JMLR: Workshop and Conference Proceedings 34:33–48, 2014 Proceedings of the 12th ICGI

A Canonical Semi-Deterministic Transducer

Achilles Beros ACHILLES.BEROS@UNIV-NANTES.FR
Colin de la Higuera CDLH@UNIV-NANTES.FR
Laboratoire LINA UMR CNRS 6241
UFR de Sciences et Techniques
2 rue de la Houssinière, BP 92208
44322 Nantes Cedex 03, France

Editors: Alexander Clark, Makoto Kanazawa and Ryo Yoshinaka

Abstract

We prove the existence of a canonical form for semi-deterministic transducers with sets of pairwise incomparable output strings. Based on this, we develop an algorithm which learns semi-deterministic transducers given access to translation queries. We also prove that there is no learning algorithm for semi-deterministic transducers that uses only domain knowledge.

Keywords: Transducers, Semi-deterministic

1. Introduction

Transducers, introduced by Reutenauer and Schützenberger (1991, 1995), are a type of abstract machine which defines a relation between two formal languages. As such, they are interpreted as modeling translation in any context where formal languages are applicable. We provide no background on formal languages in this paper; an overview of the subject can be found in Berstel (1979) and Sakarovich (2009). Alternatively, transducers can be viewed as a generalization of finite state machines. This view was introduced by Mohri, who uses transducers in the context of natural language processing Mohri (1997, 2000) and Mohri et al. (2000).

A fundamental task when studying the theory of transducers is to look for classes of transducers that can be learned given access to some form of data. If a class of transducers, $\mathscr{C}$, is found to be learnable, then a predictive model can be produced in any application where a translation from the class $\mathscr{C}$ is in use. The significance of transducers, specifically expanding the range of the learnable classes, is clear from the scope of applications of transducers. Among many others, some well known applications are in the fields of morphology and phonology (Roark and Sproat, 2007), machine translation (Amengual et al., 2001; Casacuberta and Vidal, 2004; Clark, 2001), web wrappers (Carme et al., 2005), speech (Mohri, 1997) and pattern recognition (Bernard et al., 2006). In each of these cases, different classes of transducers are examined with characteristics suitable to the application. Distinguishing characteristics of different classes include determinism properties, the use of probabilites or weights, as well as details of the types of transitions that are permitted.

1.1. Transducer learning

An important step in the theory of transducers was the development of the algorithm Ostia. Introduced in Oncina et al. (1993), Ostia was designed for language comprehension tasks (Castellanos et al., 1993). A number of elaborations on the original algorithm have since arisen, many of them aimed at trying to circumvent the restriction to total functions that limited Ostia. Typically, these attempts involved adding some new source of information. For example, Ostia-N uses negative (input) examples and Ostia-D supposes the algorithm has some knowledge of the domain of the function (Oncina and Varó, 1996). Similar ideas were explored later by Kermorvant and de la Higuera (2002), Coste et al. (2004) and Kermorvant et al. (2004). An application of Ostia for active learning is presented in Vilar (1996). Using dictionaries and word alignments has been tested by Vilar (2000). A demonstrated practical success of Ostia came in 2006. The Tenjinno competition (Starkie et al., 2006) was won by Clark (2006) using an Ostia inspired algorithm.

1.2. Towards nondeterminism with transducers

Non-deterministic transducers pose numerous complex questions – even parsing becomes a difficult problem (Casacuberta and de la Higuera, 1999, 2000). Interest in non-deterministic models remains, however, as the limitations of subsequential transducers make them unacceptable for most applications. The first lifting of these constraints was proposed by Allauzen and Mohri (2002). They propose a model in which the final states may have multiple outputs. In his PhD thesis, Akram introduced a notion of semi-determinism (Akram, 2013) that strikes a balance between complete non-determinism and the very restrictive subsequential class. He provided an example witnessing that semi-deterministic transducers are a proper generalization of deterministic transducers, but did not pursue the topic further, focusing instead on probabilistic subsequential transducers. We examine an equivalent formulation of Akram's semi-determinism based on methods of mathematical logic. In particular, by viewing the definition from a higher level of the ranked universe, we convert what would be a general relation into a well-defined function. Kunen (1980) provides an overview of a number of important topics in set theory including the ranked and definable universes. Some more recent developments in set theory is Jech (2013).

A significant obstacle in learning non-deterministic transducers is the fact that an absence of information cannot be interpreted. One approach to overcoming this problem is to use probabilities. We eschew the probabilistic approach in favor of a collection of methods that have their antecedents in Beros's earlier work distinguishing learning models (Beros, 2013) and determining the arithmetic complexity of learning models (Beros, 2012).

1.3. The learning model

There are two principal learning models in grammatical inference: identification in the limit (Gold, 1967) and PAC-learning (Valiant, 1984). Each of these models admits variants depending on what additional sources of information are provided. In order to learn semi-deterministic transducers, we use queries (Angluin, 1987) as an additional resource. These queries are very limited; the oracle will be interrogated about a possible translation pair and the oracle will return either a TRUE or FALSE. We also prove that learning is not possible

without queries. We write $[x, X]_f$ to indicate a query if $\langle x, X\rangle$ is in the bi-language f. The precise definition of learning we use is adapted from the one used in de la Higuera (1997):

Definition 1 *An algorithm, A,* polynomial identifies in the limit with translation queries *a class of transducers, $\mathscr{C}$, if for any $G \in \mathscr{C}$ there is a set, CS_G, such that on any $\mathcal{D} \supseteq CS_G$ generated by G, A outputs a G' equivalent to G. The algorithm must converge within a polynomial amount of time in $|\mathcal{D}|$ and $|G|$; $|CS_G|$ must be polynomial in $|G|$.*

Note that in the above definition the number of calls to the oracle is also bounded by the overall complexity of the algorithm and is therefore polynomial in the size of the sample.

2. Notation

We make use of following common notation in the course of this paper. Throughout, the symbols x, y and z denote strings and a and b will denote elements of a given alphabet. We shall use the standard notation λ for the empty string.

- A tree is a directed acyclic graph. T' is a subtree of T if both T and T' are trees and T' is contained in T. A strict subtree is a subtree that is not equal to the containing tree.
- Given T, a tree, $T_x = \{z : xz \in T\}$. For a set of strings, S, $T[S]$ is the prefix closure of S.
- We write $x \prec y$ if there is a string z such that $y = xz$. We write $x \preceq y$ if $x \prec y$ or $x = y$. This order is called the prefix order.
- $\mathscr{P}(X) = \{Y : Y \subseteq X\}$ and $\mathscr{P}^*(X) = \{Y : Y \subseteq X \wedge |Y| < \infty\}$.
- Following the notation of set theory, the string $x = a_0 \dots a_n$ is a function with domain $n + 1$. Thus, $x{\restriction}k = a_0 \dots a_{k-1}$ for $k \leq n + 1$. $|x|$ is the length of x and x^- is the truncation $x{\restriction}(|x|-1)$. Note that the last element of x is $x(|x|-1)$ and the last element of x^- is $x(|x| - 2)$.
- We write $x \parallel y$ if $x = y$, $x \prec y$ or $x \succ y$ and say x and y are comparable. Otherwise, we write $x \perp y$ and say that x and y are incomparable.
- By $<_{lex}$ and $<_{llex}$ we denote the lexicographic and length-lexicographic orders, respectively.
- For an alphabet Σ, Σ^* is the set of all finite strings over Σ and $\Sigma^*_x = \{y \in \Sigma^* : y \succeq x\}$. A tree over Σ is a tree whose members are members of Σ^*, where the ordering of the tree is consistent with the prefix order on Σ^* and the tree is prefix closed.
- We reserve a distinguished character, #, which we exclude from all alphabets under consideration and we will use # to indicate the end of a word. We will write $x\#$ when we append the # character to x.

3. Bi-Languages and Transducers

Bi-languages are the fundamental objects of study. They capture the semantic correspondence between two languages. In principle, this correspondence does not specify any ordering of the two languages, but translation is always done from one language *to* another language. As such, we refer to the input and the output languages of a bi-language. For notational simplicity, in everything that follows Σ is the alphabet for input languages and Ω is the alphabet for output languages. Using this notation, the input language is a subset of Σ^* and the output language is a subset of Ω^*. We now present the standard definition of a bi-language.

Definition 2 *Consider two languages, $L \subseteq \Sigma^*$ and $K \subseteq \Omega^*$. A* bi-language from L to K *is a subset of $L \times K$.*

For our purposes, we wish to indicate the direction of translation and to aggregate all translations of a single string. To this end, in the remainder of this paper, we will use the following equivalent definition of a bi-language.

Definition 3 *Consider two languages, $L \subseteq \Sigma^*$ and $K \subseteq \Omega^*$. A* bi-language from L to K *is a function $f : L \to \mathscr{P}(K)$. L is said to be the* input language *and K the* output language *of f. When defined without reference to a specific output language, a bi-language is simply a function $f : L \to \mathscr{P}(\Omega^*)$*

We are interested in languages whose generating syntax is some form of transducer.

Definition 4 *A transducer is a finite state machine in which an output string is compiled from the outputs of transitions along the path that an input string follows through the machine. A transducer, G, consists of a finite set of* states*, written* STATES$[G]$*, a set of transitions, E, and a collection of initial states, $q_0, q_1, \ldots, q_k$. A transition, $e \in E$, has four components: a start state, start(e), an end state, end(e), an input, $i(e)$ and an output, $o(e)$. There is a special type of transition called a* #-transition*. If e is a #-transition, then end(e) is an initial state (by convention) and the input is #. Termination of the processing of an input string occurs if and only if the final transition is a #-transition.*

This paper addresses *semi-deterministic bi-languages* which are bi-languages generated by *semi-deterministic transducers*. These were defined in Akram (2013). We use an equivalent formulation.

Definition 5 *A* semi-deterministic transducer (SDT) *is a transducer with a unique initial state such that*

1. *$i(e) \in \Sigma$ for every transition e,*
2. *given a state, q, and $a \in \Sigma$, there is at most one transition, e, with start$(e) = q$ and $i(e) = a$ and*
3. *given a transition, e, $o(e)$ is a finite set of pairwise incomparable strings in Ω^* (i.e., $o(e) \in \mathscr{P}^*(\Omega^*)$).*

When it is necessary to refer to a transition as a single tuple, we will write the components in the following order: $\langle start(e), end(e), i(e), o(e)\rangle$. *A* semi-deterministic bi-language *(SDBL) is a bi-language that can be generated by an SDT.*

Two useful properties of SDTs follow from the definition. First, if $e \in E$ and $\lambda \in o(e)$, then $o(e) = \{\lambda\}$. Second, although there may be multiple translations of a single string, every string follows a unique path through an SDT. We must also note that, while SDBLs can be infinite, the image of any member or finite subset of L is finite. Thus, an SDBL is a function $f : L \to \mathscr{P}^*(\Omega^*)$.

Definition 6 *Let G be an SDT with input language L. A* path through G *is a string $e_0 \dots e_k \in E^*$, where E is the set of transitions, such that $start(e_{i+1}) = end(e_i)$ for $i < k$. $G[p]$ is the collection of all outputs of G that can result from following path p. p_x is the unique path through G defined by $x \in \Sigma^*$. If $x \in L$, then we assume p_x ends with a #-transition. We denote the final state of the path p_x by q_x.*

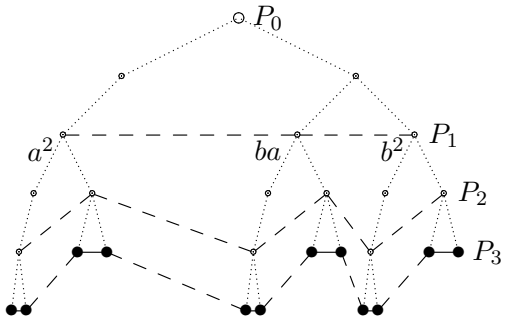
4. Ordering maximal antichains

The following definitions and results pertain to sets of strings and trees over finite alphabets.

Definition 7 *Given a set of strings, S, we call $P \subseteq T[S]$ a* maximal antichain of S *if $(\forall x, y \in P)\big(x \perp y\big)$ and $(\forall x \in S)(\exists y \in P)(y \parallel x)$. P is a* valid antichain of S *if P is a maximal antichain of S and $(\forall x, y \in P)\big(T[S]_x = T[S]_y\big)$. We define, $\mathsf{Vac}(S) = \{P : P$ is a valid antichain of $S\}$.*

Example 1 *Consider the following set of strings over the alphabet $\{a, b\}$: $S = \{a^5, a^4b, a^2ba, a^2b^2, ba^4, ba^3b, baba, bab^2, b^2a^3, b^2a^2b, b^3a, b^4\}$. Graphically, we can represent S as a tree where branching left indicates an a and branching right indicates a b.*

In the picture to the right, we highlight the four valid antichains of S: $P_0 = \{\lambda\}$, $P_1 = \{a^2, ba, b^2\}$, $P_2 = \{a^4, a^2b, ba^3, bab, b^2a^2, b^3\}$ and $P_3 = S$.



It is interesting to note that the valid antichains in the above example have a natural linear ordering. As we shall see in Theorem 11, this is not an artifact of the particular example, but is true of any finite set S.

Definition 8 *For P and Q, sets of strings over some common alphabet, we say that $P <_{ac} Q$ (P is "antichain less than" Q) if either*

1. *$|P| < |Q|$, or*
2. *$|P| = |Q|$ and, for all $x \in P$ and $y \in Q$, if $x \parallel y$, then $x \prec y$.*

We will use valid antichains to parse a set of strings as one would parse a single string into a prefix and suffix. The validity of an antichain ensures that the corresponding suffix set is well-defined. When parsing sets of strings, we will often use the following operations.

Definition 9 *Let S and P be two sets of strings. Define $P * S = \{xy : x \in P \wedge y \in S\}$ and $P^{-1}S = \{y : (\exists x \in P)\big(xy \in S\big)\}$.*

Proposition 10 *Let S be a finite set of strings. If P is a valid antichain of S, then $P * (P^{-1}S) = S$. If P and Q are maximal antichains of the same finite set of strings, then there is a relation $R \subseteq P \times Q$ such that*

1. *$dom(R) = P$*
2. *$ran(R) = Q$ and*
3. *$xRy \leftrightarrow x \parallel y$.*

Furthermore, if $|P| = |Q|$, then R is a well-defined and injective function as well.

Proof Observe that, if P is a valid antichain of S, then $T[P^{-1}S] = T[S]_x$ for all $x \in P$, thus $P * (P^{-1}S) = S$. The proof of the second claim follows from the definitions of valid antichains, $P * S$ and $P^{-1}S$. ■

Theorem 11 *If S is a finite set of strings, then $\Big(\mathsf{Vac}(S), <_{ac}\Big)$ is a finite linear order.*

Proof Consider a finite set of strings, S, and let $T = T[S]$. We begin by fixing $P, Q \in \mathsf{Vac}(S)$. We may assume that $|P| = |Q|$; if $|P| \neq |Q|$ then the claim is trivial. We pick an element $x \in P$ and observe that, by Proposition 10, there is a unique $y \in Q$ such that $x \parallel y$.

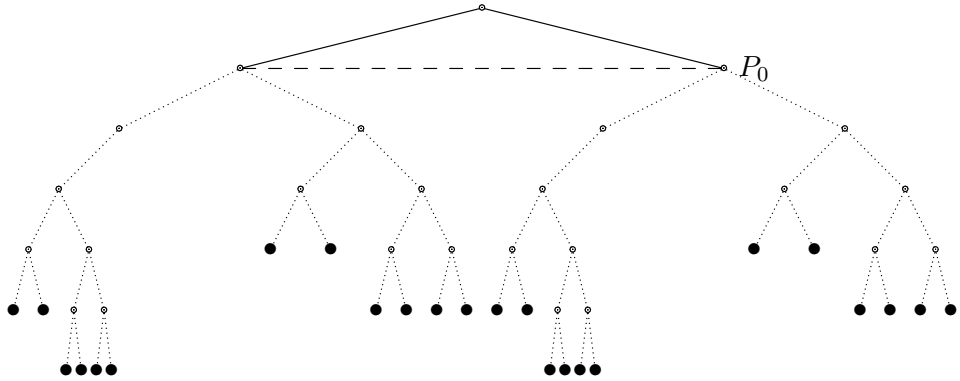
Suppose that $x = y$ and let x' be any other member of P. By Proposition 10, there is a unique $y' \in Q$ such that $x' \parallel y'$. Since P and Q are valid antichains and $x = y$, $T_{x'} = T_x = T_y = T_{y'}$. Given that $x' \parallel y'$, T is finite and $T_{x'} = T_{y'}$ we conclude that $x' = y'$. Now assume $x \prec y$. In the case $y \prec x$ simply exchange the roles of x and y. As above, we pick $x' \in P$ and its unique comparable element $y' \in Q$. Clearly T_y is a strict subtree of T_x and hence, $T_{y'}$ is a strict subtree of $T_{x'}$. We conclude that $x' \prec y'$.

We have shown that any two members of $\mathsf{Vac}(S)$ are comparable. The remaining order properties follow immediately from the definitions. ■

While the proof of Theorem 11 is quite simple, we highlight it as a theorem because it is the critical result for the applications of valid antichains that follow.

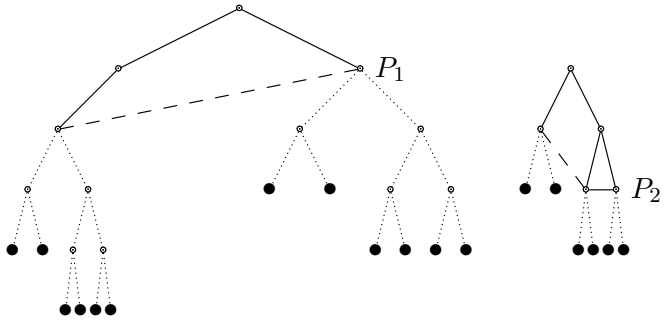
Definition 12 *Given a set of strings, S, a finite sequence of sets of strings, $P_0, \ldots, P_n$, is a* factorization of S *if $S = P_0 * \cdots * P_n$. Such a factorization is said to be* maximal *if, for each $i \in \mathbb{N}$, $\mathsf{Vac}(P_i) = \{\{\lambda\}, P_i\}$; equivalently, if P_{i+1} is the $<_{ac}$-least non-trivial valid antichain of $P_i^{-1} \cdots P_0^{-1} S$.*

Example 2 *We consider the following set of strings:* $S = \{a^5, a^4b, a^3ba^2, a^3bab, a^3b^2a, a^3b^3, aba^2, abab, ab^2a^2, ab^2ab, ab^3a, ab^4, ba^4, ba^3b, ba^2ba^2, ba^2bab, ba^2b^2a, ba^2b^3, b^2a^2, b^2ab, b^3a^2, b^3ab, b^4a^2, b^4ab, b^5a, b^6\}$. *In the figure below, we display the tree,* $T[S]$, *as well as the* $<_{ac}$*-least non-trivial valid antichain,* $P_0 = \{a, b\}$.



The corresponding set of suffixes is $P_0^{-1}S = \{a^4, a^3b, a^2ba^2, a^2bab, a^2b^2a, a^2b^3, ba^2, bab, b^2a^2, b^2ab, b^3a, b^4\}$. *Iterating, we find the next factor is* $P_1 = \{a^2, b\}$ *and its set of suffixes is* $(P_0 * P_1)^{-1}S = \{a^2, ab, ba^2, bab, b^2a, b^3\}$.

We next pick $P_2 = \{a, ab, b^2\}$. *Once we factor out* P_2, *all that remains is* $\{a, b\}$. *The only antichains of* $\{a, b\}$ *are* $\{\lambda\}$ *and* $\{a, b\}$, *both of which are valid antichains. We pick the final factor to be* $P_3 = \{a, b\}$ *and conclude that* $P_0 * P_1 * P_2 * P_3$ *is a maximal factorization of* S.



Corollary 13 *Every finite set of pairwise incomparable strings has a unique maximal factorization. Also, if* $S_0, S_1, S_2, \ldots$ *are finite sets, then* $\bigcap_{i\in\mathbb{N}} \mathsf{Vac}(S_i)$ *is linearly ordered under* $<_{ac}$.

Proof The proof can be found in Beros and de la Higuera (2014). ■

5. Semi-Deterministic Bi-Languages

We cover three topics in this section: onwarding, merging, and the non-learnability of SDBLs. Onwarding is the process of moving decisions earlier in the identification process and merging is the process of conflating identical portions of a learning machine. Taken together, onwarding and mergin prove the existence of a canonical SDT for each SDBL. Onwarding yields a "maximal" function on prefixes of the input language. Merging produces

a finite-order equivalence relation on those prefixes. Together, they will allow us to define a canonical grammar for an arbitrary SDBL. We demonstrate that, given only domain knowledge, SDBLs are not learnable. In section 6, we prove that SDBLs are learnable given access to an additional, very limited, oracle. Detailed proofs of the results in this section can be found in Beros and de la Higuera (2014).

5.1. Onwarding

Definition 14 *Let f be an SDBL. $F : T[L] \to \mathscr{P}^*(\Omega^*)$ is a* semi-deterministic function (SDF) *of f if, for $x \in L$, $f(x) = F(x\restriction 1) * F(x\restriction 2) * \cdots * F(x) * F(x\#)$. We define $\Pi F(x) = F(x\restriction 1) * F(x\restriction 2) * \cdots * F(x)$. If F and F' are SDFs of f, we say that $F <_{sdf} F'$ if $\Pi F(x)$ is a valid antichain of $\Pi F'(x)$ for all x. The SDF* induced *by f is the SDF, F, such that $F(x) = \{\lambda\}$ for all $x \in T[L]$ and $F(x\#) = f(x)$ for all $x \in L$.*

Example 3 *Suppose that $A, B, C \subseteq \Omega^*$ are finite and non-empty. Let $\Sigma = \{a\}$ be the input alphabet. Define an SDBL, f, over $L = \{a^2\}$ by $f(a^2) = A * B * C$. We define two incomparable SDFs of f as follows. The first SDF: $F(\lambda) = \{\lambda\}, F(a) = A * B, F(a^2) = \{\lambda\}$ and $F(a^2\#) = C$. The second SDF: $F'(\lambda) = \{\lambda\}, F'(a) = A, F'(a^2) = B * C$ and $F'(a^2\#) = \{\lambda\}$. Since $\Pi F(a)$ is not a valid antichain of $\Pi F'(a)$, $F \not<_{sdf} F'$. Likewise, since $\Pi F'(a^2)$ is not a valid antichain of $\Pi F(a^2)$, $F' \not<_{sdf} F$.*

Example 3 demonstrates that $<_{sdf}$ is not a linear ordering of the SDFs of a fixed SDBL. Nonetheless, there is a $<_{sdf}$-maximum SDF of f.

Theorem 15 *If f is an SDBL and F and F' are SDFs of f, then is a $<_{sdf}$-maximum SDF of f.*

Proof The proof is inductive. We sketch the construction of a $<_{sdf}$-maximum SDF. Consider $x \in T[L]$ and the factorizations of the images under f of all extensions of x in L. The product of the common factors of all the factorizations are taken to be output on input x of the SDF under construction. By implementing this strategy inductively, we construct the desired SDF. ■

5.2. Merging

The second phase of building a canonical form for SDTs is to define an equivalence relation on the domain of a maximum SDF. This means identifying which paths lead to the same state.

Definition 16 *Let F be an SDF of f over L and $x \in T[L]$. We define $\text{FUTURE}_F[x] = F\restriction\{y \in T[L] : y \succ x\}$. If $x, y \in dom(F)$, we say that $x \equiv y$ if $\text{FUTURE}_F[x] = \text{FUTURE}_F[y]$. Given x, we define (x) to be the $<_{llex}$-least element of $dom(F)$ that is equivalent to x.*

Proposition 17 *$\equiv$ is an equivalence relation on the domain of an SDF. If $x \equiv y$ and $xz, yz \in T[L]$, then $xz \equiv yz$. If F is an SDF of f over L, then there are only a finite number of $\equiv$-equivalence classes on the domain of F. Thus, there is an n such that for all $x, y \in T[L]$, $x \equiv y$ if and only if $\text{FUTURE}_F[x]\restriction x\Sigma^n = \text{FUTURE}_F[y]\restriction y\Sigma^n$.*

Proof The proof follows immediately from the definitions. ■

The maximum SDF and the equivalence relation on its domain depend only on the underlying SDBL. Thus, we have defined a machine-independent canonical form. As a footnote, we demonstrate how to produce an SDT from the canonical form which is unique up to isomorphism. Let f be an SDBL, let F_m be the maximum SDF for f and let $\equiv$ be the equivalence relation on the domain of F_m. We define a finite state machine, G_f, with

1. $\textsc{states}[G_f] = \{q_{(x)} : x \in T[L]\}$,
2. initial state q_λ, and
3. $E_{G_f} = \{\langle q_{(x^-)}, q_{(x)}, x(|x|-1), F_m(x)\rangle : x \in T[L]\} \cup \{\langle q_{(x)}, q_\lambda, \#, F_m(x\#)\rangle : x \in L\}$.

Although L and $T[L]$ may be infinite sets, the set of transitions, E_{G_f}, and the set of states, $\textsc{states}[G_f]$, are finite by Proposition 17.

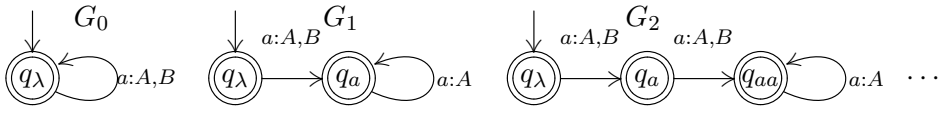
5.3. SDBLs are not learnable

We assume domain knowledge (i.e., access to the characteristic function of the input language). In the proof of the following theorem, we encode a standard example of a "topological" failure of learning in the limit. In particular, we encode the family $\mathcal{H} = \{\mathbb{N}\} \cup \{A : |A| < \infty\}$ into a sequence of SDTs.

Definition 18 *Let f be a bi-language. We define DK_f to be the oracle that, when asked about x, returns a boolean value $DK_f(x)$. If $DK_f(x) = \textsc{true}$, then x is in the input language of f (in other words, the domain of f). Otherwise, x is not in the input language of f. An algorithm which has access to DK_f is said to have domain knowledge about f.*

Theorem 19 *There is a collection of SDBLs, $\mathscr{C}$, such that there is no algorithm which learns in the limit every member of $\mathscr{C}$, even given domain knowledge of each member of $\mathscr{C}$.*

Proof To avoid degenerate cases, we assume the output alphabet has at least two characters, A and B, and the input alphabet has at least one character, a. We exhibit a sequence of SDTs, $\{G_i\}_{i\in\mathbb{N}}$, such that no program can successfully learn every member of the sequence. In the following graphical representation of $\{G_i\}_{i\in\mathbb{N}}$ we omit the #-transitions, indicating terminal nodes with a double border.



Let f_i be the SDBL generated by G_i. The fact that no algorithm can learn every f_i follows from the fact that any finite sequence of translation pairs from f_i (for $i \geq 1$) can be extended to an enumeration either of f_{i+1} or f_0. Thus, given any fixed learning machine M, we can

construct an enumeration of a member of $\mathscr{C}$ that M cannot identify. $\mathscr{C} = \{f_i : i \in \mathbb{N}\}$ is the desired collection of SDBLs.

■

6. Learning with translation queries

We define an oracle which answers questions of the form "is y a valid translation of x?"; equivalently, the oracle answers membership queries about the graph of the bi-language.

Definition 20 *Let f be a bi-language. The translation query $[x, y]_f$ returns* TRUE *if $y \in f(x)$ and* FALSE *otherwise. We call this oracle $[f]$. Where it is clear from context, we will write $[x, y]$ instead of $[x, y]_f$.*

In the remainder of the paper, we exhibit an algorithm that can learn any SDBL, f, in the limit, provided the algorithm has access to the oracles DK_f and $[f]$. We present some of the algorithms that witness the learnability of SDBLs and summarize the result in Theorem 22. A detailed presentation of the algorithms is given in Beros and de la Higuera (2014)

6.1. The characteristic sample

The characteristic sample must contain sufficient data to unambiguously perform two operations: onwarding and merging. Throughout this section f is an SDBL over L and G is the canonical SDT that generates f. We define x' to be the $<_{llex}$-least member of L that extends x. We now proceed to define the characteristic sample for f, denoted CS_f.

Let $x_0, \ldots, x_n$ enumerate the minimal paths to each of the states of G. For each x_i, let P_i be the $<_{ac}$-greatest maximal antichain that is a member of $\mathsf{Vac}(f(x_i y))$ for all y such that $x_i y \in L$. Fix $i \leq n$ and let $P_{i-1} = \{\lambda\}$ if $i = 0$. Define ℓ_i to be the $<_{llex}$-least member of $P_{i-1}^{-1} f(x_i')$. For each ℓ, a prefix of ℓ_i that is not a member of a valid antichain of $P_{i-1}^{-1} f(x_i')$, there is a $y \in P_{i-1}^{-1} f(x_i')$ no prefix of which has the same future in $P_{i-1}^{-1} f(x_i')$ as ℓ. For each prefix of y there is either an extension in $f(x_i)$ which is not in the future of ℓ or there is an extension of ℓ which is not in the future of the prefix of y under consideration. Let $N_0^*(\ell)$ be the set which consists of y together with the witnesses required to show that each prefix of y has a different future from ℓ. Let $N_0(x_i) = \bigcup_{\ell \prec \ell_i} N_0^*(\ell)$ and $N_0(f) = \bigcup_{i \leq n} N_0(x_i)$. Observe that $N_0(f)$ is polynomial in the size of G since $P_i, \ell_i, y < |G|$ for $i \leq n$.

Consider $x \in T[L]$. Let $\mathsf{Vac} = \bigcap_{x \prec y \in L} \mathsf{Vac}(f(y))$. For each $P \in \mathsf{Vac}(f(x)) \setminus \mathsf{Vac}$, observe that there is an example that witnesses the fact that P is not in Vac. Such examples demonstrate either violations of the maximality or the validity of the given antichain. In both cases, the witness is a single element of the graph of f. Since $\mathsf{Vac}(f(x))$ is finite, we can define a finite set, $N_1(f)$, which contains a sufficient number of examples for each minimal path to a state of G.

N_0 and N_1 are required to perform onwarding correctly. In order to perform merges, we must include enough data to identify the equivalence classes of states whose futures are the same. There are two ways in which the futures may differ:

1. there is a string, z, such that $xz \in L$, but $yz \notin L$ or
2. for $X \in G[p_x]$ and $Y \in G[p_y]$, there are z and Z such that $XZ \in G[p_{xz}]$, but $YZ \notin G[p_{yz}]$.

For each member of STATES$[G]$ there is a finite collection of examples which uniquely identify the state. Let $N_2(q_x)$ be a canonically chosen collection of such examples for q_x. Let e be a transition and p' be the $<_{llex}$-least path starting at the initial state, ending with a #-transition and including e. Define $N_2^*(e)$ to be the set of those translations of p' each of which uses a different output of the transition e and is $<_{llex}$-least amongst the translations of p' that use that output. $|N_2^*(e)| = |o(e)|$. We define the final component of CS_f as follows. $N_2(f) = \bigcup_{x \in W} N_2(q_x) \cup \bigcup_{e \in E_G} N_2^*(e)$, where W consists of the minimal paths to each state of G as well as all paths that are immediate extensions of those paths.

Definition 21 *For an SDBL, f, we define the characteristic sample of f, $CS_f = N_0(f) \cup N_1(f) \cup N_2(f)$.*

6.2. Algorithms

Rather than present all the algorithms in detail, we will describe how we use each component of the characteristic sample to make onwarding and merging decisions. This will simultaneously provide the intuition of the algorithms, justify the characteristic sample and sketch the proof of the learnability result. For a detailed presentation, we direct the reader to Beros and de la Higuera (2014).

Consider a dataset, $\mathcal{D}$. We define an initial transducer by creating a state for every member of $T[\text{dom}(\mathcal{D})]$. A tree-like transducer is produced where all transitions output only λ except for the #-transitions at members of $\text{dom}(\mathcal{D})$. All outputs in the dataset are assigned to the #-transitions. Next, by following the $<_{llex}$-least path (and translation) to a state, we can accurately form an array of valid antichains of the outputs of subsequent transitions using the N_0-component of CS_f. We accomplish this using a function, $COMPARE$. Given an input string, x, output strings, Z and W, the function $COMPARE(x, Z, W, \mathcal{D})$ returns TRUE if, for every $\langle x, ZR\rangle, \langle x, WS\rangle \in \mathcal{D}$, the queries $[x, WR]_f$ and $[x, ZS]_f$ return values of TRUE. Otherwise, $COMPARE(x, Z, W, \mathcal{D})$ returns FALSE. From this array, we eliminate those members that are not valid antichains of output trees in the future using the N_1-component of CS_f and translation queries. The $<_{ac}$-greatest antichain that remains is onwarded. As the onwarding method we present is one of the key novel concepts, we include the two algorithms that execute the antichain selection process described above.

In the following algorithm, LLEX-LEAST returns the $<_{llex}$-least member of a set. For sets of strings P and S, we will use $P^{-1}S$, $P * S$ and some set theoretic mainpulations as built-in arithmetic operations.

Algorithm 1: List the valid antichains (VAC)

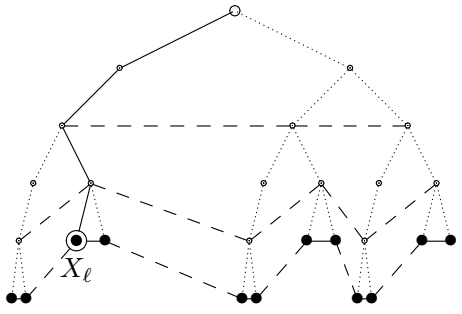
Data: A collection of translation pairs, $\mathcal{D}$; $x \in L$; X_ℓ, the current least translation prefix for x.

Result: An array, A, of all maximal antichains of the translations of x in $\mathcal{D}$ which extend X_ℓ and are not demonstrably invalid.

$X_\ell^{-1}\{Y : Y \succ X_\ell \wedge \langle x, Y\rangle \in \mathcal{D}\} \to T$
$LLEX - LEAST(T) \to Z$
for $\hat{Z} \prec Z$ **do**
  $\hat{Z} \to AC[0]$
  for $R \in T \wedge R \neq Z$ **do**
    for $\hat{R} \prec R$ **do**
      $COMPARE(x, X_\ell\hat{Z}, X_\ell\hat{R}, \mathcal{D}) \to status$
      if $status =$ TRUE **then**
        $\hat{R} \to AC[|AC|]$
        break
      end
    end
    if $status =$ FALSE **then**
      break
    end
  end
  if $status =$ TRUE **then**
    $AC \to A[|A|]$
  end
end
return A

One of the inputs of Algorithm 1 is the "current least translation prefix of x". The current translation prefix will converge to the $<_{llex}$-least output string generated along the unique path corresponding to x. X_ℓ provides a canonical input prefix for testing outputs using translation queries.

Algorithm 1 restricts to the tree of translation pairs whose second component extends the least translation prefix. Every antichain of the tree must contain a prefix of the $<_{llex}$-member of the tree. Because of the linear ordering of the valid antichains (see Theorem 11), there is at most one valid antichain for each prefix of the least member of the tree. COMPARE is used to look for matching nodes to form valid antichains.



As can be seen in the figure, all valid antichains include prefixes of the $<_{llex}$-least member and no two valid antichains contain the same prefix. This provides both a bound on the number of valid antichains and a convenient method to search for the valid antichains.

The next algorithm takes an array of antichains and produces the $<_{ac}$-greatest antichain that appears to be a valid antichain of all trees of outputs on inputs extending x. As data may still be absent, testing the validity for other trees is done using translation queries.

Algorithm 2: Testing an array of antichains against a dataset (TESTVPS)

Data: A string, x, over the input alphabet; an array, A, of antichains for the output tree of input x'; a collection of translation pairs, $\mathcal{D}$.

Result: The $<_{ac}$-greatest member of the array, A, for which there is no evidence in $\mathcal{D}$ that the selected antichain is not valid for all output trees in the future of x.

```
for i = |A| − 1; i ≥ 0; i − − do
    'not valid' → status
    for ⟨xy, Z⟩ ∈ D do
        for R ∈ A[i] do
            if R ≺ Z then
                R^{-1}Z → W
                'valid' → status
                for Q ∈ A[i] do
                    if [xy, QW]_f = FALSE then
                        'not valid' → status
                        break
                    end
                end
                if status = 'not valid' then
                    break
                end
            end
        end
    end
    if status = 'valid' then
        return A[i]
    end
end
```

Observe that there will always be a valid antichain that causes the above algorithm to terminate; if there is no other, then it will terminate on $\{\lambda\}$.

Merging is accomplished using a function, $FUTURE$, that compares the futures of input strings. Define $FUTURE(x, y, G, \mathcal{D}) = \text{TRUE}$ if $(\forall X \in G[p_x], Y \in G[p_y], \langle z, Z\rangle \in \mathcal{D})\big((x \preceq z \wedge X \preceq Z \rightarrow [yx^{-1}z, Y_0X^{-1}Z]_f = \text{TRUE}) \wedge (y \preceq z \wedge Y \preceq Z \rightarrow [xy^{-1}z, X_0X^{-1}Z]_f = \text{TRUE})\big)$, where X_0 and Y_0 are the $<_{llex}$-least members of $G[p_x]$ and $G[p_y]$, respectively. Otherwise, $FUTURE(x, y, G, \mathcal{D}) = \text{FALSE}$. If $FUTURE(x, y, G, \mathcal{D}) = \text{TRUE}$, then the states corresponding to x and y can be merged.

We proceed through the states of the initial transducer in $<_{llex}$-order, first onwarding and then attempting to merge with lesser states. If a state cannot be merged with any lesser state, it is fixed and will not subsequently be changed. The first onwarding and merging

decisions (those about the initial state) are trivially correct. For subsequent decisions, if all previous decisions are correct and we are still using data from CS_f, then we will make correct decisions. Since we will never have to consider any decisions not covered by CS_f, the learning result follows. We state this result in Theorem 22.

Theorem 22 *The class of SDBLs is polynomial identifiable in the limit with translation queries.*

Proof See Beros and de la Higuera (2014) for the precise algorithms, proof that SDBLs are learned by the algorithms and that polynomial bounds are respected. ■

7. Conclusion

We have presented a novel algorithm that learns a powerful class of transducers with the help of reasonable queries. A probabilistic version of these transducers was defined by Akram (2013). We are unaware of any results involving this version. As both probabilities and translation queries can serve the purpose of answering questions about translation pairs not present in the given data, it seems possible that probabilistic transducers could be learned without translation queries, with statistical analysis taking the role of translation queries.

References

H. I. Akram. *Learning Probabilistic Subsequential Transducers*. PhD thesis, Technische Universität München, 2013.

C. Allauzen and M. Mohri. p-subsequentiable transducers. In *Implementation and Application of Automata, 7th International Conference,* Ciaa *2002, Revised Papers*, volume 2608 of Lncs, pages 24–34. Springer-Verlag, 2002.

J. C. Amengual, J. M. Benedí, F. Casacuberta, A. Castaño, A. Castellanos, V. M. Jiménez, D. Llorens, A. Marzal, M. Pastor, F. Prat, E. Vidal, and J. M. Vilar. The EuTrans-I speech translation system. *Machine Translation*, 15(1):75–103, 2001.

D. Angluin. Queries and concept learning. *Machine Learning Journal*, 2:319–342, 1987.

M. Bernard, J.-C. Janodet, and M. Sebban. A discriminative model of stochastic edit distance in the form of a conditional transducer. In Sakakibara et al. (2006), pages 240–252.

A. Beros. Learning in the arithmetic hierarchy. 2012. To appear in the Journal of Symbolic Logic.

A. Beros. Anomalous vacillatory learning. *Journal of Symbolic Logic*, 78(4):1183–1188, 12 2013.

A. Beros and C. de la Higuera. A canonical semi-deterministic transducer. 2014. http://arxiv.org/abs/1405.2476/.

J. Berstel. *Transductions and context-free languages*. Teubner, Leipzig, 1979.

J. Carme, R. Gilleron, A. Lemay, and J. Niehren. Interactive learning of node selecting tree transducer. In Ijcai *Workshop on Grammatical Inference*, 2005.

F. Casacuberta and C. de la Higuera. Optimal linguistic decoding is a difficult computational problem. *Pattern Recognition Letters*, 20(8):813–821, 1999.

F. Casacuberta and C. de la Higuera. Computational complexity of problems on probabilistic grammars and transducers. In de Oliveira (2000), pages 15–24.

F. Casacuberta and E. Vidal. Machine translation with inferred stochastic finite-state transducers. *Computational Linguistics*, 30(2):205–225, 2004.

A. Castellanos, E. Vidal, and J. Oncina. Language understanding and subsequential transducer learning. In *International Colloquium on Grammatical Inference*, pages 11/1–11/10, 1993.

A. Clark. Partially supervised learning of morphology with stochastic transducers. In *Proceedings of the Sixth Natural Language Processing Pacific Rim Symposium*, pages 341–348, 2001.

A. Clark. Large scale inference of deterministic transductions: Tenjinno problem 1. In Sakakibara et al. (2006), pages 227–239.

F. Coste, D. Fredouille, C. Kermorvant, and C. de la Higuera. Introducing domain and typing bias in automata inference. In G. Paliouras and Y. Sakakibara, editors, *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'04*, volume 3264 of Lnai, pages 115–126. Springer-Verlag, 2004.

C. de la Higuera. Characteristic sets for polynomial grammatical inference. *Machine Learning Journal*, 27:125–138, 1997.

A. L. de Oliveira, editor. *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'00*, volume 1891 of Lnai, 2000. Springer-Verlag.

E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

T. Jech. *Set Theory: The Third Millennium Edition, Revised and Expanded*. Springer Monographs in Mathematics. Springer, 2013. ISBN 9783642078996. URL http://books.google.fr/books?id=70N-cgAACAAJ.

C. Kermorvant and C. de la Higuera. Learning languages with help. In P. Adriaans, H. Fernau, and M. van Zaannen, editors, *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'02*, volume 2484 of Lnai, pages 161–173. Springer-Verlag, 2002.

C. Kermorvant, C. de la Higuera, and P. Dupont. Improving probabilistic automata learning with additional knowledge. In A. Fred, T. Caelli, R. Duin, A. Campilho, and D. de Ridder, editors, *Structural, Syntactic and Statistical Pattern Recognition, Proceedings of* Sspr *and* Spr *2004*, volume 3138 of Lncs, pages 260–268. Springer-Verlag, 2004.

K. Kunen. *Set theory: An introduction to independence proofs*, volume 102 of *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Co., Amsterdam-New York, 1980. ISBN 0-444-85401-0.

L. Miclet and C. de la Higuera, editors. *Proceedings of* Icgi *'96*, number 1147 in Lnai, 1996. Springer-Verlag.

M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(3):269–311, 1997.

M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234:177–201, 2000.

M. Mohri, F. C. N. Pereira, and M. Riley. The design principles of a weighted finite-state transducer library. *Theoretical Computer Science*, 231(1):17–32, 2000.

J. Oncina and M. A. Varó. Using domain information during the learning of a subsequential transducer. In Miclet and de la Higuera (1996), pages 301–312.

J. Oncina, P. García, and E. Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.

C. Reutenauer and M.-P. Schützenberger. Minimization of rational word functions. *SIAM Journal of Computing*, 20(4):669–685, 1991.

C. Reutenauer and M.-P. Schützenberger. Variétés et fonctions rationnelles. *Theoretical Computer Science*, 145(1&2):229–240, 1995.

B. Roark and R. Sproat. *Computational Approaches to Syntax and Morphology*. Oxford University Press, 2007.

Y. Sakakibara, S. Kobayashi, K. Sato, T. Nishino, and E. Tomita, editors. *Grammatical Inference: Algorithms and Applications, Proceedings of* Icgi *'06*, volume 4201 of Lnai, 2006. Springer-Verlag.

J. Sakarovich. *Elements of Automata Theory*. Cambridge University Press, 2009.

B. Starkie, M. van Zaanen, and D. Estival. The Tenjinno machine translation competition. In Sakakibara et al. (2006), pages 214–226.

L. G. Valiant. A theory of the learnable. *Communications of the Association for Computing Machinery*, 27(11):1134–1142, 1984.

J. M. Vilar. Query learning of subsequential transducers. In Miclet and de la Higuera (1996), pages 72–83.

J. M. Vilar. Improve the learning of subsequential transducers by using alignments and dictionaries. In de Oliveira (2000), pages 298–312.