LEARNING REGULAR LANGUAGES OVER LARGE ORDERED ALPHABETS

IRINI-ELEFTHERIA MENS AND ODED MALER

VERIMAG, CNRS and University of Grenoble, France
e-mail address: irini-eleftheria.mens@imag.fr

VERIMAG, CNRS and University of Grenoble, France
e-mail address: oded.maler@imag.fr

Abstract. This work is concerned with regular languages defined over large alphabets, either infinite or just too large to be expressed enumeratively. We define a generic model where transitions are labeled by elements of a finite partition of the alphabet. We then extend Angluin's L^* algorithm for learning regular languages from examples for such automata. We have implemented this algorithm and we demonstrate its behavior where the alphabet is a subset of the natural or real numbers. We sketch the extension of the algorithm to a class of languages over partially ordered alphabets.

Introduction

The main contribution of this paper is a generic algorithm for learning regular languages defined over a large alphabet Σ. Such an alphabet can be infinite, like $\mathbb{N}$ or $\mathbb{R}$ or just so large, like $\mathbb{B}^n$ for very large n or large subsets of $\mathbb{N}$, so that it is impossible or impractical to treat it in an enumerative way, that is, to write down the entries of the transition function $\delta(q, a)$ for every $a \in \Sigma$. The obvious solution is to use a *symbolic* representation where transitions are labeled by predicates which are applicable to the alphabet in question. Learning algorithms infer an automaton from a finite set of words (the *sample*) for which membership is known. Over small alphabets, the sample should include the set S of all the shortest words that lead to each state (access sequences) and, in addition, the set $S \cdot \Sigma$ of all their Σ-continuations. Over large alphabets this is not a practical option and as an alternative we develop a symbolic learning algorithm over *symbolic words* which are only partially backed up by the sample. In a sense, our algorithm is a combination of automaton learning and learning of non-temporal predicates. Before getting technical, let us discuss briefly some motivation.

Finite automata are among the corner stones of Computer Science. From a practical point of view they are used routinely in various domains ranging from syntactic analysis,

1998 ACM Subject Classification: [Theory of computation] Formal languages and automata theory. [Learning] Learning regular languages.

Key words and phrases: symbolic automata, active learning.

This paper is an extended version of [MM14].

design of user interfaces or administrative procedures to implementation of digital hardware and verification of software and hardware protocols. Regular languages admit a very nice, clean and comprehensive theory where different formalisms such as automata, logic, regular expressions, semigroups and grammars are shown to be equivalent. The problem of learning automata from examples was introduced already in 1956 by Moore [Moo56]. This problem, like the problem of automaton minimization, is closely related to the Nerode right-congruence relation over words associated with every language or sequential function [Ner58]. This relation declares two *input histories* as equivalent if they lead to the same *future continuations*, thus providing a crisp characterization of what a *state* in a dynamical system is in terms of observable input-output behavior. All algorithms for learning automata from examples, starting with the seminal work of Gold [Gol72] and culminating in the well- known L^* algorithm of Angluin [Ang87] are based on this concept [DlH10].

One weakness, however, of the classical theory of regular languages is that it is rather "thin" and "flat". In other words, the alphabet is often considered as a small set devoid of any additional structure. On such alphabets, classical automata are good for expressing and exploring the temporal (sequential, monoidal) dimension embodied by the concatenation operations, but less good in expressing "horizontal" relationships. To make this statement more concrete, consider the verification of a system consisting of n automata running in parallel, making independent as well as synchronized transitions. To express the set of joint behaviors of this product of automata as a formal language, classical theory will force you to use the exponential alphabet of global states and indeed, a large part of verification is concerned with fighting this explosion using constructs such as BDDs and other logical forms that exploit the sparse interaction among components. This is done, however, without a real interaction with classical formal language theory (one exception is the theory of *traces* [DR95] which attempts to treat this issue but in a very restricted context).

These and other considerations led us to use *symbolic automata* as a generic framework for recognizing languages over large alphabets where transitions outgoing from a state are labeled, semantically speaking, by *subsets* of the alphabet. These subsets are expressed syntactically according to the specific alphabet used: Boolean formulae when $\Sigma = \mathbb{B}^n$ or by some classes of inequalities when $\Sigma \subseteq \mathbb{N}$ or $\Sigma \subseteq \mathbb{R}$. Determinism and completeness of the transition relation, which are crucial for learning and minimization, can be enforced by requiring that the subsets of Σ that label the transitions outgoing from a given state form a *partition* of the alphabet. Such symbolic automata have been used in the past for Boolean vectors [HJJ$^+$95] and have been studied extensively in recent years as acceptors and transducers where transitions are guarded by predicates of various theories [HV11, VHL$^+$12].

Readers working on program verification or hybrid automata are, of course, aware of automata with symbolic transition guards but it should be noted that in the model that we use, *no auxiliary variables* are added to the automaton. Let us stress this point by looking at a popular extension of automata to infinite alphabets, initiated in [KF94] using *register automata* to accept *data languages* (see [BLP10] for a good exposition of theoretical properties and [HSJC12] for learning algorithms). In that framework, the automaton is augmented with additional registers that can store some input letters. The registers can then be compared with newly-read letters and influence transitions. With register automata one can express, for example, the requirement that the password at login is the same as the password at sign-up. This very restricted use of memory makes register automata much simpler than more notorious automata with variables whose emptiness problem is typically undecidable. The downside is that beyond *equality* they do not really exploit the potential richness of the alphabets and their corresponding theories.

Our approach is different: we do allow the *values* of the input symbols to influence transitions via predicates, possibly of a restricted complexity. These predicates involve domain *constants* and they partition the alphabet into finitely many classes. For example, over the integers a state may have transitions labeled by conditions of the form $c_1 \leq x \leq c_2$ which give real (but of limited resolution) access to the input domain. On the other hand, we insist on a finite (and small) memory so that the exact value of x *cannot* be registered and has no future influence beyond the transition it has triggered. Many control systems, artificial (sequential machines working on quantized numerical inputs) as well as natural (central nervous system, the cell), are believed to operate in this manner. The automata that we use, like the symbolic automata and transducers studied in [HV11, VHL$^+$12, VB12], are geared toward languages recognized by automata having a large alphabet and a relatively-small state space.

We then develop a symbolic version of Angluin's L^* algorithm for learning regular sets from queries and counter-examples whose output is a symbolic automaton. The main difference relative to the concrete algorithm is that in the latter, every transition $\delta(q, a)$ in a conjectured automaton has at least one word in the sample that exercises it. In the symbolic case, a transition $\delta(q, \boldsymbol{a})$ where $\boldsymbol{a}$ stands for a *set* of concrete symbols, will be backed up in the sample only by a *subset* of $\boldsymbol{a}$. Thus, unlike concrete algorithms where a counter-example always leads to a discovery of one or more new states, in our algorithm it may sometimes only modify the boundaries between partition blocks without creating new states. There are some similarities between our work and another recent adaptation of the L^* algorithm to symbolic automata, the Σ^* algorithm of [BB13]. This work is incomparable to ours as they use a richer model of transducers and more general predicates on inputs and outputs. Consequently their termination result is weaker and is relative to the termination of the counter-example guided abstraction refinement procedure.

The rest of the paper is organized as follows. In Section 1 we provide a quick summary of learning algorithms over small alphabets. In Section 2 we define symbolic automata and then extend the structure which underlies all automaton learning algorithms, namely the *observation table*, to be symbolic, where symbolic letters represent sets, and where entries in the table are supported only by partial evidence. In Section 4 we write down a symbolic learning algorithm, an adaptation of L^* for totally ordered alphabets such as $\mathbb{R}$ or $\mathbb{N}$ and illustrate the behavior of a prototype implementation. The algorithm is then extended to languages over partially ordered alphabets such as $\mathbb{N}^d$ and $\mathbb{R}^d$ where in each state, the labels of outgoing transition from a monotone partition of the alphabet are represented by finitely many points. We conclude by a discussion of past and future work.

1. Learning Regular Sets

We briefly survey Angluin's L^* algorithm [Ang87] for learning regular sets from membership queries and counter-examples, with slightly modified definitions to accommodate for its symbolic extension. Let Σ be a finite alphabet and let Σ^* be the set of sequences (words) over Σ. Any order relation $<$ over Σ can be naturally lifted to a lexicographic order over Σ^*. With a language $L \subseteq \Sigma^*$ we associate a *characteristic function* $f : \Sigma^* \to \{+, -\}$, where $f(w) = +$ if the word $w \in \Sigma^*$ belongs to L and $f(w) = -$, otherwise.

A *deterministic finite automaton* over Σ is a tuple $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$, where Q is a non-empty finite set of *states*, $q_0 \in Q$ is the *initial* state, $\delta : Q \times \Sigma \to Q$ is the *transition function*, and $F \subseteq Q$ is the set of *final* or *accepting* states. The transition function δ can be extended to $\delta : Q \times \Sigma^* \to Q$, where $\delta(q, \epsilon) = q$, and $\delta(q, u \cdot a) = \delta(\delta(q, u), a)$ for $q \in Q$, $a \in \Sigma$ and $u \in \Sigma^*$. A word $w \in \Sigma^*$ is *accepted* by $\mathcal{A}$ if $\delta(q_0, w) \in F$, otherwise w is *rejected.* The language recognized by $\mathcal{A}$ is the set of all accepted words and is denoted by $L(\mathcal{A})$.

Learning algorithms, represented by the *learner*, are designed to infer an unknown regular language L (the *target language*). The learner aims to construct a finite automaton that recognizes L by gathering information from the *teacher.* The *teacher* knows L and can provide information about it. It can answer two types of queries: *membership queries*, i.e., whether a given word belongs to the target language, and *equivalence queries*, i.e., whether a conjectured automaton suggested by the learner is the right one. If this automaton fails to accept L the teacher responds to the equivalence query by a *counter-example*, a word miss-classified by the conjectured automaton.

In the L^* algorithm, the learner starts by asking membership queries. All information provided is suitably gathered in a table structure, the *observation table.* Then, when the information is sufficient, the learner constructs a *hypothesis automaton* and poses an equivalence query to the teacher. If the answer is positive then the algorithm terminates and returns the conjectured automaton. Otherwise the learner accommodates the information provided by the counter-example into the table, asks additional membership queries until it can suggest a new hypothesis and so on, until termination.

A prefix-closed set $S \uplus R \subset \Sigma^*$ is a *balanced* Σ*-tree* if $\forall a \in \Sigma$: 1) For every $s \in S$ $s \cdot a \in S \cup R$, and 2) For every $r \in R$, $r \cdot a \notin S \cup R$. Elements of R are called *boundary elements* or *leaves.* [1]

Definition 1.1 (Observation Table)**.** An *observation table* is a tuple $T = (\Sigma, S, R, E, f)$ such that Σ is an alphabet, $S \cup R$ is a balanced Σ-tree, E is a subset of Σ^* and $f : (S \cup R) \cdot E \to \{-, +\}$ is the classification function, a restriction of the characteristic function of the target language L.

The set $(S \cup R) \cdot E$ is the *sample* associated with the table, that is, the set of words whose membership is known. The elements of S admit a tree structure isomorphic to a *spanning tree* of the transition graph rooted in the initial state. Each $s \in S$ corresponds to a state q of the automaton for which s is an *access sequence*, one of the shortest words that lead from the initial state to q. The elements of R should tell us about the back- and cross-edges in the automaton and the elements of E are "experiments" that should be sufficient to distinguish between states. This works by associating with every $s \in S \cup R$ a specialized classification function $f_s : E \to \{-, +\}$, defined as $f_s(e) = f(s \cdot e)$, which characterizes the row of the observation table labeled by s. To build an automaton from a table it should satisfy certain conditions.

Definition 1.2 (Closed, Reduced and Consistent Tables)**.** An observation table T is:

- Closed if for every $r \in R$, there exists an $s \in S$, such that $f_r = f_s$;
- Reduced if for every $s, s' \in S$ $f_s \neq f_{s'}$;
- Consistent if for every $s, s' \in S$, $f_s = f_{s'}$ implies $f_{s \cdot a} = f_{s' \cdot a}, \forall a \in \Sigma$.

Note that a reduced table is trivially consistent and that for a closed and reduced table we can define a function $g : R \to S$ mapping every $r \in R$ to the unique $s \in S$ such that

[1]We use $\uplus$ for disjoint union.

$f_s = f_r$. From such an observation table $T = (\Sigma, S, R, E, f)$ one can construct an automaton $\mathcal{A}_T = (\Sigma, Q, q_0, \delta, F)$ where $Q = S$, $q_0 = \epsilon$, $F = \{s \in S : f_s(\epsilon) = +\}$ and

$$\delta(s,a) = \begin{cases} s \cdot a & \text{when } s \cdot a \in S \\ g(s \cdot a) & \text{when } s \cdot a \in R \end{cases}$$

The learner attempts to keep the table closed at all times. The table is not closed when there is some $r \in R$ such that f_r is different from f_s for all $s \in S$. To close the table, the learner moves r from R to S and adds the Σ-successors of r, i.e., all words $r \cdot a$ for $a \in \Sigma$, to R. The extended table is then filled up by asking membership queries until it becomes closed.

Variants of the L^* algorithm differ in the way they treat counter-examples, as described in more detail in [BR04]. The original algorithm [Ang87] adds all the *prefixes* of the counter-example to S and thus possibly creating inconsistency that should be fixed. The version proposed in [MP95] for learning ω-regular languages adds all the *suffixes* of the counter-example to E. The advantage of this approach is that the table always remains consistent and reduced with S corresponding exactly to the set of states. A disadvantage is the possible introduction of redundant columns that do not contribute to further discrimination between states. The symbolic algorithm that we develop in this paper is based on an intermediate variant, referred to in [BR04] as the *reduced observation algorithm*, where some prefixes of the counter-example are added to S and some suffixes are added to E.

Example 1.3. We illustrate the behavior of the L^* algorithm while learning a language L over $\Sigma = \{1, 2, 3, 4, 5\}$. We use the tuple $(w, +)$ to indicate a counter-example $w \in L$ rejected by the conjectured automaton, and $(w, -)$ for the opposite case. Initially, the observation table is $T_0 = (\Sigma, S, R, E, f)$ with $S = E = \{\epsilon\}$ and $R = \Sigma$ and we ask membership queries for all words in $(S \cup R) \cdot E$ to obtain table T_0, shown in Fig. 1. The table is not closed so we move word 1 to S, add its continuations, $1 \cdot \Sigma$ to R and ask membership queries to obtain table T_1 which is now closed. We construct an hypothesis $\mathcal{A}_1$ (Fig. 2) from this table, and pose an equivalence query for which the teacher returns counter-example $(3 \cdot 1, -)$. We add $3 \cdot 1$ and its prefix 3 to set S and add all their continuations to the boundary of the table resulting table T_2 of Fig. 1. This table is not consistent: two elements ϵ and 3 in S are equivalent but their successors 1 and $3 \cdot 1$ are not. In order to distinguish the two strings we add to E the suffix 1 and end up with a closed and consistent table T_3. The new hypothesis for this table is $\mathcal{A}_3$, shown in Fig. 2. Once more the equivalence query will return a counter-example, $(1 \cdot 3 \cdot 3, -)$. We again add the counter-example and prefixes to the table, ask membership queries to fill in the table and solve the inconsistency that appears for 1 and $1 \cdot 3$ by adding suffix 3 to the table. The table corresponds now to the correct hypothesis $\mathcal{A}_5$, and the algorithm terminates. $\square$

2. Symbolic Automata

In this section we introduce the variant of *symbolic automata* that we use. Symbolic automata [HV11, VB12] give a more succinct representation for languages over large finite alphabets and can also represent languages over infinite alphabets such as $\mathbb{N}$, $\mathbb{R}$, or $\mathbb{R}^n$. The size of a standard automaton for a language grows linearly with the size of the alphabet and so does the complexity of learning algorithms such as L^*. As we shall see, symbolic automata admit a variant of the L^* algorithm whose complexity is independent of the alphabet size.

T_0

	ϵ
ϵ	-
1	+
2	+
3	-
4	-
5	-

T_1

	ϵ
ϵ	-
1	+
2	+
3	-
4	-
5	-
$1 \cdot 1$	-
$1 \cdot 2$	-
$1 \cdot 3$	+
$1 \cdot 4$	-
$1 \cdot 5$	-

T_2

	ϵ
ϵ	-
1	+
3	-
$3 \cdot 1$	-
2	+
4	-
5	-
$1 \cdot 1$	-
$1 \cdot 2$	-
$1 \cdot 3$	+
$1 \cdot 4$	-
$\vdots$	

T_3

	ϵ	1
ϵ	-	+
1	+	-
3	-	-
$3 \cdot 1$	-	-
2	+	-
4	-	-
5	-	-
$1 \cdot 1$	-	-
$1 \cdot 2$	-	-
$1 \cdot 3$	+	-
$1 \cdot 4$	-	-
$\vdots$		

T_4

	ϵ	1
ϵ	-	+
1	+	-
3	-	-
$1 \cdot 3$	+	-
$3 \cdot 1$	-	+
$1 \cdot 3 \cdot 3$	-	-
2	+	-
4	-	-
5	-	-
$1 \cdot 1$	-	-
$1 \cdot 2$	-	-
$\vdots$		

T_5

	ϵ	1	3
ϵ	-	+	-
1	+	-	+
3	-	-	-
$1 \cdot 3$	+	-	-
$3 \cdot 1$	-	+	-
$1 \cdot 3 \cdot 3$	-	-	-
2	+	-	+
4	-	-	-
5	-	-	-
$1 \cdot 1$	-	-	-
$1 \cdot 2$	-	-	-
$\vdots$			

Figure 1. Observation tables for Example 1.3.

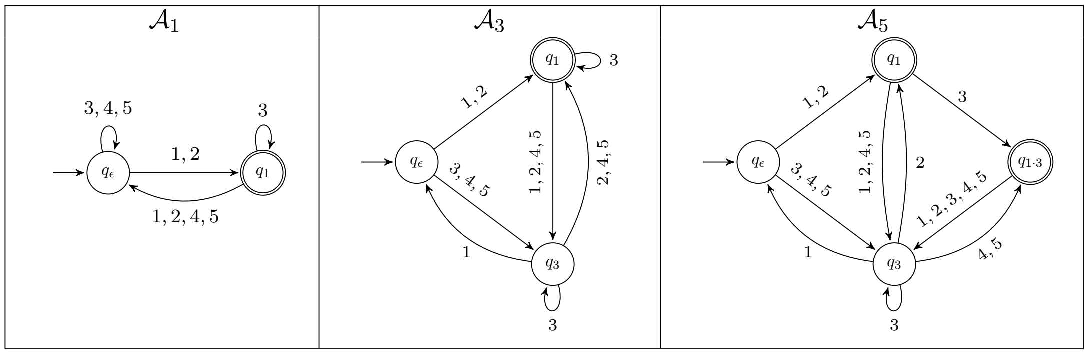


Figure 2. Hypotheses for Example 1.3

Let Σ be a large, possibly infinite, alphabet, to which we will refer from now on as the *concrete* alphabet. We define a symbolic automaton to be an automaton over Σ where each state has a small number of outgoing transitions labeled by symbols that represent subsets of Σ. For every state, these subsets form a (possibly different) *partition* of Σ and hence the automaton is complete and deterministic. We start with an arbitrary alphabet viewed as an unstructured set and present the concept in purely semantic manner before we move to ordered sets and inequalities in subsequent sections.

Let $\boldsymbol{\Sigma}$ be a finite alphabet, that we call the *symbolic alphabet* and its elements *symbolic letters* or *symbols*. Let $\psi : \Sigma \to \boldsymbol{\Sigma}$ map concrete letters into symbolic ones. The Σ-semantics of a *symbolic letter* $\boldsymbol{a} \in \boldsymbol{\Sigma}$ is defined as $[\boldsymbol{a}]_\psi = \{a \in \Sigma : \psi(a) = \boldsymbol{a}\}$ and the set $\{[\boldsymbol{a}]_\psi : \boldsymbol{a} \in \boldsymbol{\Sigma}\}$ forms a *partition* of Σ. We will often omit ψ from the notation and use $[\boldsymbol{a}]$ when ψ, which is always present, is clear from the context. The Σ-semantics can be extended to symbolic words of the form $\boldsymbol{w} = \boldsymbol{a}_1 \cdot \boldsymbol{a}_2 \cdots \boldsymbol{a}_k \in \boldsymbol{\Sigma}^*$ as the concatenation of the concrete one-letter languages associated with the respective symbolic letters or, recursively speaking, $[\boldsymbol{\epsilon}] = \{\epsilon\}$ and $[\boldsymbol{w} \cdot \boldsymbol{a}] = [\boldsymbol{w}] \cdot [\boldsymbol{a}]$ for $\boldsymbol{w} \in \boldsymbol{\Sigma}^*$, $\boldsymbol{a} \in \boldsymbol{\Sigma}$.

Definition 2.1 (Symbolic Automaton)**.** A *deterministic symbolic automaton* is a tuple $\boldsymbol{\mathcal{A}} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \delta, \boldsymbol{\delta}, q_0, F)$, where

- Σ is the input alphabet,

- $\boldsymbol{\Sigma}$ is a finite alphabet, decomposable into $\boldsymbol{\Sigma} = \biguplus_{q \in Q} \boldsymbol{\Sigma}_q$,
- $\psi = \{\psi_q : q \in Q\}$ is a family of surjective functions $\psi_q : \Sigma \to \boldsymbol{\Sigma}_q$,
- Q is a finite set of states,
- $\delta : Q \times \Sigma \to Q$ and $\boldsymbol{\delta} : Q \times \boldsymbol{\Sigma} \to Q$ are the concrete and symbolic transition functions respectively, such that $\delta(q,a) = \boldsymbol{\delta}(q, \psi_q(a))$,
- q_0 is the initial state and F is a set of accepting states.

The transition function is extended to words as in the concrete case and the symbolic automaton can be viewed as an acceptor of a concrete language. When at q and reading a concrete letter a, the automaton will take the transition $\boldsymbol{\delta}(q, \boldsymbol{a})$ where $\boldsymbol{a}$ is the *unique* element of $\boldsymbol{\Sigma}_q$ satisfying $a \in [\boldsymbol{a}]$. Hence $L(\boldsymbol{\mathcal{A}})$ consists of all concrete words whose run leads from q_0 to a state in F. A language L over alphabet Σ is symbolic recognizable if there exists a symbolic automaton $\boldsymbol{\mathcal{A}}$ such that $L = L(\boldsymbol{\mathcal{A}})$.
Remark: The association of a *symbolic language* with a symbolic automaton is more subtle because we allow different partitions of Σ and hence different symbolic input alphabets at different states. The transition to be taken while being in a state q and reading a symbol $\boldsymbol{a} \notin \boldsymbol{\Sigma}_q$ is well defined only when $[\boldsymbol{a}] \subseteq [\boldsymbol{a'}]$ for some $\boldsymbol{a'} \in \boldsymbol{\Sigma}_q$. Such a model can be transformed into an automaton which is complete over a symbolic alphabet which is common to all states as follows. Let

$$\boldsymbol{\Sigma}' = \prod_{q \in Q} \boldsymbol{\Sigma}_q, \text{ with the } \Sigma\text{-semantics } [(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)] = [\boldsymbol{a_1}] \cap \ldots \cap [\boldsymbol{a}_n],$$

and let $\tilde{\boldsymbol{\Sigma}} = \{\boldsymbol{b} \in \boldsymbol{\Sigma}' : [\boldsymbol{b}] \neq \emptyset\}$. Then we define $\widetilde{\boldsymbol{\mathcal{A}}} = (\tilde{\boldsymbol{\Sigma}}, Q, \tilde{\delta}, q_0, F)$ where, by construction, for every $\boldsymbol{b} \in \tilde{\boldsymbol{\Sigma}}$ and every $q \in Q$, there is a unique $\boldsymbol{a} \in \boldsymbol{\Sigma}_q$ such that $[\boldsymbol{b}] \subseteq [\boldsymbol{a}]$ and hence one can define the transition function as $\tilde{\boldsymbol{\delta}}(q, \boldsymbol{b}) = \boldsymbol{\delta}(q, \boldsymbol{a})$. This model is more comfortable for language-theoretic studies but in the learning context it introduces an unnecessary blow-up in the alphabet size and the number of queries for every state. For this reason we stick in this paper to the Definition 2.1 which is more economical. A similar approach of state-local abstraction has been taken in [IHS13] for learning parameterized language. The construction of $\boldsymbol{\Sigma}'$ is similar to the minterm construction of [DV14] used to create a common alphabet in order to apply the minimization algorithm of Hopcroft to symbolic automata. Anyway, in our learning framework symbolic automata are used to read concrete and not symbolic words. □

It is straightforward that for a finite concrete alphabet Σ the set of languages accepted by symbolic automata coincides with the set of recognizable regular languages over Σ. Moreover, even when the alphabet is infinite, closure under Boolean operations is preserved.

Proposition 2.2 (Closure under Boolean Operations)**.** *Languages accepted by deterministic symbolic automata are effectively closed under Boolean operations.*

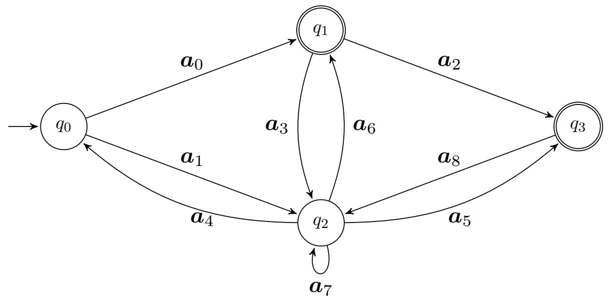
Proof. Closure under complement is immediate by complementing the set of accepting states. For intersection the standard product construction is adapted as follows. Let L_1, L_2 be languages recognized by the symbolic automata $\boldsymbol{\mathcal{A}}_1 = (\Sigma, \boldsymbol{\Sigma}_1, \psi_1, Q_1, \delta_1, \boldsymbol{\delta}_1, q_{01}, F_1)$, and $\boldsymbol{\mathcal{A}}_2 = (\Sigma, \boldsymbol{\Sigma}_2, \psi_2, Q_2, \delta_2, \boldsymbol{\delta}_2, q_{02}, F_2)$, respectively. Let $\boldsymbol{\mathcal{A}} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \delta, \boldsymbol{\delta}, q_0, F)$, where

- $Q = Q_1 \times Q_2$, $q_0 = (q_{01}, q_{02})$, $F = F_1 \times F_2$,
- For every $(q_1, q_2) \in Q$
  - $\boldsymbol{\Sigma}_{(q_1,q_2)} = \{(\boldsymbol{a}_1, \boldsymbol{a}_2) \in \boldsymbol{\Sigma}_1 \times \boldsymbol{\Sigma}_2 \mid [\boldsymbol{a}_1] \cap [\boldsymbol{a}_2] \neq \emptyset\}$
  - $\psi_{(q_1,q_2)}(a) = (\psi_{1,q_1}(a), \psi_{2,q_2}(a))$, $\forall a \in \Sigma$

– $\boldsymbol{\delta}((q_1, q_2), (\boldsymbol{a}_1, \boldsymbol{a}_2)) = (\boldsymbol{\delta}_1(q_1, \boldsymbol{a}_1), \boldsymbol{\delta}_2(q_2, \boldsymbol{a}_2))$, $\forall(\boldsymbol{a}_1, \boldsymbol{a}_2) \in \boldsymbol{\Sigma}_{(q_1,q_2)}$

It is sufficient to observe that the corresponding implied concrete automata $\mathcal{A}_1$, $\mathcal{A}_2$ and $\mathcal{A}$ satisfy $\delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$ and the standard proof that $L(\mathcal{A}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$ follows. Closure under union and set difference is then evident. □

The above product construction is used to implement equivalence queries where both the target language and the current conjecture are represented by symbolic automata. A counter-example is found by looking for a shortest path in the product automaton from the initial state to a state in $F_1 \times (Q_2 - F_2) \cup (Q_1 - F_1) \times F_2$ and selecting a lexicographically minimal concrete word along that path.



$\boldsymbol{\delta}$	q_0	q_1	q_2	q_3
q_0	–	$\boldsymbol{a}_0$	$\boldsymbol{a}_1$	–
q_1	–	–	$\boldsymbol{a}_3$	$\boldsymbol{a}_2$
q_2	$\boldsymbol{a}_4$	$\boldsymbol{a}_6$	$\boldsymbol{a}_7$	$\boldsymbol{a}_5$
q_3	–	–	$\boldsymbol{a}_8$	–

FIGURE 3. A symbolic automaton $\boldsymbol{\mathcal{A}}$ with its symbolic transition function.

Example 2.3. Figure 3 shows a symbolic automaton equivalent to automaton $\mathcal{A}_5$ of Figure 2. The symbolic alphabets for the states are $\boldsymbol{\Sigma}_{q_0} = \{\boldsymbol{a}_0, \boldsymbol{a}_1\}$, $\boldsymbol{\Sigma}_{q_1} = \{\boldsymbol{a}_2, \boldsymbol{a}_3\}$, $\boldsymbol{\Sigma}_{q_2} = \{\boldsymbol{a}_4, \boldsymbol{a}_5, \boldsymbol{a}_6, \boldsymbol{a}_7\}$, $\boldsymbol{\Sigma}_{q_3} = \{\boldsymbol{a}_8\}$, and the Σ-semantics for the symbols is $[\boldsymbol{a}_0] = \{1, 2\}$, $[\boldsymbol{a}_1] = \{3, 4, 5\}$, $[\boldsymbol{a}_2] = \{3\}$, $[\boldsymbol{a}_3] = \{1, 2, 4, 5\}$, etc.. The same automaton can accept a language over the uncountable alphabet $\Sigma = [0, 100) \subset \mathbb{R}$, defining ψ as shown in Figure 4.

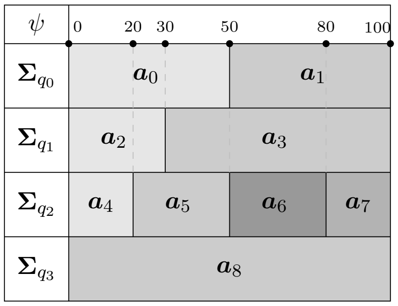


FIGURE 4. The concrete semantics of the symbols of automaton $\boldsymbol{\mathcal{A}}$ of Fig. 3, when defined over $\Sigma = [0, 100) \subseteq \mathbb{R}$.

3. SYMBOLIC OBSERVATION TABLES

In this section we adapt observation tables to the symbolic setting. They are similar to the concrete case with the additional notions of evidences and evidence compatibility.

Definition 3.1 (Balanced Symbolic Σ-Tree)**.** A *balanced symbolic* Σ*-tree* is a tuple $(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi)$ where

- $\boldsymbol{S} \uplus \boldsymbol{R}$ is a prefix-closed subset of $\boldsymbol{\Sigma}^*$
- $\boldsymbol{\Sigma} = \biguplus_{\boldsymbol{s}\in\boldsymbol{S}} \boldsymbol{\Sigma}_{\boldsymbol{s}}$ is a symbolic alphabet
- $\psi = \{\psi_{\boldsymbol{s}}\}_{\boldsymbol{s}\in\boldsymbol{S}}$ is a family of total surjective functions of the form $\psi_{\boldsymbol{s}} : \Sigma \to \boldsymbol{\Sigma}_{\boldsymbol{s}}$.

It is required that for every $\boldsymbol{s} \in \boldsymbol{S}$ and $\boldsymbol{a} \in \boldsymbol{\Sigma}_{\boldsymbol{s}}$, $\boldsymbol{s} \cdot \boldsymbol{a} \in \boldsymbol{S} \cup \boldsymbol{R}$ and for any $\boldsymbol{r} \in \boldsymbol{R}$ and $\boldsymbol{a} \in \boldsymbol{\Sigma}$, $\boldsymbol{r} \cdot \boldsymbol{a} \notin \boldsymbol{S} \cup \boldsymbol{R}$. Elements of $\boldsymbol{R}$ are called boundary elements of the tree.

We will use observation tables whose rows are symbolic words and hence an entry in the table will constitute a statement about the inclusion or exclusion of a large *set* of concrete words in the language. We will not ask membership queries concerning all those concrete words, but only for a small representative subset that we call *evidence*.

Definition 3.2 (Symbolic Observation Table)**.** A *symbolic observation table* is a tuple $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$ such that

- Σ is an alphabet,
- $(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi)$ is a balanced symbolic Σ-tree (with $\boldsymbol{R}$ being its *boundary*),
- E is a subset of Σ^*,
- $\boldsymbol{f} : (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E \to \{-, +\}$ is the symbolic classification function
- $\mu : (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E \to 2^{\Sigma^*} - \{\emptyset\}$ is an evidence function satisfying $\mu(\boldsymbol{w}) \subseteq [\boldsymbol{w}]$. The image of the evidence function is prefix-closed: $w \cdot a \in \mu(\boldsymbol{w} \cdot \boldsymbol{a}) \Rightarrow w \in \mu(\boldsymbol{w})$.

As for the concrete case we use $\boldsymbol{f}_{\boldsymbol{s}} : E \to \{-, +\}$ to denote the partial evaluation of $\boldsymbol{f}$ to some symbolic word $\boldsymbol{s} \in \boldsymbol{S} \cup \boldsymbol{R}$, such that, $\boldsymbol{f}_{\boldsymbol{s}}(e) = \boldsymbol{f}(\boldsymbol{s} \cdot e)$. Note that the set E consists of *concrete* words but this poses no problem because elements of E are used only to distinguish between states and do not participate in the derivation of the symbolic automaton from the table. Concatenation of a symbolic word and a concrete one follows concatenation of symbolic words as defined above where each concrete letter a is considered as a symbolic letter $\boldsymbol{a}$ with $[\boldsymbol{a}] = \{a\}$ and $\mu(\boldsymbol{a}) = a$. The notions of closed, consistent and reduced table are similar to the concrete case.

The set $\boldsymbol{M_T} = (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E$ is called the *symbolic sample* associated with $\boldsymbol{T}$. We require that for each word $\boldsymbol{w} \in \boldsymbol{M_T}$ there is at least one concrete $w \in \mu(\boldsymbol{w})$ whose membership in L, denoted by $f(w)$, is known. The set of such words is called the *concrete sample* and is defined as $M_{\boldsymbol{T}} = \{s \cdot e : s \in \mu(\boldsymbol{s}), \boldsymbol{s} \in \boldsymbol{S} \cup \boldsymbol{R}, e \in E\}$. A table where all evidences of the same symbolic word admit the same classification is called *evidence-compatible.*

Definition 3.3 (Table Conditions)**.** A table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$ is

- Closed if $\forall \boldsymbol{r} \in \boldsymbol{R}$, $\exists \boldsymbol{s} = g(\boldsymbol{r}) \in \boldsymbol{S}$, $\boldsymbol{f}_{\boldsymbol{r}} = \boldsymbol{f}_{\boldsymbol{s}}$,
- Reduced if $\forall \boldsymbol{s}, \boldsymbol{s'} \in \boldsymbol{S}$, $\boldsymbol{f}_{\boldsymbol{s}} \neq \boldsymbol{f}_{\boldsymbol{s'}}$,
- Consistent if $\forall \boldsymbol{s}, \boldsymbol{s}' \in \boldsymbol{S}$, $\boldsymbol{f}_{\boldsymbol{s}} = \boldsymbol{f}_{\boldsymbol{s'}}$ implies $\boldsymbol{f}_{\boldsymbol{s}\cdot\boldsymbol{a}} = \boldsymbol{f}_{\boldsymbol{s'}\cdot\boldsymbol{a}}, \forall \boldsymbol{a} \in \boldsymbol{\Sigma}_{\boldsymbol{s}}$.
- Evidence compatible if $\forall \boldsymbol{w} \in \boldsymbol{M_T}$, $\forall w_1, w_2 \in \mu(\boldsymbol{w}), f(w_1) = f(w_2)$.

When a table $\boldsymbol{T}$ is evidence compatible the symbolic classification function $\boldsymbol{f}$ can be defined for every $\boldsymbol{s} \in (\boldsymbol{S} \cup \boldsymbol{R})$ and $e \in E$ as $\boldsymbol{f}(\boldsymbol{s} \cdot e) = f(s \cdot e)$, $s \in \mu(\boldsymbol{s})$.

Theorem 3.4 (Automaton from Table)**.** *From a closed, reduced and evidence compatible table one can construct a deterministic symbolic automaton compatible with the concrete sample.*

Proof. The proof is similar to the concrete case. Let $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$ be such a table, which is reduced and closed and thus a function $g : \boldsymbol{R} \to \boldsymbol{S}$ such that $g(\boldsymbol{r}) = \boldsymbol{s}$ iff $\boldsymbol{f_r} = \boldsymbol{f_s}$ is well defined. The automaton derived from the table is then $\boldsymbol{\mathcal{A}_T} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \boldsymbol{\delta}, q_0, F)$ where:

- $Q = \boldsymbol{S}$, $q_0 = \boldsymbol{\epsilon}$
- $F = \{\boldsymbol{s} \in \boldsymbol{S} \mid \boldsymbol{f_s}(\epsilon) = +\}$
- $\boldsymbol{\delta} : Q \times \boldsymbol{\Sigma} \to Q$ is defined as $\boldsymbol{\delta}(\boldsymbol{s}, \boldsymbol{a}) = \begin{cases} \boldsymbol{s} \cdot \boldsymbol{a} & \text{when } \boldsymbol{s} \cdot \boldsymbol{a} \in \boldsymbol{S} \\ g(\boldsymbol{s} \cdot \boldsymbol{a}) & \text{when } \boldsymbol{s} \cdot \boldsymbol{a} \in \boldsymbol{R} \end{cases}$

By construction and like the L^* algorithm, $\boldsymbol{\mathcal{A}_T}$ classifies correctly the symbolic sample and, due to evidence compatibility, this holds also for the concrete sample. □

4. Learning Languages over Ordered Alphabets

In this section we present a symbolic learning algorithm starting with an intuitive verbal description. The algorithmic scheme is similar to the concrete L^* algorithm but differs in the treatment of counter-examples and the new concept of evidence compatibility. Whenever the table is not closed, $\boldsymbol{S} \cup \boldsymbol{R}$ is extended until closure. Then a conjectured automaton $\boldsymbol{\mathcal{A}_T}$ is constructed and an equivalence query is posed. If the answer is positive we are done. Otherwise, the teacher provides a counter-example leading to the extension of $\boldsymbol{S} \cup \boldsymbol{R}$ and/or E. Whenever such an extension occurs, additional membership queries are posed to fill the table. The table is always kept evidence compatible and reduced except temporarily during the processing of counter-examples.

From now on we assume Σ to be a *totally ordered* alphabet with a minimal element a_0 and restrict ourselves to symbolic automata where the concrete semantics for every symbolic letter is an interval. In the case of a dense order like in $\mathbb{R}$, we assume the intervals to be left-closed and right-open. The order on the alphabet can be extended naturally to a lexicographic order on Σ^*. Our algorithm also assumes that the teacher provides a counter-example of minimal length which is minimal with respect to the lexicographic order. This strong assumption improves the performance of the algorithm and its relaxation is discussed in Section 7.

The rows of the observation table consist of symbolic words because we want to group together all concrete letters and words that are assumed to induce the same behavior in the automaton. New symbolic letters are introduced in two occasions: when a new state is discovered or when a partition is modified due to a counter-example. In both cases we set the concrete semantics $[\boldsymbol{a}]$ to the largest possible subset of Σ, given the current evidence (in the first case it will be Σ). As an evidence we always select the smallest possible $a \in [\boldsymbol{a}]$ (a_0 when $[\boldsymbol{a}] = \Sigma$). The choice of the right evidences is a key point for the performance of the algorithm as we want to keep the concrete sample as small as possible and avoid posing unnecessary queries. For infinite concrete alphabets this choice of evidence guarantees termination.

The initial symbolic table is $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$, where $\boldsymbol{\Sigma} = \{\boldsymbol{a_0}\}$, $[\boldsymbol{a}_0] = \Sigma$, $\boldsymbol{S} = \{\boldsymbol{\epsilon}\}$, $\boldsymbol{R} = \{\boldsymbol{a_0}\}$, $E = \{\epsilon\}$, and $\mu(\boldsymbol{a_0}) = \{a_0\}$. The table is filled by membership queries concerning ϵ and a_0. Whenever $\boldsymbol{T}$ is not closed, there is some $\boldsymbol{r} \in \boldsymbol{R}$ such that $\boldsymbol{f_r} \neq \boldsymbol{f_s}$ for every $\boldsymbol{s} \in \boldsymbol{S}$. To close the table we move $\boldsymbol{r}$ from $\boldsymbol{R}$ to $\boldsymbol{S}$, recognizing it as a new state, and checking the behavior of its continuation. To this end we add to $\boldsymbol{R}$ the word $\boldsymbol{r'} = \boldsymbol{r} \cdot \boldsymbol{a}$, where $\boldsymbol{a}$ is a new symbolic letter with $[\boldsymbol{a}] = \Sigma$. We extend the evidence function by letting

Algorithm 1 The symbolic algorithm

```
procedure SYMBOLIC
    $learned = false$
    Initialize the table $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu)$
      $\boldsymbol{\Sigma} = \{\boldsymbol{a}\}$; $\psi_\epsilon(a) = \boldsymbol{a}, \forall a \in \Sigma$
      $\boldsymbol{S} = \{\boldsymbol{\epsilon}\}$; $\boldsymbol{R} = \{\boldsymbol{a}\}$; $E = \{\epsilon\}$
      $\mu(\boldsymbol{a}) = \{a_0\}$
      Ask MQ on $\epsilon$ and $a_0$ to fill $\boldsymbol{f}$
    if $\boldsymbol{T}$ is not closed then
        CLOSE
    end if
    repeat
        if $EQ(\mathcal{A}_{\boldsymbol{T}})$ then                      ▷ $\mathcal{A}_{\boldsymbol{T}}$ is correct
            $learned = true$
        else                                     ▷ A counter-example $w$ is provided
            $M = M \cup \{w\}$
            COUNTER-EX($w$)                           ▷ Process counter-example
        end if
    until $learned$
end procedure
```

Procedure 2 Close the table

```
procedure CLOSE
    while $\exists \boldsymbol{r} \in \boldsymbol{R}$ such that $\forall \boldsymbol{s} \in \boldsymbol{S}$, $\boldsymbol{f_r} \neq \boldsymbol{f_s}$ do
        $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{r}\}$                          ▷ $\boldsymbol{r}$ becomes a new state
        $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}_{\text{new}}\}$
        $\psi' = \psi \cup \{\psi_{\boldsymbol{r}}\}$ with $\psi_{\boldsymbol{r}}(a) = \boldsymbol{a}_{\text{new}}, \forall a \in \Sigma$
        $\boldsymbol{R'} = (\boldsymbol{R} - \{\boldsymbol{r}\}) \cup \{\boldsymbol{r} \cdot \boldsymbol{a}_{\text{new}}\}$
        $\mu(\boldsymbol{r} \cdot \boldsymbol{a}_{\text{new}}) = \mu(\boldsymbol{r}) \cdot a_0$
        Ask MQ for all words in $\{\mu(\boldsymbol{r} \cdot \boldsymbol{a}_{\text{new}}) \cdot e : e \in E\}$
        $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma'}, \boldsymbol{S'}, \boldsymbol{R'}, \psi', E, \boldsymbol{f'}, \mu')$
    end while
end procedure
```

$\mu(\boldsymbol{r'}) = \mu(\boldsymbol{r}) \cdot a_0$, assuming that all elements of Σ behave as a_0 from $\boldsymbol{r}$. Once $\boldsymbol{T}$ is closed we construct a hypothesis automaton as described in the proof of Theorem 3.4.

When a counter-example w is presented, it is of course not part of the concrete sample. A miss-classified word in the conjectured automaton means that somewhere a wrong transition is taken. Hence w admits a factorization $w = u \cdot b \cdot v$ where $u \in \Sigma^*$ and $b \in \Sigma$ is where the first wrong transition is taken. Obviously we do not know u and b in advance but know that this happens in the following two cases. Either b leads to an undiscovered state in the automaton of the target language, or letter b does not belong to the interval it was assumed to belong in the conjectured automaton. The latter case happens only when b does not belong to the evidence function. Since counter-example w is minimal, it admits a factorization $w = u \cdot b \cdot v$, where u is the largest prefix of w such that $u \in \mu(\boldsymbol{u})$ for some

Procedure 3 Process counter-example

```
procedure COUNTER-EX($w$)
    Find a factorization $w = u \cdot b \cdot v$, $b \in \Sigma$, $u$, $v \in \Sigma^*$ such that
        $\exists \boldsymbol{u} \in \boldsymbol{M_T}$, $u \in \mu(\boldsymbol{u})$ and $\forall \boldsymbol{u'} \in \boldsymbol{M_T}$, $u \cdot b \notin \mu(\boldsymbol{u'})$
    if $\boldsymbol{u} \in \boldsymbol{S}$ then                                  ▷ $\boldsymbol{u}$ is already a state
        Find $\boldsymbol{a} \in \boldsymbol{\Sigma_u}$ such that $b \in [\boldsymbol{a}]$                        ▷ refine $[\boldsymbol{a}]$
        $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}_{\text{new}}\}$
        $\boldsymbol{R'} = \boldsymbol{R} \cup \{\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}\}$
        $\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) = \mu(\boldsymbol{u}) \cdot b$
        Ask MQ for all words in $\{\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) \cdot e : e \in E\}$
        $\psi'_{\boldsymbol{u}}(a) = \begin{cases} \psi_{\boldsymbol{u}}(a) & \text{if } a \notin [\boldsymbol{a}] \\ \boldsymbol{a}_{\text{new}} & \text{if } a \in [\boldsymbol{a}] \text{ and } a \geq b \\ \boldsymbol{a} & \text{otherwise} \end{cases}$
        $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma'}, \boldsymbol{S}, \boldsymbol{R'}, \psi', E, \boldsymbol{f'}, \mu')$
    else                                                     ▷ $\boldsymbol{u}$ is in the boundary
        $\boldsymbol{S'} = \boldsymbol{S} \cup \{\boldsymbol{u}\}$                                  ▷ and becomes a state
        if $b = a_0$ then
            $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}_{\text{new}}\}$
            $\psi' = \psi \cup \{\psi_{\boldsymbol{u}}\}$, with $\psi_{\boldsymbol{u}}(a) = \boldsymbol{a}_{\text{new}}, \forall a \in \Sigma$
            $\boldsymbol{R'} = (\boldsymbol{R} - \{\boldsymbol{u}\}) \cup \{\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}\}$
            $E' = E \cup \{\text{suffixes of } b \cdot v\}$
            $\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) = \mu(\boldsymbol{u}) \cdot a_0$
            Ask MQ for all words in $\{\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) \cdot e : e \in E'\}$
        else
            $\boldsymbol{\Sigma'} = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}_{\text{new}}, \boldsymbol{a}'_{\text{new}}\}$
            $\psi' = \psi \cup \{\psi_{\boldsymbol{u}}\}$, with $\psi_{\boldsymbol{u}}(a) = \begin{cases} \boldsymbol{a}'_{\text{new}} & \text{if } a \geq b \\ \boldsymbol{a}_{\text{new}} & \text{otherwise} \end{cases}$
            $\boldsymbol{R'} = (\boldsymbol{R} - \{\boldsymbol{u}\}) \cup \{\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}, \boldsymbol{u} \cdot \boldsymbol{a}'_{\text{new}}\}$
            $E' = E \cup \{\text{suffixes of } b \cdot v\}$
            $\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) = \mu(\boldsymbol{u}) \cdot a_0$; $\mu(\boldsymbol{u} \cdot \boldsymbol{a}'_{\text{new}}) = \mu(\boldsymbol{u}) \cdot b$
            Ask MQ for all words in $\{(\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) \cup \mu(\boldsymbol{u} \cdot \boldsymbol{a}'_{\text{new}})) \cdot e : e \in E'\}$
        end if
        $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma'}, \boldsymbol{S'}, \boldsymbol{R'}, \psi', E', \boldsymbol{f'}, \mu')$
    end if
    if $\boldsymbol{T}$ is not closed then
        CLOSE
    end if
end procedure
```

$\boldsymbol{u} \in \boldsymbol{S} \cup \boldsymbol{R}$ but $s \cdot b \notin \mu(\boldsymbol{u'})$ for any word $\boldsymbol{u'}$ in the symbolic sample. We consider two cases, $\boldsymbol{u} \in \boldsymbol{S}$ and $\boldsymbol{u} \in \boldsymbol{R}$.

In the first case, when $\boldsymbol{u} \in \boldsymbol{S}$, $\boldsymbol{u}$ is already a state in the hypothesis but b indicates that the partition boundariues are not correctly defined and need refinement. That is, $u \cdot b$ was wrongly considered to be part of $[\boldsymbol{u} \cdot \boldsymbol{a}]$ for some $\boldsymbol{a} \in \boldsymbol{\Sigma_u}$, and thus b was wrongly considered to be part of $[\boldsymbol{a}]$. Due to minimality, all letters in $[\boldsymbol{a}]$ less than letter b behave like $\mu(\boldsymbol{a})$. We assume that all remaining letters in $[\boldsymbol{a}]$ behave like b and map them to a new symbol $\boldsymbol{a}_{\text{new}}$ that we add to $\boldsymbol{\Sigma}_{\boldsymbol{u}}$. We then update $\psi_{\boldsymbol{u}}$ such that $\psi'_{\boldsymbol{u}}(a) = \boldsymbol{a}_{\text{new}}$ for all $a \in [\boldsymbol{a}], a \geq b$, and $\psi'_{\boldsymbol{u}}(a) = \psi_{\boldsymbol{u}}(a)$, otherwise. The evidence function is updated by letting $\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) = \mu(\boldsymbol{u}) \cdot b$ and $\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}$ is added to $\boldsymbol{R}$.

In the second case, the symbolic word $\boldsymbol{u}$ is part of the boundary. From the counterexample we deduce that $\boldsymbol{u}$ is not equivalent to any of the existing states in the hypothesis and should form a new state. Specifically, we find the prefix $\boldsymbol{s}$ that was considered to be equivalent to $\boldsymbol{u}$, that is $g(\boldsymbol{u}) = \boldsymbol{s} \in \boldsymbol{S}$. Since the table is reduced $\boldsymbol{f_u} \neq \boldsymbol{f_{s'}}$ for any other $\boldsymbol{s'} \in \boldsymbol{S}$. Because w is the shortest counter-example, the classification of $\boldsymbol{s} \cdot b \cdot v$ in the automaton is correct (otherwise $s \cdot b \cdot v$, for some $s \in [\boldsymbol{s}]$ would constitute a shorter counter-example) and different from that of $u \cdot b \cdot v$. We conclude that $\boldsymbol{u}$ is a new state, which is added to $\boldsymbol{S}$. To distinguish between $\boldsymbol{u}$ and $\boldsymbol{s}$ we add to E the word $b \cdot v$, possibly with some of its suffixes (see [BR04] for a more detailed discussion of counter-example processing).

As $\boldsymbol{u}$ is a new state we need to add its continuations to R. We distinguish two subcases depending on b. If $b = a_0$, the smallest element of Σ, then a new symbolic letter $\boldsymbol{a}_{\text{new}}$ is added to $\boldsymbol{\Sigma}$, with $[\boldsymbol{a}_{\text{new}}] = \Sigma$ and $\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) = \mu(\boldsymbol{u}) \cdot a_0$, and the symbolic word $\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}$ is added to $\boldsymbol{R}$. If $b \neq a_0$ then *two* new symbolic letters, $\boldsymbol{a}_{\text{new}}$ and $\boldsymbol{a}'_{\text{new}}$, are added to $\boldsymbol{\Sigma}$ with $[\boldsymbol{a}_{\text{new}}] = \{a : a < b\}$, $[\boldsymbol{a}'_{\text{new}}] = \{a : a \geq b\}$, $\mu(\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}) = \mu(\boldsymbol{u}) \cdot a_0$ and $\mu(\boldsymbol{u} \cdot \boldsymbol{a}'_{\text{new}}) = \mu(\boldsymbol{u}) \cdot b$. The words $\boldsymbol{u} \cdot \boldsymbol{a}_{\text{new}}$ and $\boldsymbol{u} \cdot \boldsymbol{a}'_{\text{new}}$ are added to $\boldsymbol{R}$.

A detailed description of the algorithm is given in Algorithm 1 and its major procedures, table closing and counter-example treatment are described in Procedures 2 and 3 respectively. A statement of the form $\boldsymbol{\Sigma}' = \boldsymbol{\Sigma} \cup \{\boldsymbol{a}\}$ indicates the introduction of a new symbolic letter $\boldsymbol{a} \notin \boldsymbol{\Sigma}$. We use MQ and EQ as shorthands for membership and equivalence queries, respectively. In the following we illustrate the symbolic algorithm as applied to a language over an infinite alphabet.

Example 4.1. Let $\Sigma = [0, 100) \subset \mathbb{R}$ with the usual order and let $L \subseteq \Sigma^*$ be a target language. Fig. 5 shows the evolution of the symbolic observation tables and Fig. 6 depicts the corresponding automata and the concrete semantics of the symbolic alphabets.

We initialize the table with $\boldsymbol{S} = \{\boldsymbol{\epsilon}\}$, $\boldsymbol{R} = \{\boldsymbol{a}_0\}$, $\mu(\boldsymbol{a}_0) = \{0\}$ and $E = \{\epsilon\}$ and ask membership queries for ϵ (rejected) and 0 (accepted). The obtained table, $\boldsymbol{T}_0$ is not closed so we move $\boldsymbol{a}_0$ to $\boldsymbol{S}$, introduce $\boldsymbol{\Sigma}_{\boldsymbol{a}_0} = \{\boldsymbol{a}_1\}$, where $\boldsymbol{a}_1$ is a new symbol, and add $\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$ to R with $\mu(\boldsymbol{a}_0 \cdot \boldsymbol{a}_1) = 0 \cdot 0$. Asking membership queries we obtain the closed table $\boldsymbol{T}_1$ and its automaton $\boldsymbol{\mathcal{A}}_1$. We pose an equivalence query and obtain $(50, -)$ as a (minimal) counter-example which implies that all words smaller than 50 are correctly classified. We add a new symbol $\boldsymbol{a}_2$ to $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$ and redefine the concrete semantics to $[\boldsymbol{a}_0] = \{a < 50\}$ and $[\boldsymbol{a}_2] = \{a \geq 50\}$. As evidence we select the smallest possible letter, $\mu(\boldsymbol{a}_2) = 50$, ask membership queries to obtain the closed table $\boldsymbol{T}_2$ and automaton $\boldsymbol{\mathcal{A}}_2$.

For this hypothesis we get a counter-example $(0 \cdot 30, -)$ whose prefix 0 is already in the sample, hence the misclassification occurs in the second transition. We refine the alphabet partition for state $\boldsymbol{a}_0$ by introducing a new symbol $\boldsymbol{a}_3$ and letting $[\boldsymbol{a}_1] = \{a < 30\}$ and $[\boldsymbol{a}_3] = \{a \geq 30\}$. Table $\boldsymbol{T}_3$ is closed but automaton $\boldsymbol{\mathcal{A}}_3$ is still incorrect and a counter-example $(50 \cdot 0, -)$ is provided. The prefix 50 belongs to the evidence of $\boldsymbol{a}_2$ and is moved from the boundary to become a new state and its successor $\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$, for a new symbol $\boldsymbol{a}_4$, is added to $\boldsymbol{R}$. To distinguish $\boldsymbol{a}_2$ from $\boldsymbol{\epsilon}$, the suffix 0 of the counter-example is added to E resulting in $\boldsymbol{T}_4$ which is not closed. The newly discovered state $\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$ is added to $\boldsymbol{S}$, the filled table $\boldsymbol{T}_5$ is closed and the conjectured automaton $\boldsymbol{A}_5$ has two additional states.

$\boldsymbol{T}_0$

	ϵ
ϵ	-
$\boldsymbol{a}_0$	+

$\boldsymbol{T}_1$

	ϵ
ϵ	-
$\boldsymbol{a}_0$	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+

$\boldsymbol{T}_2$

	ϵ
ϵ	-
$\boldsymbol{a}_0$	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+
$\boldsymbol{a}_2$	-

$\boldsymbol{T}_3$

	ϵ
ϵ	-
$\boldsymbol{a}_0$	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+
$\boldsymbol{a}_2$	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-

$\boldsymbol{T}_4$

	ϵ	0
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+

$\boldsymbol{T}_5$

	ϵ	0
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-

$\boldsymbol{T}_6$

	ϵ	0
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_6$	+	-

$\boldsymbol{T}_7$

	ϵ	0
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_6$	+	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_7$	-	-

$\boldsymbol{T}_8$

	ϵ	0
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_6$	+	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_7$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_8$	+	+

FIGURE 5. Observation tables for Example 4.1.

Subsequent equivalence queries result counter-examples $(50 \cdot 20, +)$, $(50 \cdot 80, -)$ and $(50 \cdot 50 \cdot 0, +)$ which are used to refine the alphabet partition at state $\boldsymbol{a}_2$ and modify its outgoing transitions progressively as seen in automata $\boldsymbol{\mathcal{A}}_6$, $\boldsymbol{\mathcal{A}}_7$ and $\boldsymbol{\mathcal{A}}_8$, respectively. Automaton $\boldsymbol{\mathcal{A}}_8$ accepts the target language and the algorithm terminates. □

Note that for the language in Example 1.3, the symbolic algorithm needs around 30 queries instead of the 80 queries required by L^*. If we choose to learn a language as the one described in Example 4.1, restricting the concrete alphabet to the finite alphabet $\Sigma = \{1, \ldots, 100\}$, then L^* requires around 1000 queries compared to 17 queries required by our symbolic algorithm. As we shall see in Section 6, the complexity of the symbolic algorithm does not depend on the size of the concrete alphabet, only on the number of transitions.

5. Learning Languages over Partially-ordered Alphabets

In this section we sketch the extension of the results of this paper to partially-ordered alphabets of the form $\Sigma = X^d$ where X is a totally-ordered set such as an interval $[0, k) \subseteq \mathbb{R}$. Letters of Σ are d-tuples of the form $\mathbf{x} = (x_1, \ldots, x_d)$ and the minimal element is $\mathbf{0} = (0, \ldots, 0)$. The usual partial order on this set is defined as $\mathbf{x} \leq \mathbf{y}$ if and only if $x_i \leq y_i$ for all $i = 1, \ldots, d$. When $\mathbf{x} \leq \mathbf{y}$ and $x_i \neq y_i$ for some i the inequality is strict, denoted by $\mathbf{x} < \mathbf{y}$, and we say then that $\mathbf{x}$ *dominates* $\mathbf{y}$. Two elements are *incomparable*, denoted by $\mathbf{x} || \mathbf{y}$, if $x_i < y_i$ and $x_j > y_j$ for some i and j.

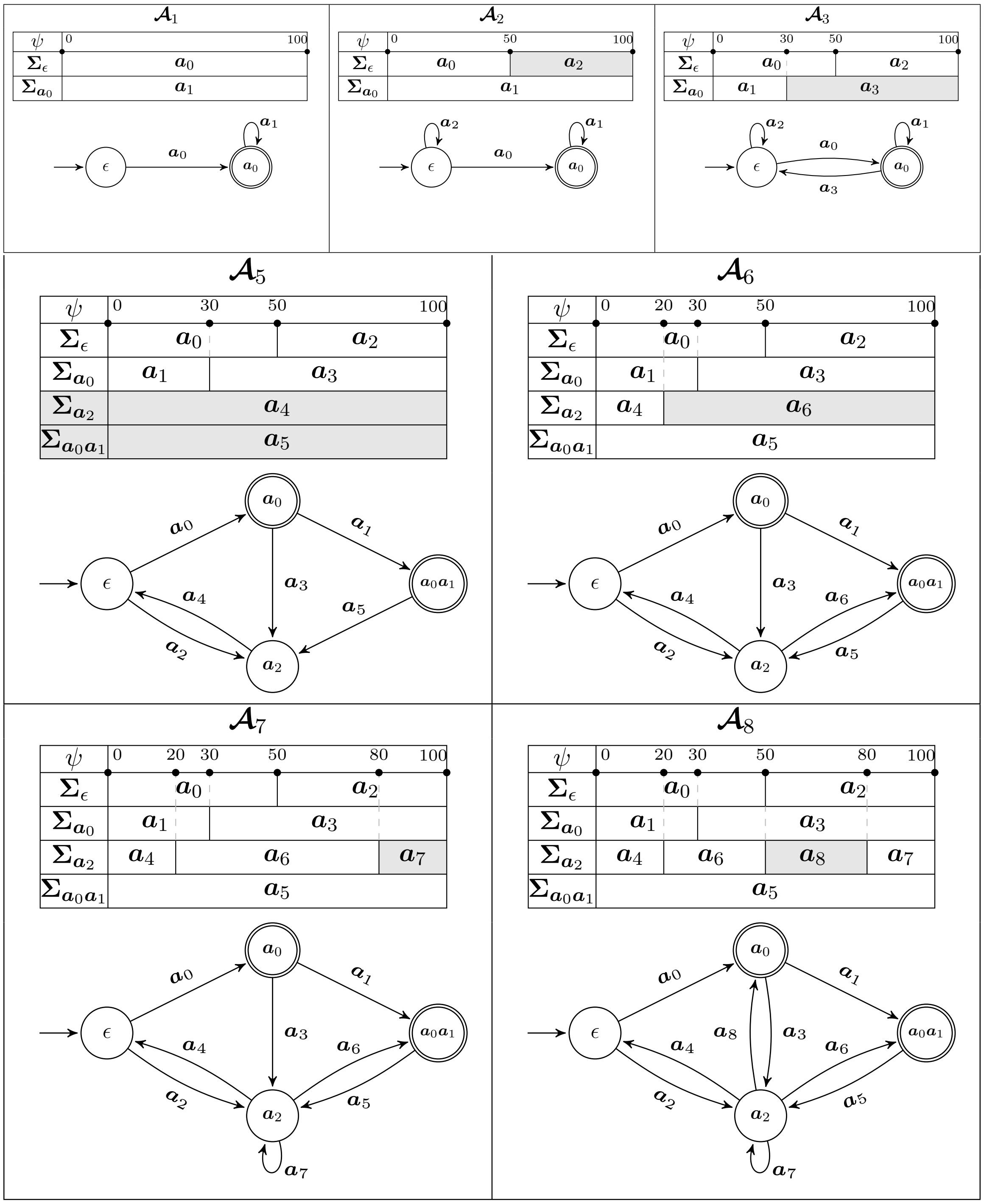
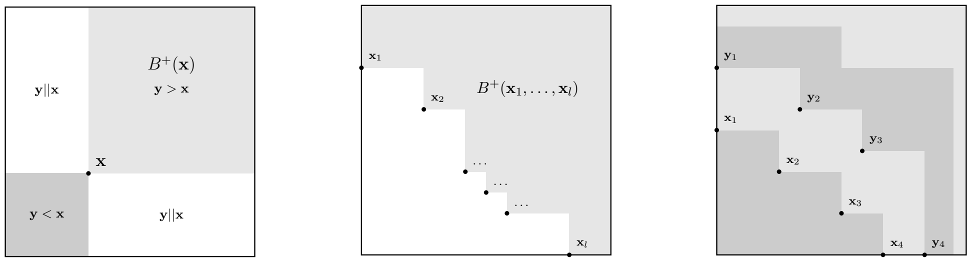

$\mathcal{A}_1$
ψ 0 100
Σ_ϵ a_0
Σ_{a_0} a_1
$\mathcal{A}_2$
ψ 0 50 100
Σ_ϵ a_0 a_2
Σ_{a_0} a_1
$\mathcal{A}_3$
ψ 0 30 50 100
Σ_ϵ a_0 a_2
Σ_{a_0} a_1 a_3
$\mathcal{A}_5$
ψ 0 30 50 100
Σ_ϵ a_0 a_2
Σ_{a_0} a_1 a_3
Σ_{a_2} a_4
$\Sigma_{a_0a_1}$ a_5
$\mathcal{A}_6$
ψ 0 20 30 50 100
Σ_ϵ a_0 a_2
Σ_{a_0} a_1 a_3
Σ_{a_2} a_4 a_6
$\Sigma_{a_0a_1}$ a_5
$\mathcal{A}_7$
ψ 0 20 30 50 80 100
Σ_ϵ a_0 a_2
Σ_{a_0} a_1 a_3
Σ_{a_2} a_4 a_6 a_7
$\Sigma_{a_0a_1}$ a_5
$\mathcal{A}_8$
ψ 0 20 30 50 80 100
Σ_ϵ a_0 a_2
Σ_{a_0} a_1 a_3
Σ_{a_2} a_4 a_6 a_8 a_7
$\Sigma_{a_0a_1}$ a_5
ϵ a_0 a_2 a_0a_1
a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8

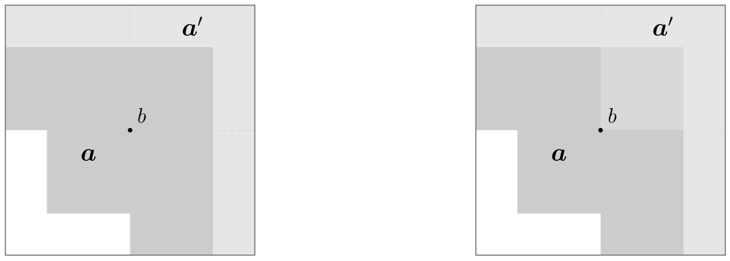
FIGURE 6. Hypotheses and Σ-semantics for Example 4.1

For partially-ordered sets, a natural extension of the partition of an ordered set into intervals is a *monotone* partition, where for each partition block P there are no three points



(A) Backward and forward cone for $\mathbf{x}$ (B) Union of cones (C) An alphabet partition

FIGURE 7



(A) (B)

FIGURE 8. Modifying the alphabet partition for state $\boldsymbol{u}$ after receiving $u \cdot b \cdot v$ as counter-example. Letters above b are moved from $[\boldsymbol{a}]$ to $[\boldsymbol{a}']$.

such that $\mathbf{x} < \mathbf{y} < \mathbf{z}$, $\mathbf{x}, \mathbf{z} \in P$, and $\mathbf{y} \notin P$. We define in the following such partitions represented by a finite set of points.

A *forward cone* $B^+(\mathbf{x}) \subset \Sigma$ is the set of all points dominated by a point $\mathbf{x} \in \Sigma$ (see Fig. 7a). Let $F = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$ be a set of points, then $B^+(F) = B^+(\mathbf{x}_1) \cup \ldots \cup B^+(\mathbf{x}_l)$ as shown in Fig. 7b. From a family of sets of points $\mathcal{F} = \{F_0, \ldots, F_{m-1}\}$, such that $F_0 = \{\mathbf{0}\}$ satisfying for every i: 1) $\forall \mathbf{y} \in F_i$, $\exists \mathbf{x} \in F_{i-1}$ such that $\mathbf{x} < \mathbf{y}$, and 2) $\forall \mathbf{y} \in F_i$, $\forall \mathbf{x} \in F_{i-1}$, $\mathbf{y} \not< \mathbf{x}$, we can define a monotone partition of the form $\mathcal{P} = \{P_1, \ldots, P_{m-1}\}$, where $P_i = B^+(F_{i-1}) - B^+(F_i)$, see Fig. 7c.

A subset P of Σ, as defined above, may have several mutually-incomparable minimal elements, none of which being dominated by any other element of P. One can thus apply the symbolic learning algorithm but without the presence of unique minimal evidence and minimal counter-example. For this reason a symbolic word may have more than one evidence. Evidence compatibility is preserved though due to the nature of the partition.

The teacher is assumed to return a counter-example chosen from a set of incomparable minimal counter-examples. Like in the algorithm for totally ordered alphabet, every counter-example either discovers a new state or refines a partition. The learning algorithm for partially-ordered alphabets is similar to Algorithm 1 and can be applied with only a minor modification in the treatment of the counterexamples and specifically in the refinement procedure. Lines 6-8 of Procedure 3 should be ignored in the case where there exists a symbolic letter $\boldsymbol{a}'$, as illustrated in Fig. 8a, such that $\boldsymbol{f}(\boldsymbol{u} \cdot b \cdot e) = \boldsymbol{f}(\boldsymbol{u} \cdot \boldsymbol{a}' \cdot e)$ for all $e \in E$. In such a case, function ψ is updated as in line 9 by replacing $\boldsymbol{a}_{\text{new}}$ by $\boldsymbol{a}'$ and b

should be added to $\mu(\boldsymbol{a}')$. In Fig. 8b, one can see the partition after refinement, where all letters above b have been moved from $[\boldsymbol{a}]$ to $[\boldsymbol{a}']$.

$\boldsymbol{T}_0$

	ϵ
ϵ	-
$\boldsymbol{a}_0$	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+

$\boldsymbol{T}_{1-3}$

	ϵ
ϵ	-
$\boldsymbol{a}_0$	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+
$\boldsymbol{a}_2$	-

$\boldsymbol{T}_{4-7}$

	ϵ
ϵ	-
$\boldsymbol{a}_0$	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+
$\boldsymbol{a}_2$	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-

$\boldsymbol{T}_8$

	ϵ	$\binom{0}{0}$
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+

$\boldsymbol{T}_9$

	ϵ	$\binom{0}{0}$
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-

$\boldsymbol{T}_{10-11}$

	ϵ	$\binom{0}{0}$
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_6$	+	-

$\boldsymbol{T}_{12-15}$

	ϵ	$\binom{0}{0}$
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_6$	+	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_7$	-	-

$\boldsymbol{T}_{16-18}$

	ϵ	$\binom{0}{0}$
ϵ	-	+
$\boldsymbol{a}_0$	+	+
$\boldsymbol{a}_2$	-	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1$	+	-
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_3$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_4$	-	+
$\boldsymbol{a}_0 \cdot \boldsymbol{a}_1 \cdot \boldsymbol{a}_5$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_6$	+	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_7$	-	-
$\boldsymbol{a}_2 \cdot \boldsymbol{a}_8$	+	+

FIGURE 9. Observation tables for Example 5.1

Example 5.1. Let us illustrate the learning process for a target language L defined over $\Sigma = [0, 100]^2$. All tables, hypotheses automata and alphabet partitions for this example are shown in Figures 9, 10, and 11, respectively.

The learner starts asking MQs for the empty word. A symbolic letter $\boldsymbol{a}_0$ is chosen to represent its continuations with the minimal element of Σ as evidence, i.e., $\mu(\boldsymbol{a}_0) = \binom{0}{0}$. The symbolic word $\boldsymbol{a}_0$ is moved to $\boldsymbol{S}$ for the table $\boldsymbol{T}_0$ to be closed. The symbolic letter $\boldsymbol{a}_1$ is added to the alphabet of state $\boldsymbol{a}_0$, and the learner asks a MQ for $\binom{0}{0}\binom{0}{0}$, the evidence of the symbolic word $\boldsymbol{a}_0\boldsymbol{a}_1$. The first hypothesis automaton is $\boldsymbol{\mathcal{A}}_0$ with Σ-semantics $[\boldsymbol{a}_0] = [\boldsymbol{a}_1] = \Sigma$. The counter-example $(\binom{45}{50}, -)$ refines the partition for the initial state. The symbolic alphabet is extended to $\boldsymbol{\Sigma}_\epsilon = \{\boldsymbol{a}_0, \boldsymbol{a}_2\}$ with $[\boldsymbol{a}_2] = \{x >= \binom{45}{50}\}$, $[\boldsymbol{a}_0] = \Sigma - [\boldsymbol{a}_2]$, and $\mu(\boldsymbol{a}_2) = \binom{45}{50}$. The new observation table and hypothesis are $\boldsymbol{T}_1$ and $\boldsymbol{\mathcal{A}}_1$. Two more counter-examples will come to refine the partition for the initial state, $(\binom{60}{0}, -)$ and $(\binom{0}{70}, -)$, that will modify the partition for the initial state, moving all letters greater than $\binom{60}{0}$ and $\binom{0}{70}$ to the Σ-semantics of $\boldsymbol{a}_2$ as can be seen in ψ_2 and ψ_3 respectively.

After the hypothesis $\boldsymbol{\mathcal{A}}_3$, the counter-example $(\binom{0}{0}\binom{0}{80}, -)$ adds a new symbol $\boldsymbol{a}_3$ and a new transition in the hypothesis automaton. The counter-examples that follow, namely, $(\binom{0}{0}\binom{80}{0}, -)$, $(\binom{0}{0}\binom{40}{15}, -)$, and $(\binom{0}{0}\binom{30}{30}, -)$ refine the Σ-semantics for symbols in $\boldsymbol{\Sigma}_{\boldsymbol{a}_0}$ as shown in ψ_{4-7}.

Then counter-example $(\binom{45}{50}\binom{0}{0}, +)$ is presented. As we can see, the prefix $\binom{45}{50}$ exist already in $\mu(\boldsymbol{a}_2)$ and $\boldsymbol{a}_2 \in \boldsymbol{R}$ which means $\boldsymbol{a}_2$ becomes a state, and to distinguish it from the

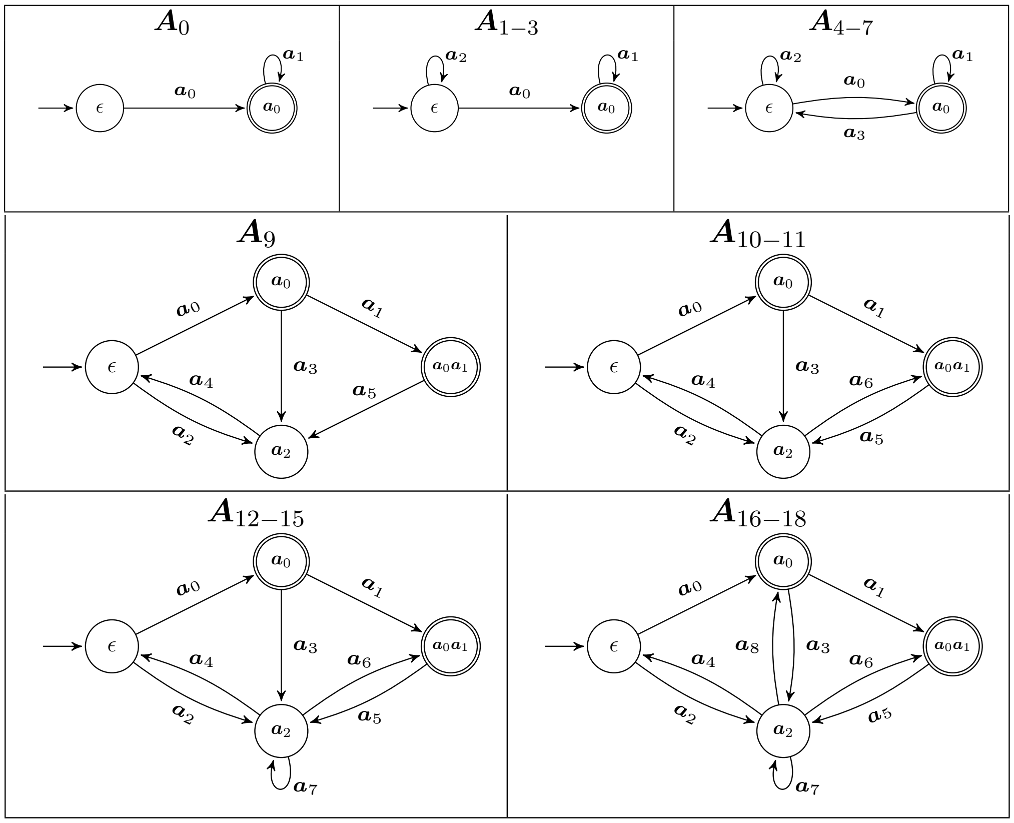


FIGURE 10. Hypothesis automata for Example 5.1

state represented by the empty word the learner adds to E the suffix of the counter-example $\binom{0}{0}$. The resulting table $\boldsymbol{T}_8$ is not closed and $\boldsymbol{a}_0\boldsymbol{a}_1$ is moved to $\boldsymbol{S}$. The new table $\boldsymbol{T}_9$ is closed and evidence compatible. The hypothesis $\boldsymbol{\mathcal{A}}_9$ has now four states and the symbolic alphabet and Σ-semantics for each state can be seen in ψ_9. The counter-examples that follow will refine the partition at state $\boldsymbol{a}_2$. The new transitions discovered and all refinements are shown in $\boldsymbol{\mathcal{A}}_{10-18}$ and $\psi_{10} - \psi_{18}$. The language was learned using 20 membership queries and 17 counter-examples. □

6. On Complexity

The complexity of the symbolic algorithm is influenced not by the size of the alphabet but by the resolution (partition size) with which we observe it. Let $L \subset \Sigma$ be the target language and let $\boldsymbol{\mathcal{A}}$ be the minimal symbolic automaton recognizing this language with state set Q of size n and a symbolic alphabet $\boldsymbol{\Sigma} = \biguplus_q \boldsymbol{\Sigma}_q$ such that $|\boldsymbol{\Sigma}_q| \leq m$ for every q.

Each counter-example improves the hypothesis in one out of two ways. Either a new state is discovered or a partition gets refined. Hence, at most $n-1$ equivalence queries of the first type can be asked and $n(m-1)$ of the second, resulting in $\mathcal{O}(mn)$ equivalence queries.

Concerning the size of the table, the set of prefixes $\boldsymbol{S}$ is monotonically increasing and reaches the size of exactly n elements. Since the table, by construction, is always kept

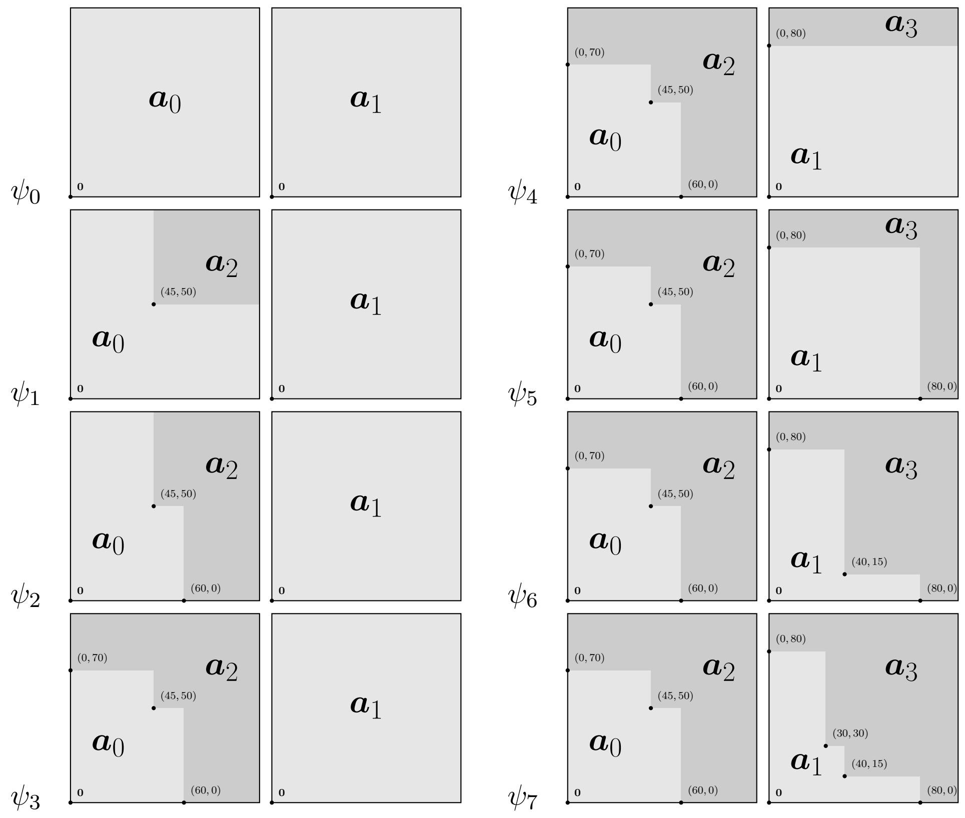


Figure 11. Alphabet partition for Example 5.1 (part 1)

reduced, the elements in $\boldsymbol{S}$ represent exactly the states of the automaton. The size of the boundary is always smaller than the total number of transitions in the automaton, that is $mn - n + 1$. The number of suffixes in E, playing a distinguishing role for the states of the automaton, range between $\log_2 n$ and n. Hence, the size of the table ranges between $(n + m)\log_2 n$ and $n(mn + 1)$.

For a totally ordered alphabet the size of the concrete sample coincides with the size of the symbolic sample associated with the table and hence the number of membership queries asked is $\mathcal{O}(mn^2)$. For a partially ordered alphabet with each F_i defined by at most l points, some additional queries are asked. For every row in $\boldsymbol{S}$, at most $n(m-1)(l-1)$ additional words are added to the concrete sample, hence more membership queries might need to be asked. Furthermore, at most $l - 1$ more counter-examples are given to refine a partition. To conclude, the number of queries in total asked to learn language L is $\mathcal{O}(mn^2)$ if $l < n$ and $\mathcal{O}(lmn)$ otherwise.

7. Conclusion

We have defined a generic algorithmic scheme for automaton learning, targeting languages over large alphabets that can be recognized by finite symbolic automata having a modest number of states and transitions. Some ideas similar to ours have been proposed for the particular case of parametric languages [BJR06] and recently in a more general setting

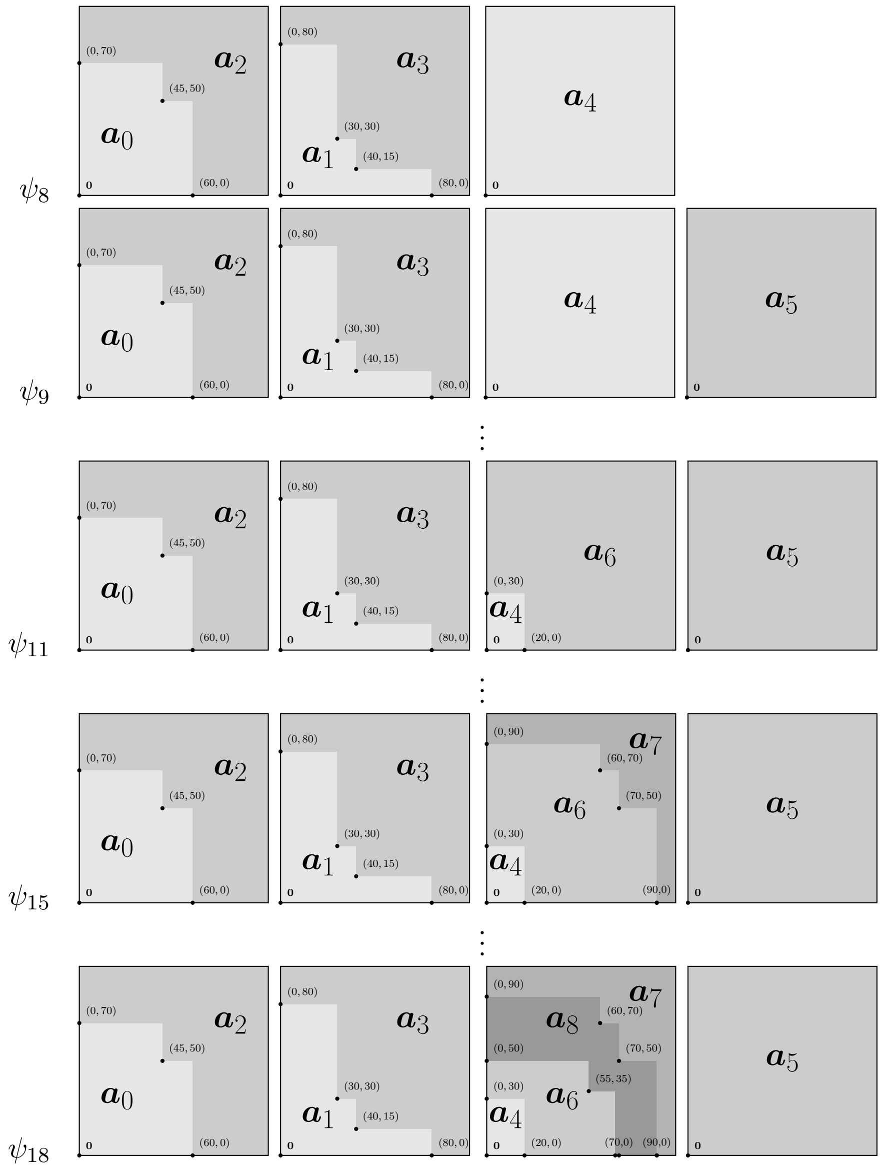


FIGURE 11. Alphabet partition for Example 5.1 (part 2)

[HSM11, IHS13, BB13] including partial evidential support and alphabet refinement during the learning process.

The genericity of our algorithm is due to a semantic approach (alphabet partitions) but of course, each and every domain will have its own semantic and syntactic specialization in terms of the size and shape of the alphabet partitions. In this work we have implemented an instantiation of this scheme for alphabets such as $(\mathbb{N}, \leq)$ and $(\mathbb{R}, \leq)$. When dealing with numbers, the partition into a finite number of intervals (and monotone sets in higher dimensions) is very natural and used in many application domains ranging from quantization of sensor readings to income tax regulations. It will be interesting to compare the expressive power and succinctness of symbolic automata with other approaches for representing numerical time series and to compare our algorithm with other inductive inference techniques for sequences of numbers.

As a first excursion into the domain, we have made quite strong assumptions on the nature of the equivalence oracle, which, already for small alphabets, is a bit too strong and pedagogical to be realistic. We assumed that it provides the shortest counter-example and also that it chooses always the minimal available concrete symbol. We can relax the latter (or both) and even omit this oracle altogether and replace it by random sampling, as already proposed in [Ang87] for concrete learning. Over large alphabets, it might be even more appropriate to employ probabilistic convergence criteria a-la *PAC learning* [Val84] and be content with a correct classification of a large fraction of the words, thus tolerating imprecise tracing of boundaries in the alphabet partitions. This topic is subject to ongoing work. Another challenging research direction is the adaptation of our framework to languages over Boolean vectors.

Acknowledgement

This work was supported by the French project EQINOCS (ANR-11-BS02-004). We thank Peter Habermehl, Eugene Asarin and anonymous referees for useful comments and pointers to the literature.

References

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[BB13] Matko Botinčan and Domagoj Babić. Sigma*: Symbolic learning of Input-Output specifications. In *POPL*, pages 443–456. ACM, 2013.

[BJR06] Therese Berg, Bengt Jonsson, and Harald Raffelt. Regular inference for state machines with parameters. In *FASE*, volume 3922 of *LNCS*, pages 107–121. Springer, 2006.

[BLP10] Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In *AMW*, volume 619 of *CEUR Workshop Proceedings*, 2010.

[BR04] Therese Berg and Harald Raffelt. Model checking. In *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 557–603. Springer, 2004.

[DlH10] Colin De la Higuera. *Grammatical inference: learning automata and grammars*. Cambridge University Press, 2010.

[DR95] Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific, 1995.

[DV14] Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *POPL*, pages 541–554. ACM, 2014.

[Gol72] E. Mark Gold. System identification via state characterization. *Automatica*, 8(5):621–636, 1972.

[HJJ$^+$95] Jesper G. Henriksen, Ole J.L. Jensen, Michael E. Jrgensen, Nils Klarlund, Robert Paige, Theis Rauhe, and Anders B. Sandholm. Mona: Monadic second-order logic in practice. In *TACAS*, volume 1019 of *LNCS*, pages 80–110. Springer, 1995.

[HSJC12] Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In *VMCAI*, volume 7148 of *LNCS*, pages 251–266. Springer, 2012.

[HSM11] Falk Howar, Bernhard Steffen, and Maik Merten. Automata learning with automated alphabet abstraction refinement. In *VMCAI*, volume 6538 of *LNCS*, pages 263–277. Springer, 2011.

[HV11] Pieter Hooimeijer and Margus Veanes. An evaluation of automata algorithms for string analysis. In *VMCAI*, volume 6538 of *LNCS*, pages 248–262. Springer, 2011.

[IHS13] Malte Isberner, Falk Howar, and Bernhard Steffen. Inferring automata with state-local alphabet abstractions. In *NASA Formal Methods*, volume 7871 of *LNCS*, pages 124–138. Springer, 2013.

[KF94] Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

[MM14] Oded Maler and Irini-Eleftheria Mens. Learning regular languages over large alphabets. In *TACAS*, volume 8413 of *LNCS*, pages 485–499. Springer, 2014.

[Moo56] Edward F Moore. Gedanken-experiments on sequential machines. In *Automata studies*, volume 34 of *Annals of Mathematical Studies*, pages 129–153. Princeton, 1956.

[MP95] Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995.

[Ner58] Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[Val84] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[VB12] Margus Veanes and Nikolaj Bjørner. Symbolic automata: The toolkit. In *TACAS*, volume 7214 of *LNCS*, pages 472–477. Springer, 2012.

[VHL$^+$12] Margus Veanes, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Björner. Symbolic finite state transducers: algorithms and applications. In *POPL*, pages 137–150. ACM, 2012.