another *l* unless an *r* intervenes, no matter the distance between them (Jensen, 1974; Odden, 1994; Heinz, 2010a). This can be seen in the *-alis* adjectival suffix (Example 1), which appears as *-aris* if the word it attaches to already contains an *l* ((d) through (f) in Example 1), except in cases where there is an intervening *r*, in which it appears again as *-alis* ((g) through (i) in Example 1). In the examples, the sounds in question are underlined for emphasis (data from (Heinz, 2010a)), and for (d) through (f), illicit forms are given, marked with a star (*).

Example 1.

a.	*navalis*	*'naval'*
b.	*episcopalis*	*'infinitalis'*
c.	*infinitalis*	*'negative'*
d.	*solaris*	*'solar' (*solalis)*
e.	*lunaris*	*'lunar' (*lunalis)*
f.	*militaris*	*'military' (*militalis)*
g.	*floralis*	*'floral'*
h.	*sepulkralis*	*'funeareal'*
i.	*litoralis*	*'of the shore'*

This non-local alternating pattern is not SP because SP languages cannot express blocking effects (Heinz, 2010a). However, it can be described with a TSL grammar in which the tier is $\{r, l\}$ and the permissible tier 2-factors do not include $*ll$ and $*rr$. This yields exactly the set of strings in which an l is always immediately (disregarding all sounds besides $\{r, l\}$) followed by an r, and vice versa.

Formal learning algorithms for SL and SP languages can provide a model for human learning of SL and SP sound patterns (Heinz, 2010a). TSL languages are also similarly learnable, given the stipulation that both the tier and k are fixed. For natural language, the value for k never seems to go above 2 (Heinz et al., 2011). However, tiers vary in human language—TSL patterns occur both with different kinds of vowels (Nevins, 2010) and consonants (Suzuki, 1998; Bennett, 2013). For example, in Turkish the tier is the entire vowel inventory (Clements and Sezer, 1982), while in Finnish it is vowels except /i,e/ (Ringen, 1975). In Samala consonant harmony, the tier is sibilants (Rose and Walker, 2004), whereas in Koorete, the tier is sibilants and {b,r,g,d} (McMullin and Hansson, to appear). Thus, it is of interest to understand how *both* the tier and permissible tier 2-factors for TSL_2 grammars might be learned efficiently.

3 Preliminaries

Basic knowledge of set theory is assumed. Let $S_1 - S_2$ denote the set theoretic difference of sets S_1 and

[1] The relationship between these formal language classes and human cognition (linguistic and otherwise) is discussed in more detail in (Rogers and Pullum, 2011; Rogers and Hauser, 2010; Rogers et al., 2013; Lai, 2013; Lai, 2015).

S_2. Let $\mathcal{P}(S)$ denote the powerset of S and $\mathcal{P}_{fin}(S)$ be the set of all finite subsets of S.

Let Σ denote a set of symbols, referred to as the *alphabet*, and let a *string* over Σ be a finite sequence of symbols from that alphabet. The length of a string w will be denoted $|w|$. Let λ denote the empty string; $|\lambda| = 0$. Let Σ^* (Kleene star) represent all strings over this alphabet, and Σ^k represent all strings of length k. Concatenation of two strings w and v (or symbols σ_1 and σ_2) will be written wv ($\sigma_1\sigma_2$). Special beginning ($\rtimes$) and end ($\ltimes$) symbols ($\rtimes, \ltimes \notin \Sigma$) will often be used to mark the beginning and end of words; the alphabet Σ augmented with these, $\Sigma \cup \{\rtimes, \ltimes\}$, will be denoted $\Sigma_{\rtimes\ltimes}$.

A string $u \in \Sigma^*$ is said to be a *factor* or *substring* of another string w if there are two other strings $v, v' \in \Sigma^*$ such that $w = vuv'$. We call u a k-factor of w if it is a factor of w and $|u| = k$. Let $\texttt{fac}_k : \Sigma^* \rightarrow \mathcal{P}(\Sigma^{\leq k})$ be a function mapping strings to their k-factors, where $\texttt{fac}_k(w)$ equals $\{u|u \text{ is a } k\text{-factor of } w\}$ if $|w| > k$ and equals $\{w\}$ otherwise. For example, $\texttt{fac}_2(aab) = \{aa, ab\}$ and $\texttt{fac}_8(aab) = \{aab\}$. We extend the k-factor function to languages; $\texttt{fac}_k(L) = \bigcup_{w \in L} \texttt{fac}_k(w)$.

3.1 Grammars, languages, and learning

A *language* (or *stringset*) L is a subset of Σ^*. If L is finite, let $|L|$ denote the cardinality of L, and let $||L||$ denote the *size* of L, which is defined to be $\sum_{w \in L} |w|$. Let $L_1 \cdot L_2$ denote the concatenation of the languages L_1 and L_2, i.e., the pairwise concatenation of each word in L_1 to each word in L_2. For notational simplicity, the concatenation of a singleton language $\{w\}$ to another language L_2 (or L_1 to $\{w\}$) will be written wL_2 (L_1w).

A *grammar* is a finite representation of a possibly infinite language. A class $\mathbb{L}$ of languages is represented by a class $\mathbb{R}$ of representations if every $r \in \mathbb{R}$ is of finite size and there is a naming function $\mathcal{L} : \mathbb{R} \rightarrow \mathbb{L}$ which is both total and surjective.

The learning paradigm used in this paper is identification in the limit learning paradigm (Gold, 1967), with polynomial bounds on time and data (de la Higuera, 1997). This paradigm has two complementary aspects. First, it requires that the information which distinguishes a learning target from other potential targets be present in the input for algorithms to successfully learn. Second, it requires successful algorithms to return a hypothesis in time polynomial of the size of the sample, and that the size of the sample itself must be polynomial in the size of grammar.

The definition of the learning paradigm (Definition 3) depends on some preliminary notions.

Definition 1. *Let $\mathbb{T}$ be a class of functions represented by some class $\mathbb{R}$ of representations.*

1. *An input* sample *I for a language $L \in \mathbb{L}$ is a finite set of data consistent with L, that is to say $I \subseteq L$.*
2. *A* $(\mathbb{L}, \mathbb{R})$-learning algorithm $\mathfrak{A}$ *is a program that takes as input a sample for a language $L \in \mathbb{L}$ and outputs a representation from $\mathbb{R}$.*

The notion of characteristic sample is integral.

Definition 2 (Characteristic sample)**.** *For a $(\mathbb{L}, \mathbb{R})$-learning algorithm $\mathfrak{A}$, a sample CS is a* characteristic sample *of a language $L \in \mathbb{L}$ if for all samples I for L such that $CS \subseteq I$, $\mathfrak{A}$ returns a representation r such that $\mathcal{L}(r) = L$.*

Now the learning paradigm can be defined.

Definition 3 (Identification in polynomial time and data)**.** *A class $\mathbb{L}$ of languages is identifiable in polynomial time and data if there exists a $(\mathbb{L}, \mathbb{R})$-learning algorithm $\mathfrak{A}$ and two polynomials $p()$ and $q()$ such that:*

1. *For any sample I of size m for $L \in \mathbb{L}$, $\mathfrak{A}$ returns a hypothesis $r \in \mathbb{R}$ in $\mathcal{O}(p(m))$ time.*
2. *For each representation $r \in \mathbb{R}$ of size n, there exists a characteristic sample of r for $\mathfrak{A}$ of size at most $\mathcal{O}(q(n))$.*

4 Tier-based Strictly Local Languages

This section introduces the Tier-based Strictly Local (TSL) languages (Heinz et al., 2011). The TSL languages generalize the SL languages (McNaughton and Papert, 1971; García et al., 1990; Caron, 2000), and as such these will briefly be discussed first.

4.1 The Strictly Local Languages

The SL languages can be defined as follows (Heinz et al., 2011):

Definition 4 (SL languages)**.** *A language L is* Strictly k-Local *(SL_k) iff there exists a finite set $S \subseteq \texttt{fac}_k(\rtimes\Sigma^*\ltimes)$ such that $L = \{w \in \Sigma^* | \texttt{fac}_k(\rtimes w \ltimes) \subseteq S\}$*

Such a set S is sometimes referred to as the *permissible k-factors* or *permissible substrings* of L. For example, let $L = \{\lambda, ab, abab, ababab, ...\}$. This L can be described with a set of permissible 2-factors $S = \{\rtimes\ltimes, \rtimes a, ab, ba, b\ltimes\}$ because every 2-factor of every word in L is in S; thus, L is *Strictly 2-Local* (abbreviated SL$_2$).

As a set S of permissible k-factors is finite it can also be viewed as a SL *grammar* where $L(S) = \{w \in \Sigma^* | \texttt{fac}_k(\rtimes w \ltimes) \subseteq S\}$. An element s of an SL grammar S for a language L is *useful* (resp. *useless*) iff $s \in \texttt{fac}_k(L)$ ($s \notin \texttt{fac}_k(L)$). A *canonical* SL grammar contains no useless elements.

In the example above, $aa \notin S$ and $aa \notin \texttt{fac}_2(w)$ for any $w \in L$. Such a string is referred to as a *forbidden k-factor* or a *restriction* on L. The set of forbidden k-factors R is $\texttt{fac}_k(\rtimes\Sigma\ltimes) - S$. Thinking about the grammar in terms of S or in terms of R is equivalent, but in some cases it is simpler to refer to one rather than the other, so we shall use both.

Any SL$_k$ class of languages is learnable with polynomial bounds on time and data if k is known in advance (García et al., 1990; Heinz, 2010b).

The class of SL$_k$ languages (for each k) belongs to a collection of language classes called *string extension language classes* (Heinz, 2010b). The discussion above presents SL$_k$ languages from this perspective. These language classes have many desirable learning properties, due to their underlying lattice structure (Heinz et al., 2012).

4.2 The TSL languages

The TSL languages can be thought of as a further parameterization on the k-factor function where a certain subset of the alphabet takes part in the grammar and all other symbols in the alphabet are ignored. This special subset is referred to as a *tier* $T \subseteq \Sigma$. Symbols not on the tier are removed from consideration of the grammar by an *erasing function* $E_T : \Sigma^* \rightarrow T^*$ defined as $E_T(\sigma_0\sigma_1...\sigma_n) = u_0u_1...u_n$ where $u_i = \sigma_i$ if $\sigma_i \in T$; else $u_i = \lambda$.

For example, if $\Sigma = \{a, b, c\}$ and $T = \{a, c\}$ then $E_T(bbabbcbba) = aca$. We can then define a tier version $\texttt{fac}_{T\text{-}k}$ of $\texttt{fac}_k$ as $\texttt{fac}_{T\text{-}k}(w) = \texttt{fac}_k(\rtimes E_T(w)\ltimes)$.

Here, $\rtimes$ and $\ltimes$ are built into the function as they are always treated as part of the tier. Continuing the example from above, $\texttt{fac}_{T\text{-}2}(bbabbcbba) = \{\rtimes a, ac, ca, a\ltimes\}$. $\texttt{fac}_{T\text{-}k}$ can be extended to languages as with $\texttt{fac}_k$ above.

The TSL languages can now be defined parallel to the SL languages (the following definition is equivalent to the one in (Heinz et al., 2011)):

Definition 5 (TSL languages)**.** *A language L is* Tier-based Strictly k-Local *iff there exists a subset $T \subseteq \Sigma$ of the alphabet and a finite set $S \subseteq \texttt{fac}_k(\rtimes \mathrm{T}^* \ltimes)$ such that $L = \{w \in \Sigma^* | \texttt{fac}_{T\text{-}k}(\rtimes w \ltimes) \subseteq S\}$*

Parallel to SL grammars above, $\langle T, S\rangle$ can be thought of as a TSL grammar of L. Likewise, the *forbidden tier substrings* (or *tier restrictions)* R is simply the set $\texttt{fac}_{T\text{-}k}(\Sigma^*) - S$. Finally, $\langle T, S\rangle$ is canonical if S contains no useless elements (i.e., $s \in S \Leftrightarrow s \in \texttt{fac}_{T\text{-}k}(L(\langle T, S\rangle)))$ and R is nonempty (this second restriction is explained below).

For example, let $\Sigma = \{a, b, c\}$ and $T = \{a, c\}$ as above and let $S = \{\rtimes\ltimes, \rtimes a, \rtimes c, ac, a\ltimes, ca, cc, c\ltimes\}$. Plugging these into Definition 5, we obtain a language L which only contains those strings without the forbidden 2-factor aa on tier T. These are words which may contain bs interspersed with as and cs provided that no a precedes another unless a c intervenes. For example, $bbabbcbba \in L$ but $bbabbbbabb \notin L$, because $E_T(bbabbbbabb) = aa$ and $aa \in \texttt{fac}_2(\rtimes aa\ltimes)$ but $aa \notin S$.

Like the class of SL$_k$ languages, the class of TSL languages (for fixed T and k) are a string extension language class. The relationship of the TSL languages to other sub-Regular language classes (McNaughton and Papert, 1971; Rogers et al., 2013) is studied in (Heinz et al., 2011).

Given a fixed k and T, S is easily learnable in the same way as the SL languages (Heinz et al., 2011). However, as discussed above, in the case of natural language phonology, it is not clear that information about T is available *a priori*. Learning both T and S simultaneously is thus an interesting problem.

This problem admits a technical, but unsatisfying, solution. The number of subsets T such that $T \subseteq \Sigma$ is finite, so there are a number of TSL$_k$ languages given a fixed k is finite. It is already known that any finite class of languages which can be enumerated can be identified in the limit by an algorithm which checks through the enumeration (Gold, 1967). However, given the cognitive relevance of TSL languages, it is of interest to pursue a smarter, computationally efficient method for learning them.

What are the consequences of varying T? When $T = \Sigma$, the result is an SL$_k$ language, because the erasing function operates vacuously (Heinz et al., 2011). Conversely, when $T = \emptyset$, by Definition 5, S is either $\emptyset$ or $\{\rtimes\ltimes\}$. The former obtains the empty language while the latter obtains Σ^*. By Definition 5, Σ^* can be described with any $T \subseteq \Sigma$ as long as $S = \texttt{fac}_2(\rtimes T^* \ltimes)$. In such a grammar, none of the members of T serve any purpose; thus we stipulate that for a canonical grammar R is nonempty.

Importantly, a member of T may not belong to a string in R and still serve a purpose. For example, let $\Sigma = \{a, b, c\}$, $T = \{a, c\}$, and $S = \texttt{fac}_2(\rtimes T^* \ltimes) - \{aa\}$. Because c appears in no forbidden tier substrings in R, it is freely distributed in $L = L(\langle T, S\rangle)$. However, it makes a difference in the language, because $aca \in L$ but $aba \notin L$. If c (and the relevant tier substrings in S) were missing from the tier, neither aba nor aca would be in L. This can be thought of a 'blocking' function of c, because it allows sequences $a...a$ even though $aa \in R$.

We may now return to the Latin example from §2 in a little more detail. The Latin pattern, in which rs and ls must alternate, regardless of other intervening sounds. This can be captured by a TSL grammar in which $T = \{l, r\}$ and $S = \{\rtimes l, \rtimes r, lr, rl, rr, r\ltimes, l\ltimes\}$. This captures the generalization that, ignoring all sounds besides l and r, l and l are never allowed to be adjacent. The remainder of the paper discusses how such a grammar, including T, may be induced from positive data.

5 Algorithm

This section introduces the Tier-based Strictly 2-Local Inference Algorithm (2TSLIA), which induces both a tier and a set of permissible tier 2-factors from positive data. First, in §5.1, the concept of a *path*, which is crucial to the learner, is defined. §5.3 introduces and describes the algorithm, and §5.4 defines a distinguishing example and proves that it is a characteristic sample for which the algorithm is guaranteed to converge. Time and data complexity for the algorithm are discussed there.

5.1 Paths

First, we define the concept of 2-*paths*, which serve a function similar to the 2-subsequences of SP$_2$ grammars. How this concept might be generalized to k is discussed in §6. Like subsequences, paths denote precedence relations between symbols in a string, but they are further augmented with sets of intervening symbols. Formally, a 2-path is a 3-tuple $\langle x, Z, y\rangle$ where x and y are a symbol in $\Sigma_{\rtimes\ltimes}$ and Z is a subset of Σ. The 2-paths of a string $w = \sigma_0\sigma_1 \dots \sigma_n$, denoted $paths_2(w)$, are

$$paths_2(w) = \big\{\langle \sigma_i, Z, \sigma_j\rangle \big| \quad \begin{array}{r} i < j \text{ and} \\ Z = \{\sigma_z | i < z < j\}\big\} \end{array}$$

Essentially, the 2-paths are 2-subsequences of a string interpolated with sets of intervening symbols. For example, $\langle a, \{b, c\}, d\rangle$ is in $paths_2(abbcd)$ because a precedes d and $\{b, c\}$ is the set of symbols that come between them. Intuitively, it gives the set of symbols one must 'travel over' in order to get from one symbol to another in a string. As shall be seen shortly, this information is essential for how the algorithm works. Let $paths_2()$ be extended to languages such that $paths_2(L) = \bigcup_{w\in L} paths_2(w)$. As we are here only concerned with 2-paths, we henceforth simply refer to them as 'paths'.

Remark 1. *The paths of a string w can be calculated in at most time quadratic in the size of $w = \sigma_1 \cdots \sigma_n$.*

To see why, consider a $n\times n$ table and consider i, j such that $1 \leq i < j, \leq n$. The idea is each nonempty cell in the table contains the set of intervening symbols between σ_i and σ_j. Let $p(x, y)$ denote the entry in the x-th row and the y-th column. The following hold: $p(i, i+1) = \emptyset$; $p(i, i+2) = \{\sigma_{i+1}\}$; and $p(i, j) = \bigcup_{i\leq x\leq j-2} p(x, x+2)$ for any $j \geq i+3$. Since each of these operations is linear time or less, the size of the table, which equals n^2, provides an upper bound on the time complexity of $paths_2(w)$. (This bound is not tight since half the table is empty.)

5.2 Terminology

Before introducing the algorithm we define some terms which are useful for understanding its operation. For a TSL$_2$ grammar $G = \langle T, S\rangle$, $T \subseteq \Sigma$ will be referred to as the *tier*, S the *allowed tier substrings*, and $R = \texttt{fac}_{T\text{-}2}(\Sigma^*) - S$ as the *forbid-*

den tier substrings. Let $H = \Sigma - T$ be the *non-tier elements* of G. For any elements $\sigma_i, \sigma_j \in T$, their *tier adjacency* with respect to a set L of strings refers to whether or not they may appear adjacent on the tier; formally, σ_i, σ_j are tier adjacent in L iff $\sigma_i\sigma_j \in \mathtt{fac}_{T\text{-}2}(w)$ for some $w \in L$.

An *exclusive blocker* is a $\sigma \in T$ which does not appear in R. That is, $\forall w \in R, \sigma \notin \mathtt{fac}_1(w)$. An exclusive blocker is not restricted in its distribution but may intervene between other elements on the tier. The *free elements* of a grammar G will refer to the union of the set of the nontier elements of G and exclusive blockers given G.

Given an order $\sigma_0, \sigma_1, ..., \sigma_n$ on Σ, let $\Sigma_i = \{\sigma_h | h < i\}$ refer to the elements of Σ less than σ_i and $J_i = \Sigma - \Sigma_i$ be the elements σ_i and greater. Note that $\Sigma_0 = \emptyset$ and $J_0 = \Sigma$. Let $H_i = H \cap \Sigma_i$ be the *non-tier elements less than* σ_i and $T_i = (T \cap \Sigma_i) \cup J_i$ be the *expanded tier given* σ_i, or the tier plus elements from J_i. Note that H_i and T_i are complements of each other with respect to Σ. In the context of the algorithm, T_i will represent the algorithm's current hypothesis for the tier. When referring not to the order on Σ but to the index of positions in a string or path, $\tau_1, \tau_2, ..., \in \Sigma$ shall be used.

For a path $\langle \tau_1, X, \tau_2 \rangle$, we refer to τ_1 and τ_2 as the *symbols* on the path, and X as the *intervening set*. For a set of paths P, the term *tier paths* will refer to the paths whose symbols are on $T_{\rtimes\ltimes}$. Formally, the tier paths are $P_T = \{\langle \tau_1, X, \tau_2 \rangle \in P | \tau_1, \tau_2 \in T_{\rtimes\ltimes}, X \subseteq \Sigma\}$. Note that P_T is restricted only by the symbols on the path; the intervening set X may be *any* subset of Σ.

Example 2. *Let* $\Sigma = \{a, b, c, d\}$ *with the order* $a < b < c < d$ *and let* $T = \{b, d\}$*. Referring to* b *as* σ_2 *in the order,* $\Sigma_2 = H_2 = \{a\}$*, and* $T_2 = \{b, c, d\}$*. If* $w = bacd$*, and* $P = paths_2(w)$*, then*

$$P = \Big\{ \langle \rtimes, \{\}, b \rangle, \langle \rtimes, \{b\}, a \rangle, \langle \rtimes, \{a, b\}, c \rangle, \langle \rtimes, \{a, b, c\}, d \rangle, \langle \rtimes, \{a, b, c, d\}, \ltimes \rangle, \langle b, \{\}, a \rangle, \langle b, \{a\}, c \rangle, \langle b, \{a, c\}, d \rangle, \langle b, \{a, c, d\}, \ltimes \rangle, \langle a, \{\}, c \rangle, \langle a, \{c\}, d \rangle, \langle a, \{c, d\}, \ltimes \rangle, \langle c, \{d\}, \ltimes \rangle, \langle c, \{\}, d \rangle, \langle d, \{\}, \ltimes \rangle \Big\},$$

$$P_{T_2} = \Big\{ \langle \rtimes, \{\}, b \rangle, \langle \rtimes, \{a, b\}, c \rangle, \langle \rtimes, \{a, b, c\}, d \rangle, \langle \rtimes, \{a, b, c, d\} \ltimes \rangle, \langle b, \{a\}, c \rangle, \langle b, \{a, c\}, d \rangle, \langle b, \{a, c, d\}, \ltimes \rangle, \langle c, \{d\}, \ltimes \rangle, \langle c, \{\}, d \rangle, \langle d, \{\}, \ltimes \rangle \Big\},$$

and $P_T = \Big\{ \langle \rtimes, \{\}, b \rangle, \langle \rtimes, \{a, b, c\}, d \rangle, \langle \rtimes, \{a, b, c, d\} \ltimes \rangle, \langle b, \{a, c\}, d \rangle, \langle b, \{a, c, d\}, \ltimes \rangle, \langle d, \{\}, \ltimes \rangle \Big\}$

5.3 Algorithm

The 2TSLIA$_2$ (Algorithm 1 on the following page) takes an ordered alphabet Σ and a set of input strings I and returns a TSL grammar $\langle T, S \rangle$. It has two main functions: `get_tier`, which calculates T, and `main`, which calls `get_tier` and uses the resulting tier to determine S.

First, `get_tier` takes as arguments an index i, an expanded tier T_i, and a set of paths P. The expanded tier is the algorithm's hypothesis for the tier at stage i. It is first called with $T_i = \Sigma$ (the most conservative hypothesis for the tier) and $i = 0$. The goal of `get_tier` is to whittle down T_i, which is the set of elements known to be in T plus the set of elements whose membership in T has not yet been determined, down to only elements known to be in T.

The `get_tier` function recursively iterates through T_i, starting with σ_0, to determine which members of T_i should be in the final hypothesis T. Two other pieces of data are important for this: P_{T_i}, or the tier paths of T_i whose symbols are in $T_i \cup \{\rtimes, \ltimes\}$, and H_i, or the set of non-tier elements less than σ_i. The set H_i is the algorithm's largest safe hypothesis for non-tier elements, and it will reason about restrictions on σ_i using paths from P_{T_i}.

Elements of Σ are checked for membership on the tier in order; thus, when checking σ_i, `get_tier` has already checked every other σ_h for $h < i$. For each σ_i, membership in T is decided on two conditions labeled (a) and (b) in the **if-then** statement of `get_tier`. Condition (a) tests whether σ_i is a free element, and condition (b) further tests whether σ_i is an exclusive blocker. If σ_i is found to be a free element and *not* an exclusive blocker, then it is a non-tier element (see §5.2), and should be removed from T. In detail:

Condition (a). To test whether σ_i is a free element, this condition checks to see if there are any restrictions on σ_i given P_{T_i} and H_i. It searches through P_{T_i} for $\langle \rtimes, X, \sigma_i \rangle$, $\langle \sigma_i, X', \ltimes \rangle$, and, for all $\sigma' \in T_i$, $\langle \sigma_i, Y, \sigma' \rangle$ and $\langle \sigma', Y', \sigma_i \rangle$, where the intervening sets X, X', Y, Y' are all subsets of H_i. If all of these appear in P_{T_i}, it means that σ_i is a free

```
Data: An alphabet $\Sigma = \{\sigma_0, \sigma_1, ..., \sigma_n\}$; finite
    set $I$ of input strings in $\Sigma^*$
Result: A $TSL_2$ grammar $\langle T, S \rangle$
function get_tier($T_i, P, i$):
    Initialize $P_{T_i} = \{\langle \tau_1, X, \tau_2 \rangle \in P | \tau_1, \tau_2 \in T_i \cup \{\rtimes, \ltimes\}\}$;
    Initialize $H_i = \Sigma - T_i$;
    for $i \leq n$ do
        Let $\sigma = \sigma_i$;
        if a.) $\exists X, X' \subseteq H_i$ such that
          $\langle \rtimes, X, \sigma \rangle, \langle \sigma, X', \ltimes \rangle \in P_{T_i}$ and
          $\forall \sigma' \in T_i, \exists Y, Y' \subseteq H_i$ such that
          $\langle \sigma, Y, \sigma' \rangle, \langle \sigma', Y', \sigma \rangle \in P_{T_i}$ and
            b.) for each $\langle \tau_1, Z, \tau_2 \rangle \in P_{T_i}$ with
          $\tau_1, \tau_2 \in ((T_i \cup \{\rtimes, \ltimes\}) - \{\sigma\}), \sigma \in Z, Z - \{\sigma\} \subseteq H_i$, $\exists Z' \subseteq H_i$ such that
          $\langle \tau_1, Z', \tau_2 \rangle \in P_{T_i}$ then
            Return get_tier($T_i - \{\sigma\}, P_{T_i}, i + 1$);
        else
            i = i+1
        end
    end
    Return $\langle T_i, P_{T_i} \rangle$;
function main($I, \Sigma$):
    Initialize $P = paths_2(I)$;
    Initialize $S = \emptyset$;
    Set $T, P_T =$ get_tier($\Sigma, P, 0$);
    for $p = \langle \tau_1, X, \tau_2 \rangle \in P_T$ do
        if $X \subseteq \Sigma - T$ then
            Add $\tau_1 \tau_2$ to $S$;
        end
    end
    Return $\langle T, S \rangle$;
```

Algorithm 1: The TSL$_2$ Learning Algorithm (2TSLIA)

element with respect to T_i.

Example 3. *Let $\Sigma = \{a, b, c, d\}$ from Example 2, with that order, and let $I = \{aaa, bab, cac, dad\}$. In the initial condition of the algorithm, $i = 0$, so $\sigma_i = a$, $T_i = \Sigma$, $H_i = \emptyset$, and $P_{T_i} = paths_2(I)$.*

Because $H_i = \emptyset$, a satisifes condition (a) if $\langle \rtimes, \emptyset, a \rangle \in P_{T_i}$, $\langle a, \emptyset, \ltimes \rangle \in P_{T_i}$, and for each $\sigma \in T_i = \{a, b, c, d\}$, $\langle \sigma, \emptyset, a \rangle \in P_{T_i}$ and $\langle a, \emptyset, \sigma \rangle \in P_{T_i}$. This is true given this particular I, because $\langle \rtimes, \emptyset, a \rangle, \langle a, \emptyset, \ltimes \rangle, \langle a, \emptyset, a \rangle, \in paths_2(aaa)$, $\langle b, \emptyset, a \rangle, \langle a, \emptyset, b \rangle \in paths_2(bab)$, $\langle c, \emptyset, a \rangle, \langle a, \emptyset, c \rangle \in paths_2(cac)$, and $\langle d, \emptyset, a \rangle, \langle a, \emptyset, d \rangle \in paths_2(dad)$.

Condition (b). This condition checks whether σ_i is an exclusive blocker, and thus should not be removed from T_i. It does this by looking for a pair τ_1, τ_2 of members of T_i distinct from σ_i whose tier-adjacency is *dependent* on σ_i. It searches through paths of the form $\langle \tau_1, Z, \tau_2 \rangle$, where Z includes σ_i and some subset of H_i. For each such $\langle \tau_1, Z, \tau_2 \rangle$, it searches for a $\langle \tau_1, Z', \tau_2 \rangle$ where Z' is *only* a subset of H_i. If such a $\langle \tau_1, Z', \tau_2 \rangle$ exists, then τ_1 and τ_2 are tier-adjacent in the data regardless of σ_i. However, if no such path exists, then it may be that τ_1 and τ_2 can only appear with σ_i in between them, and thus get_tier infers that σ_i is an exclusive blocker. If any such pair is found, then condition (b) fails, and σ_i must remain in T_i.

Example 4. *Continuing with Example 3, a would not satisfy condition (b) given I. Given that $i = 0$, in order for a to satisfy condition (b), for all $\tau_1, \tau_2 \in \{\rtimes, b, c, d, \ltimes\}$, if $\langle \tau_1, \{a\}, \tau_2 \rangle$ is in P_{T_i}, then $\langle \tau_1, \emptyset, \tau_2 \rangle$ must also be in P_{T_i}. For this I, this is false for $\tau_1 = \tau_2 = b$, because $\langle b, \{a\}, b \rangle \in paths_2(bab)$, but $\langle b, \emptyset, b \rangle \notin paths_2(I)$. Thus* get_tier *would infer that a is an exclusive blocker.*

However, the reader can confirm that a would satisfy condition (b) for an input $I' = I \cup \{\lambda, bb, cc, dd\}$.

If both conditions (a) and (b) are met, then the algorithm determines that σ_i should not appear on the final hypothesis T for the tier, and so get_tier is recursively called with $T_i - \{\sigma_i\}$ and $i + 1$ as arguments. Because the hypothesis for the tier has changed, P_{T_i} is also passed, so on the next call when get_tier calculates $P_{T_{i+1}}$, it only has to search through P_{T_i} instead of the full set of paths P.

If neither condition is met, no change is made to T_i, i is increased, and the next σ_i is checked by continuing through the **for** loop. This process is repeated for each member of the alphabet, after which T_i is returned as the final answer for the tier. Its tier paths P_{T_i} are also returned.

The function main calls get_tier and takes its result as T and P_T. Using this, it finds each $\tau_1 \tau_2 \in S$. It does this by searching through P_T and finding paths $\langle \tau_1, X, \tau_2 \rangle$ whose intervening set X is a subset of the non-tier elements $\Sigma - T$. Such $\tau_1\tau_2$ pairs are thus tier-adjacent in I. The resulting grammar $\langle T, S \rangle$ is then returned.

5.4 Identification in polynomial time and data

Here we establish the main result of the paper that 2TSLIA identifies the TSL$_2$ class in polynomial time and data. As is typical, the proof relies on a establishing a characteristic sample for the 2TSLIA.

Lemma 1 establishes 2TSLIA runs efficiently.

Lemma 1. *Given an input sample I of size n, 2TSLIA outputs a grammar in time polynomial in the size of n.*

Proof. The paths are calculated for each word once at the beginning of `main`. This takes time quadratic in the size of the sample (Remark 1), so $\mathcal{O}(n^2)$. Call this set of paths P (note P is also bounded in size by n^2).

Additionally, the loop in `get_tier` is called exactly $|\Sigma|$ times. Checking condition (a) in `get_tier` requires a single pass through the paths. On other hand, condition (b) requires one search through P for every path element $p \in P$ in the worst case, which is $\mathcal{O}(n^4)$. Thus the time complexity for `get_tier` is $\mathcal{O}(|\Sigma|(n^2 + n^4)) = \mathcal{O}(n^4)$ since $|\Sigma|$ is a constant.

Lastly, finding the permissible substrings on the tier also requires a single pass through P, which also takes $\mathcal{O}(n^2)$. Altogether then an upper bound on the time complexity of 2TSLIA is given by $\mathcal{O}(n^2 + n^4 + n^2) = \mathcal{O}(n^4)$, which is polynomial. □

Next we define a *distinguishing sample* for a target language for the 2TSLIA. We first show it is polynomial in the size of the target grammar. We then show that it is a characteristic sample.

Definition 6 (Distinguishing sample)**.** *Given an alphabet Σ, with some order $\sigma_1, \sigma_2, ..., \sigma_n$ on Σ, the target language $L \in$ TSL$_2$, and the canonical grammar $G = \langle T, S \rangle$ for L, a distinguishing sample D for G is the set meeting the following conditions. Recall that $H = \Sigma - T$ are the non-tier elements and H_i refers to the non-tier elements less than σ_i.*

1. ***The non-tier element condition.*** *For all non-tier elements $\sigma_i \in H$,*
   - *i.* $\exists w_1, w_2 \in D$ *s.t.* $\langle \rtimes, X, \sigma \rangle \in paths_2(w_1)$ *and* $\langle \sigma, X', \ltimes \rangle \in paths_2(w_2)$, $X, X' \subseteq H_i$
     $\forall \sigma' \in T_i$, $\exists v_1, v_2 \in D$ *s.t.* $\langle \sigma, Y, \sigma' \rangle \in paths_2(v_1)$ *and* $\langle \sigma', Y', \sigma \rangle \in paths_2(v_2)$, $Y, Y' \subseteq H_i$.
   - *ii.* $\forall \tau_1\tau_2 \in (\mathtt{fac}_{T_i\text{-}2}((\Sigma - \{\sigma_i\})^*) - R)$, $w \in D$ *s.t.* $\langle \tau_1, X, \tau_2 \rangle \in paths_2(w)$, $X \subseteq H_i$.
2. ***The exclusive blocker condition.*** *For each $\sigma_i \in T$ which is an exclusive blocker, $w \in D$ s.t. $\langle \tau_1, X, \tau_2 \rangle \in paths_2(w)$, where $\tau_1\tau_2 \in R$, $\tau_1 \neq \sigma_i$, $\tau_2 \neq \sigma_i$, $\sigma_i \in X$, and $X - \sigma_i \subseteq H_i$*
3. ***The allowed tier substring condition.*** *$\forall \tau_1\tau_2 \in S$, some w s.t. $\langle \tau_1, X, \tau_2 \rangle \in paths_2(w)$ where $X \subseteq H$*

Essentially, item (1) ensures that all symbols σ_i not on the target T will meet conditions (a) and (b) in the for loop and be removed from the algorithm's hypothesis for T. Item (2) ensures that, in the situation an exclusive blocker $\sigma_i \in T$ meets condition (a) for removal from the tier hypothesis, it will not meet condition (b). Item (3) ensures that the sample contains every $\tau_1\tau_2$ in S. These points will be discussed more detail in the proof that the distinguishing example is a characteristic sample.

Lemma 2. *Given a grammar $G = \langle T, S \rangle$ whose size is $|T| + ||S||$, the size of a distinguishing sample D for G is polynomial in the size of G.*

Proof. Recall that $T \subseteq \Sigma$ and $S \subseteq \Sigma^2_{\rtimes\ltimes}$, that $H = \Sigma - T$ and $R = \Sigma^2 - S$. The non-tier element condition requires that for each $\sigma \in H$ and $\sigma' \in T$, the sample minimally contains the words σ, $\sigma'\sigma$, and $\sigma'\sigma$, whose total length is $|H| + 2|H||T|$. The exclusive blocker condition requires for each exclusive blocker $\sigma \in T$ and each $\tau_1\tau_2 \in R$ that minimally $\tau_1\sigma\tau_2$ is contained in the sample. Letting B denote the set of exclusive blockers, we have the total length of words in the characteristic sample is $||B|| \times ||R||$. Finally, the allowed tier substring condition is requires for each $\tau_1\tau_2 \in S$ that minimally $\tau_1\tau_2$ is contained in the sample. Hence, the length of these words equals $|S|$.

Altogether this means there is a characteristic sample D such that $||D|| = |H| + 2|H||T| + ||B|| \times ||R|| + |S|$. Since $H, T, B \subseteq \Sigma$ and $R, S \subseteq$

Σ^2, $||D|| \leq |\Sigma| + 2|\Sigma||\Sigma| + |\Sigma||\Sigma|^2 + |\Sigma|^2 = |\Sigma| + 3|\Sigma|^2 + |\Sigma|^3$. Thus, $||D||$ is bounded by $\mathcal{O}(|\Sigma|^3)$ which, since $|\Sigma|$ is a constant, is effectively $\mathcal{O}(1)$. □

Next we prove Lemma 3, which shows that the tier conjectured by the 2TSLIA at step i is correct for all symbols up to σ_i. Thus, once all symbols are treated by the 2TSLIA, its conjecture for the tier is correct. Let T'_i correspond to the algorithm's current tier hypothesis when σ_i is being checked in the **for** loop of the `get_tier` function, let $H'_i = \Sigma - T'_i$ be the algorithm's hypothesis for the set of non-tier elements less than σ_i, and let $P_{T'_i}$ be the set of paths under consideration (i.e., the set of paths from the initialization of P_{T_i} before the **for** loop). As above, $\tau_0\tau_1...\tau_m$ index positions in a string or path.

Lemma 3. *Let $\Sigma = \{\sigma_0, \ldots, \sigma_n\}$ and consider any $G = \langle T, S\rangle$. Given any finite input sample I which contains a distinguishing sample D for G, it is the case that for all i $(0 \leq i \leq n)$, $T'_i = T_i$.*

Proof. The proof is by recursion on i. The base case is when $i = 0$. By definition, $T_0 = \Sigma$. The algorithm starts with $T'_0 = \Sigma$, so $T'_0 = T_0$.

Next we assume the recursive hypothesis (RH): for some $i \in \mathbb{N}$ that $T'_i = T_i$. We prove that $T'_{i+1} = T_{i+1}$. Specifically, we show that if RH is true for i, then (Case 1) $\sigma_i \in H_{i+1}$ implies $\sigma_i \in H'_{i+1}$ and (Case 2) $\sigma_i \notin H_{i+1}$ implies $\sigma_i \notin H'_{i+1}$.

Case 1. This is the case in which σ_i is a non-tier element. The non-tier element condition in Definition 6 for D ensures that the data in I will meet both conditions (a) and (b) in the **for** loop in `get_tier()` for removing σ_i from the tier hypothesis.

Condition (a) requires that $\langle \rtimes, X, \sigma_i\rangle \in P_{T'_i}$ and $\langle \sigma_i, X', \ltimes\rangle \in P_{T'_i}$ for some $X, X' \subseteq H'_i$, and $\forall \sigma' \in T_i, \langle \sigma_i, Y, \sigma'\rangle \in P_{T'_i}$ and $\langle \sigma', Y', \sigma_i\rangle \in P_{T'_i}$ for some $Y, Y' \subseteq H'_i$. Part (i) of the non-tier element condition in Defintion (6) ensures that for σ_i, there are words $w_1, w_2 \in I$ such that $\langle \rtimes, X, \sigma_i\rangle \in paths_2(w_1)$, $\langle \sigma_i, X', \ltimes\rangle \in paths_2(w_2)$, and, for all $\sigma' \in T_i$, $v_1, v_2 \in I$ s.t. $\langle \sigma_i, Y, \sigma'\rangle \in paths_2(v_1)$ and $\langle \sigma', Y', \sigma_i\rangle \in paths_2(v_2)$, where the intervening sets X, X', Y, Y' are all subsets of H_i. Because by RH $H'_i = H_i$, condition (a) for removing σ_i from the tier hypothesis is satisfied.

For σ_i to satisfy condition (b), for any path $\langle \tau_1, X, \tau_2\rangle \in P_{T'_i}$ such that $\sigma_i \in X$ and $X - \{\sigma_i\} \subseteq H'_i$, there must be another path $\langle \tau_1, X', \tau_2\rangle \in P_{T'_i}$ where $X' \subseteq H'_i$. If $\tau_1\tau_2 \in R$, then such a $\langle \tau_1, X, \tau_2\rangle$ is guaranteed not to exist in $P_{T'_i}$, because τ_1 and τ_2 will, by the definition of R, not appear in the data with only non-tier elements between them. For $\tau_1\tau_2 \notin R$, part (ii) of the nontier element condition in Definition 6 ensures that some $\langle \tau_1, Y, \tau_2\rangle$ wher e $Y \subseteq H_i$ exists in $P_{T'_i}$, as it requires that for each such τ_1, τ_2 there is some $w \in I$ such that $\langle \tau_1, X, \tau_2\rangle \in paths_2(w)$ where $X \subseteq H_i$. By hypothesis $H'_i = H_i$, and so there is always guaranteed to be some $\langle \tau_1, X', \tau_2\rangle \in P_{T'_i}$ where $X' \subseteq H'_i$. Thus, condition (b) will always be satisfied for σ_i.

Thus, assuming the RH, $\sigma_i \in H_{i+1}$ is guaranteed to satisfy conditions (a) and (b) and be removed from the algorithm's hypothesis for the tier, so $\sigma_i \in H'_{i+1}$.

Case 2. This is the case in which $\sigma_i \in T$. There are two mutually exclusive possibilities. The first possibility is that σ_i is not a free element. Here, σ_i is guaranteed to not be taken off of the tier hypothesis, because condition (a) for removing a symbol from the tier requires that there exists some path $\langle \sigma_j, X, \sigma_i\rangle$ and $\langle \sigma_i, X', \sigma_j\rangle$ where $X, X' \subseteq H'_i$. From the definition of a TSL grammar, there will exist no $\langle \sigma_j, Y, \sigma_i\rangle$ and $\langle \sigma_i, Y', \sigma_j\rangle$, where $Y, Y' \subseteq H$, in the paths of I. Because $H_i \subseteq H$ and by hypothesis $H'_i = H_i$, $H'_i \subseteq H$, so the algorithm will correctly register that $\sigma_j\sigma_i \in R$ or $\sigma_i\sigma_j \in R$ and σ_i will remain on the tier hypothesis. Thus $\sigma_i \notin H'_{i+1}$.

The other possibility is that σ_i is an exclusive blocker. If so, σ_i may satisfy condition (a) for tier removal (as discussed above for the case in which $\sigma_i \in H$). However, the exclusive blocker condition in Definition 6 for D guarantees that σ_i will not meet condition (b). As discussed above, condition (b) will fail if there is a $\langle \tau_1, X, \tau_2\rangle \in P_{T'_i}$ such that X includes σ_i and zero or more elements of H'_i and no other path $\langle \tau_1, X', \tau_2\rangle \in P_{T'_i}$ where X' only includes elements of H'_i. The exclusive blocker condition requires that there will be some $\tau_1\tau_2 \in R$ such that there is a word $w \in D$ s.t. $\langle \tau_1, Y, \tau_2\rangle \in paths_2(w)$ where $\sigma_i \in Y$ and $Y - \{\sigma_i\} = H_i$. From the definition of a TSL grammar, no w such that $\langle \tau_1, Y', \tau_2\rangle \in paths_2(w)$ where $Y_i \subseteq H_i$ will appear in I, because $H_i \subseteq H$. Because by hypothesis $H'_i = H_i$, the algorithm will correctly find

$\langle \tau_1, Y, \tau_2 \rangle$ and also find that there is no $\langle \tau_1, Y', \tau_2 \rangle$. Thus when σ_i is an exclusive blocker, it will not be removed from the tier hypothesis, and $\sigma_i \notin H'_i$.

We now know that both Cases (1) and (2) are true; thus, assuming the RH, $H_{i+1} = H'_{i+1}$. Thus, by induction, $(\forall i)[H_i = H'_i]$. Because H_i and H'_i are the complements of T_i and T'_i, respectively, $(\forall i)[T_i = T'_i]$. □

Lemma 4 (Distinguishing sample = characteristic sample)**.** *A* distinguishing sample *for a TSL*$_2$ *language as defined in Definition 6 is a* characteristic sample *for that language for the 2TSLIA.*

Proof. As above, let $G' = (T', S')$ be the output of the algorithm. From Lemma 3, we know that for any language $L(G)$, $G = \langle T, S \rangle$, and a characteristic sample D for G, given any sample I of L such that $D \subseteq I$, $(\forall i)[T_i = T'_i]$. That $T' = T$ immediately follows from this fact.

That $S' = S$ follows directly from $T' = T$ and the allowed tier substring condition of Definition 6 for D. The allowed tier substring condition states that for all $\tau_1\tau_2 \in S$, the distinguishing sample will contain some w s.t. $\langle \tau_1, X, \tau_2 \rangle \in paths_2(w)$ where $X \subseteq H$. Because $T = T'$, the **for** loop of `main` will correctly find all such $\tau_1\tau_2$. Thus, $S = S'$, and $G = G'$. □

Theorem 1 (Identification in the Limit in polynomial time and data)**.** *The 2TSLIA identifies the TSL*$_k$ *languages in polynomial time and data.*

Proof. Immediate from Lemmas 1, 2, and 4. □

We note further that in the worst case the time complexity is polynomial (degree four Lemma 1) and the data complexity is constant (Lemma 2).

6 Discussion

This algorithm opens up multiple paths for future work. The most immediate theoretical question is whether the algorithm here can be (efficiently) generalized to any TSL class as long as the learner knows k *a priori*. We believe that it can. The notion of 2-paths can be straightforwardly extended to k such that a k-path is a $2k - 1$ tuple of the form $\langle \sigma_0, X_0, \sigma_1, X_1, ..., X_{k-1}, \sigma_k \rangle$, where each set X_i represents the symbols between σ_i and σ_{i+1}. The algorithm presented here can then be modified to check a set of such k-paths. We believe such an algorithm could be shown to be provably correct using a proof of similar structure to the one here, although time and data complexity will likely increase.

However, in terms of applying these algorithms to natural language phonology, it is likely that a k value of 2 is sufficient. As pointed out by Heinz et al. (2011), TSL$_2$ can describe both long-distance dissimilation and assimilation patterns. However, other issues arise with natural language data. One is that natural language corpora often include some exceptions to phonotactic generalizations. Algorithms which take as input such noisy data and output grammars that are guaranteed to be representations of languages ‘close’ to the target language have been obtained and studied in the PAC learning paradigm (Angluin and Laird, 1988). It would be interesting to apply such techniques and other similar ones to adapt 2TSLIA into an algorithm that remains effective despite noisy input data.

One final issue is how to generalize over multiple tiers. Jardine (to appear), in running versions of the 2TSLIA on natural language corpora, shows that it fails because local dependencies (which again can be modeled where $T = \Sigma$) prevent crucial information in the CS from appearing in the data. Furthermore, natural languages can have separate long-distance phonotactics which hold over distinct tiers. For example, KiYaka has both a vowel harmony pattern (Hyman, 1998) and a liquid-nasal harmony pattern over the tier {l,m,n,ŋ} (Hyman, 1995). Thus, words in KiYaka exhibit a pattern corresponding to the *intersection* of two TSL$_2$ grammars, one with a vowel tier and one with a nasal-liquid tier. The problem of learning a finite intersection of TSL$_2$ languages is thus another relevant learning problem.

7 Conclusion

This paper has presented an algorithm which can learn a grammar for any TSL$_2$ language in time polynomial time in the size of the input sample, whose size is bounded by a constant. As the TSL$_2$ languages can model long-distance phonotactics in natural language, this represents a step towards understanding how humans internalize such patterns.

References

Dana Angluin and Philip Laird. 1988. Learning from noisy examples. *Machine Learning*, 2:343–370.

William Bennett. 2013. *Dissimilation, Consonant Harmony, and Surface Correspondence*. Ph.D. thesis, Rutgers, the State University of New Jersey.

Pascal Caron. 2000. Families of locally testable languages. *Theoretical Computer Science*, 242:361–376.

Noam Chomsky and Morris Halle. 1965. Some controversial questions in phonological theory. *Journal of Linguistics*, 1(2):pp. 97–138.

George N. Clements and Engin Sezer. 1982. Vowel and consonant disharmony in Turkish. In Harry van der Hulst and Norval Smith, editors, *The Structure of Phonological Representations (Part II)*. Foris, Dordrecht.

Eung-Do Cook. 1984. *A Sarcee Grammar*. Vancouver: University of British Columbia Press.

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

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

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

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

John Goldsmith and Jason Riggle. 2012. Information theoretic approaches to phonological structure: the case of finnish vowel harmony. *Natural Language & Linguistic Theory*, 30(3):859–896.

John Goldsmith and Aris Xanthos. 2009. Learning phonological categories. *Language*, 85(1):4–38.

Jeffrey Heinz and James Rogers. 2013. Learning subregular classes of languages with factored deterministic automata. In Andras Kornai and Marco Kuhlmann, editors, *Proceedings of the 13th Meeting on the Mathematics of Language (MoL 13)*, pages 64–71, Sofia, Bulgaria, August. Association for Computational Linguistics.

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

Jeffrey Heinz, Anna Kasprzik, and Timo Kötzing. 2012. Learning with lattice-structured hypothesis spaces. *Theoretical Computer Science*, 457:111–127, October.

Jeffrey Heinz. 2010a. Learning long-distance phonotactics. *Linguistic Inquiry*, 41:623–661.

Jeffrey Heinz. 2010b. String extension learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 897–906. Association for Computational Linguistics, July.

Larry Hyman. 1995. Nasal consonant harmony at a distance: The case of Yaka. *Studies in African Linguistics*, 24:5–30.

Larry Hyman. 1998. Positional prominence and the 'prosodic trough' in Yaka. *Phonology*, 15:14–75.

Adam Jardine. to appear. Learning tiers for long-distance phonotactics. In *Proceedings of the 6th Conference on Generative Approaches to Language Acquisition North America (GALANA 2015)*.

John Jensen. 1974. Variables in phonology. *Language*, 50:675–686.

Regine Lai. 2013. *Domain Specificity in Learning Phonology*. Ph.D. thesis, University of Delaware.

Regine Lai. 2015. Learnable versus unlearnable harmony patterns. *Linguistic Inquiry*, 46(3). In press.

Kevin McMullin and Gunnar Ólafur Hansson. to appear. Long-distance phonotactics as Tier-Based Strictly 2-Local languages. In *Proceedings of the Annual Meeting on Phonology 2015*.

Robert McNaughton and Seymour Papert. 1971. *Counter-Free Automata*. MIT Press.

Andrew Nevins. 2010. *Locality in Vowel Harmony*. Number 55 in Linguistic Inquiry Monographs. MIT Press.

David Odden. 1994. Adjacency parameters in phonology. *Language*, 70(2):289–330.

Catherine Ringen. 1975. *Vowel Harmony: Theoretical Implications*. Ph.D. thesis, Indiana University.

James Rogers and Marc D. Hauser. 2010. The use of formal language theory in studies of artificial language learning: a proposal for distinguishing the differences between human and nonhuman animal lerners. In Harry van der Hulst, editor, *Recursion and Human Language (Studies in Generative Grammar [SGG]) 104*. de Gruyter.

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

James Rogers, Jeffrey Heinz, Gil Bailey, Matt Edlefsen, Molly Visscher, David Wellcome, and Sean Wibel. 2010. On languages piecewise testable in the strict sense. In Christian Ebert, Gerhard Jäger, and Jens Michaelis, editors, *The Mathematics of Language*, volume 6149 of *Lecture Notes in Artifical Intelligence*, pages 255–265. Springer.

James Rogers, Jeffrey Heinz, Margaret Fero, Jeremy Hurst, Dakotah Lambert, and Sean Wibel. 2013. Cognitive and sub-regular complexity. In *Formal Grammar*, volume 8036 of *Lecture Notes in Computer Science*, pages 90–108. Springer.

Sharon Rose and Rachel Walker. 2004. A typology of consonant agreement as correspondence. *Language*, 80:475–531.

Keiichiro Suzuki. 1998. *A typological investigation of dissimilation*. Ph.D. thesis, University of Arizona, Tucson.