A Generic Algorithm for Learning Symbolic Automata from Membership Queries

Oded Maler and Irini-Eleftheria Mens

VERIMAG
CNRS and University of Grenoble-Alpes, France

Abstract. We present a generic algorithmic scheme for learning languages defined over large or infinite alphabets such as bounded subsets of $\mathbb{N}$ and $\mathbb{R}$, or Boolean vectors of high dimension. These languages are accepted by deterministic *symbolic automata* that use predicates to label transitions, forming a finite partition of the alphabet for every state. Our learning algorithm, an adaptation of Angluin's L^*, combines standard automaton learning by state characterization, with the learning of the static predicates that define the alphabet partitions. We do not assume a helpful teacher who provides minimal counter-examples when the conjectured automaton is incorrect. Instead we use random sampling to obtain PAC (probably approximately correct) learnability. We have implemented the algorithm for numerical and Boolean alphabets and the preliminary performance results show that languages over large or infinite alphabets can be learned under more realistic assumptions.
Keywords: Symbolic automata. Automata learning. Infinite alphabets.

1 Introduction

The (classical) theory of regular languages and automata [14,19,27] deals mainly with alphabets that are small and "flat", that is, sets without any additional structure. In many applications, however, alphabets are large and structured. In hardware verification, for example, behaviors are sequences of states and inputs ranging over valuations of Boolean state variables that give rise to exponentially large alphabets, treated symbolically using BDDs and other logical formalisms. As another motivation, consider the verification of continuous and hybrid systems against specifications written in formalisms such as *signal temporal logic* (STL) [20,21]. Automata over *numerical alphabets*, admitting an order or partial-order relation, can define the semantics of such requirements.

In recent years, *symbolic automata* [31] have been studied extensively as a generic framework for recognizing regular languages over large alphabets. In such automata the number of states and transitions is typically small and the transitions are labeled by predicates from a corresponding theory, denoting subsets of the alphabet. The extension of classical results of automata theory to symbolic automata has become an active area of research, including fundamental issues such as minimization [10] or effective closure under various operations [30,32,33] as well as the adaptation of learning algorithms [6,17,23]. The comparison with other related work is postponed to the conclusion section, after the technical issues are explained in the body of the paper.

In [23] Angluin's L^* algorithm [1] for learning automata from queries and counter-examples has been extended to learn languages over an alphabet Σ which is a high-cardinality bounded subset of $\mathbb{N}$ or $\mathbb{R}$. Such languages are represented by symbolic automata where transitions are labeled by symbolic letters, such that the concrete semantics $[\![\boldsymbol{a}]\!]$ of a symbolic letter $\boldsymbol{a}$ is a sub-interval of Σ. Determinism is maintained by letting the semantics at each state form a partition of Σ. The learning algorithm uses symbolic observation tables with symbolic words in the rows to provide access sequences $\boldsymbol{s}$ to discovered states. To determine the transitions outgoing from state $\boldsymbol{s}$, one needs, in principle, to ask membership queries concerning $s \cdot a \cdot e$ for every $s \in [\![\boldsymbol{s}]\!]$, $a \in \Sigma$ and $e \in E$ where E is a set of distinguishing suffixes. To avoid a large or infinite number of queries, the characterization of states is based on partial information: a small set $\mu(\boldsymbol{a})$ of concrete letters, called the *evidence* for $\boldsymbol{a}$, is associated with every symbol $\boldsymbol{a}$. This notion is lifted to symbolic words and membership queries are asked only for the words in $\mu(\boldsymbol{s}) \cdot \mu(\boldsymbol{a}) \cdot E$.

In this framework, the learning procedure is decomposed into a vertical/temporal component, consisting of discovering new states as in the original L^* algorithm, and a horizontal/spatial component where the boundaries of the alphabet partitions in every state are learned and modified. The advantage of this decomposition is that the first part is generic, invariant under alphabet change, while the second part has some alphabet specific features. To take a concrete example, alphabet partitions over a totally-ordered alphabet such as $\mathbb{R}$ are made using intervals with endpoints that can be shifted as new evidence accumulates. On the other hand, for an alphabet like $\mathbb{B}^n$, the partitions are represented by decision trees which are modified by restructuring.

A major weakness of [23], partly inherited from L^*, is that in addition to membership queries, it also uses *equivalence* queries: each time an automaton is conjectured, an oracle EQ either confirms the conjecture or provides a counter-example which is also minimal in the lexicographic order on Σ^*. Counter-examples of this type facilitate significantly the discovery of new states and the detection of the alphabet partition boundaries but such a helpful teacher is unrealistic in most real-life situations. In this paper we develop an algorithm that uses only membership queries. Some of those are used to fill the observation table in order to characterize states, while others, posed for randomly selected words, are used to test conjectures. Consequently, we have to replace certain and exact learnability by a weaker notion, in the spirit of Valiant's PAC (probably approximately correct) learning [29]: the algorithm converges with high probability to a language close to the target language.

Adapting the algorithm to this more challenging and less pedagogical setting of random sampling involves several modifications relative to [23]. First, when a new state q is discovered, we sample the alphabet at several points rather than only at the minimal elements. The sampled letters are used in queries to characterize the successors of q. In order to avoid an exponential growth in the number of membership queries, we refine the notion of evidence for symbolic words and ask queries only for the successors of *representative* evidences. Secondly, to determine whether a given counter-example leads to the discovery of a new state, to the introduction of a new transition or just to a modification of the partition boundaries associated with some existing transitions, we use an extended version of the breakpoint method of [26] which also appears in [16] in a similar form. This method identifies an erroneous position in the counter-example, classifies its nature and reacts accordingly.

We thus obtain an efficient algorithm for learning languages over large alphabets under much more realistic assumptions concerning the information available to the learner. We use one-dimensional numerical alphabets, partitioned into a bounded number of intervals, to illustrate the principles of the algorithm. The algorithmic scheme can be easily adapted to other domains provided that alphabet partitions are not too complex and consist of a small number of simple blocks. We demonstrate this fact by adapting the algorithm to Boolean alphabets partitioned into finitely many unions of sub-cubes.

The rest of the paper is organized as follows. In Sections 2 and 3 we define, respectively, symbolic automata and symbolic observation tables while explaining their role in learning. The symbolic learning algorithm is described in detail in Section 4 for one-dimensional numerical domains, followed by its extension to Boolean alphabets in Section 5. Section 6 provide some theoretical and empirical evaluation of the algorithm performance. In Section 7 we summarize our results, compare with other work on learning over large alphabets and suggest directions for future work.

The paper assumes some familiarity with automaton learning and the reader is invited to read more detailed explanations of concrete learning algorithms in [5] and of the framework underlying this paper in [23]. Likewise, some basic acquaintance with decision trees is assumed that can be obtained by consulting [8].

2 Preliminaries

Let Σ be an alphabet, and let Σ^* be the set of all finite sequences (words) over Σ. With a language $L \subseteq \Sigma^*$ we associate a *characteristic function* $f : \Sigma^* \rightarrow \{+,-\}$, where $f(w) = +$ if $w \in L$ and $f(w) = -$, otherwise. With every $s \in \Sigma^*$ we associate a *residual characteristic function* defined as $f_s(w) = f(s \cdot w)$. Two sequences s and r are Nerode equivalent [24] with respect to L, denoted by $s \sim_L r$, if $f_s = f_r$. The relation $\sim_L$ is a right congruence satisfying $s \sim_L r \rightarrow s \cdot a \sim_L r \cdot a$ and its equivalence classes correspond to the states of the minimal automaton that accepts L. The identification of these classes underlies minimization procedures as well as most automaton learning algorithms since [13].

A *symbolic automaton* over a *concrete alphabet* Σ is an automaton whose transitions are labeled by *symbolic letters* or *symbols*, taken from a *symbolic alphabet* $\boldsymbol{\Sigma}$, that denote subsets of Σ. We assume $\boldsymbol{\Sigma}$ to be a disjoint union of finite alphabets of the form $\boldsymbol{\Sigma}_q$, each associated with a state of the automaton. Concrete letters are mapped to symbols through a mapping $\psi : \Sigma \rightarrow \boldsymbol{\Sigma}$, decomposable into state-specific mappings $\psi_q : \Sigma \rightarrow \boldsymbol{\Sigma}_q$. The *$\Sigma$-semantics* of a symbol $\boldsymbol{a} \in \boldsymbol{\Sigma}_q$ is the inverse of ψ_q, that is, $[\![\boldsymbol{a}]\!] = \{a \in \Sigma : \psi_q(a) = \boldsymbol{a}\}$. The Σ-semantics is extended to symbolic words of the form $\boldsymbol{w} = \boldsymbol{a}_1 \cdots \boldsymbol{a}_{|\boldsymbol{w}|} \in \boldsymbol{\Sigma}^*$ as the concatenation of the concrete one-letter languages associated with the respective symbolic letters or, recursively speaking, by letting $[\![\boldsymbol{\varepsilon}]\!] = \{\varepsilon\}$ and $[\![\boldsymbol{w} \cdot \boldsymbol{a}]\!] = [\![\boldsymbol{w}]\!] \cdot [\![\boldsymbol{a}]\!]$ for $\boldsymbol{w} \in \boldsymbol{\Sigma}^*$, $\boldsymbol{a} \in \boldsymbol{\Sigma}$.

A symbolic automaton is *complete* and *deterministic* over Σ when for each state q the set $\{[\![\boldsymbol{a}]\!] : \boldsymbol{a} \in \boldsymbol{\Sigma}_q\}$ forms a partition of Σ. For this, we always let ψ_q be a total function. Moreover, by letting ψ to be surjective we avoid symbols with empty semantics. We often omit ψ and ψ_q from the notation and use $[\![\boldsymbol{a}]\!]$ when ψ, which is always present, is clear from the context.

Definition 1 (Symbolic Automaton). *A* deterministic symbolic automaton *is a tuple $\boldsymbol{\mathcal{A}} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \boldsymbol{\delta}, q_0, F)$, 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 total surjective functions $\psi_q : \Sigma \to \boldsymbol{\Sigma}_q$, Q is a finite set of states, q_0 is the initial state and F is the set of accepting states, $\boldsymbol{\delta} : Q \times \boldsymbol{\Sigma} \to Q$ is a partial transition function decomposable into a family of total functions $\boldsymbol{\delta}_q : \{q\} \times \boldsymbol{\Sigma}_q \to Q$.*

The transition function is extended to words as in the concrete case. The symbolic automaton can be viewed as an acceptor of a concrete language, that is, when at q and reading a concrete letter a, the automaton takes the transition $\boldsymbol{\delta}(q, \psi(a))$. Hence, the language $L(\boldsymbol{\mathcal{A}})$ consists of all concrete words whose run leads from q_0 to a state in F. A language $L \subseteq \Sigma^*$ is *symbolic recognizable* if there exists a symbolic automaton $\boldsymbol{\mathcal{A}}$ such that $L = L(\boldsymbol{\mathcal{A}})$.

3 Symbolic Observation Tables

The present algorithm relaxes the strong assumption of a helpful teacher [1,23]. Such a teacher responds positively to an equivalence query EQ$(\boldsymbol{\mathcal{A}})$, where $\boldsymbol{\mathcal{A}}$ is an automaton conjectured by the learning algorithm, only if $L(\boldsymbol{\mathcal{A}})$ is indeed equivalent to the target language; otherwise, it returns a minimal counter-example which helps the learner to localize the modification site. In the new relaxed setting, equivalence queries are approximated by *testing queries*: a call to EQ yields membership queries for a set of randomly selected words; when all of them agree with the hypothesis, the algorithm terminates with a non-zero probability of misclassification; otherwise, we have a counter-example to process. The number of such queries may depend on what we assume about the distribution over Σ^* and what we want to prove about the algorithm, for example PAC learnability as in [1], which is further discussed in Section 6.

Counter-examples obtained via testing queries need not be minimal neither in length nor lexicographically and hence partition boundaries are determined with some possible approximation error. Unlike [23], the present algorithm requires the use of multiple evidences for each symbol. To avoid an undesirable growth in the number of queries one of the evidences is chosen as a *representative* and certain queries during subsequent stages of the learning process are restricted to words with a representative prefix.

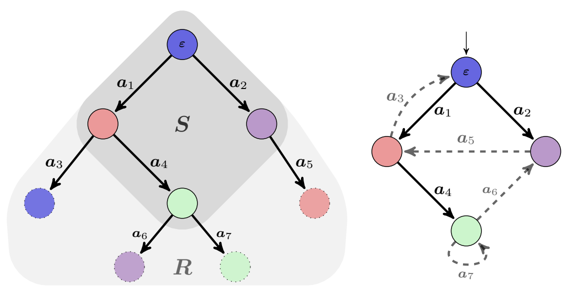
As an underlying data-structure for identifying states based on examples we use symbolic observation tables [23], slightly modified to accommodate for representative and non-representative evidences. The rows of the table correspond to symbolic words (access sequences to states) while the columns are concrete words. Readers unfamiliar with L^* [1] can find in [23] more detailed intuitive explanations of observation tables and their adaptation to the symbolic setting.

Let Σ and $\boldsymbol{\Sigma}$ be two alphabets, let $\boldsymbol{S} \uplus \boldsymbol{R}$ be a prefix-closed subset of $\boldsymbol{\Sigma}^*$ and let $\psi = \{\psi_{\boldsymbol{s}}\}_{\boldsymbol{s} \in \boldsymbol{S}}$ be a family of total surjective functions of the form $\psi_{\boldsymbol{s}} : \Sigma \to \boldsymbol{\Sigma}_{\boldsymbol{s}}$, where $\biguplus_{\boldsymbol{s} \in \boldsymbol{S}} \boldsymbol{\Sigma}_{\boldsymbol{s}} = \boldsymbol{\Sigma}$. A *balanced symbolic Σ-tree* is a tuple $(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi)$, where for

$\boldsymbol{T}$

		E	
		ε	v
S	$\boldsymbol{\varepsilon}$	$-$	$+$
	$\boldsymbol{a}_1$	$+$	$-$
	$\boldsymbol{a}_2$	$+$	$+$
	$\boldsymbol{a}_1\boldsymbol{a}_4$	$-$	$-$
R	$\boldsymbol{a}_1\boldsymbol{a}_3$	$-$	$+$
	$\boldsymbol{a}_2\boldsymbol{a}_5$	$+$	$-$
	$\boldsymbol{a}_1\boldsymbol{a}_4\boldsymbol{a}_6$	$+$	$+$
	$\boldsymbol{a}_1\boldsymbol{a}_4\boldsymbol{a}_7$	$-$	$-$

(a)



(b) (c)

Fig. 1: (a) A symbolic observation table, (b) its balanced symbolic Σ-tree, and (c) the conjectured automaton.

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.

The structure of a balanced tree appears in Figure 1-(b) together with its corresponding automaton at Figure 1-(c). The underlying intuition is that elements of $\boldsymbol{S}$, also known as access sequences, correspond to a spanning tree of the transition graph of the automaton to be learned, while elements of the boundary $\boldsymbol{R}$ correspond to back- and cross-edges relative to this spanning tree.

Definition 2 (Symbolic Observation Table). *A* symbolic observation table *is a tuple* $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu, \hat{\mu})$ *such that*

- Σ *is an alphabet,*
- $(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi)$ *is a balanced symbolic* Σ*-tree,*
- $E \subseteq \Sigma^*$ *is a set of* distinguishing words*,*
- $\boldsymbol{f} : (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E \to \{-, +\}$ *is the* symbolic classification function*,*
- $\mu : \boldsymbol{\Sigma} \to 2^{\Sigma} - \{\emptyset\}$ *is the* evidence function*, where* $\mu(\boldsymbol{a}) \subseteq [\![\boldsymbol{a}]\!]$ *for all* $\boldsymbol{a} \in \boldsymbol{\Sigma}$*,*
- $\hat{\mu} : \boldsymbol{\Sigma} \to \Sigma$ *is the* representative function*, where* $\hat{\mu}(\boldsymbol{a}) \in \mu(\boldsymbol{a})$ *for all* $\boldsymbol{a} \in \boldsymbol{\Sigma}$*.*

The evidence and representative functions are extended to symbolic words in $\boldsymbol{S} \cup \boldsymbol{R}$ as follows:

$$\begin{array}{ll} \mu(\boldsymbol{\varepsilon}) = \{\varepsilon\} & \hat{\mu}(\boldsymbol{\varepsilon}) = \varepsilon \\ \mu(\boldsymbol{s} \cdot \boldsymbol{a}) = \hat{\mu}(\boldsymbol{s}) \cdot \mu(\boldsymbol{a}) & \hat{\mu}(\boldsymbol{s} \cdot \boldsymbol{a}) = \hat{\mu}(\boldsymbol{s}) \cdot \hat{\mu}(\boldsymbol{a}). \end{array} \tag{1}$$

The symbolic characteristic function values are based on the representative of the symbolic prefix rather than the set of all evidences, i.e., to fill the $(\boldsymbol{s}, e)$ entry in the table we let $\boldsymbol{f}(\boldsymbol{s}, e) = f(\hat{\mu}(\boldsymbol{s}) \cdot e)$, where f is the characteristic function of the target language. With every $\boldsymbol{s} \in \boldsymbol{S} \cup \boldsymbol{R}$ we associate a *residual classification function* defined as $\boldsymbol{f}_{\boldsymbol{s}}(e) = \boldsymbol{f}(\boldsymbol{s}, e)$. The symbolic sample associated with $\boldsymbol{T}$ is the set $\boldsymbol{M}_{\boldsymbol{T}} = (\boldsymbol{S} \cup \boldsymbol{R}) \cdot E$ and the concrete sample is $M_{\boldsymbol{T}} = \mu(\boldsymbol{S} \cup \boldsymbol{R}) \cdot E$.

Handling multiple evidences to determine partition boundaries is the major novel feature in learning symbolic automata. Evidences of the same symbol should behave the same and when this is not the case, that is, when two concrete letters in the evidence of a symbol lead to different residual functions, we call this a manifestation of *evidence incompatibility*. The rigorous detection and resolution of evidence incompatibility is a major contribution of the algorithm presented in this work. The topic has also been addressed in [17] but in an unsatisfactory manner, leaving the transition function undefined outside the evidence. Evidence incompatibility can be characterized and measured as follows.

Definition 3 (Incompatibility Instance). *Let* $\mu_{\boldsymbol{s}} = \bigcup_{\boldsymbol{a} \in \boldsymbol{\Sigma}_{\boldsymbol{s}}} \mu(\boldsymbol{a})$ *be the set of all evidences for state* $\boldsymbol{s}$. *A state* $\boldsymbol{s} \in \boldsymbol{S}$ *has an* incompatibility instance *at evidence* $a \in \mu_{\boldsymbol{s}}$ *when* $f_{\hat{\mu}(\boldsymbol{s}) \cdot a} \neq f_{\hat{\mu}(\boldsymbol{s}) \cdot \hat{\mu}(\psi_{\boldsymbol{s}}(a))}$, *and this fact is denoted* $\textsc{inc}(\boldsymbol{s}, a)$. *The* evidence incompatibility degree *associated with* $\boldsymbol{s}$ *is* $M(\boldsymbol{s}) = |\{a \in \mu_{\boldsymbol{s}} : \textsc{inc}(\boldsymbol{s}, a)\}|$.

Definition 4 (Table Properties). *A table* $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu, \hat{\mu})$ *is*

– Closed *if* $\forall \boldsymbol{r} \in \boldsymbol{R}$, $\exists \boldsymbol{s} \in \boldsymbol{S}$, $\boldsymbol{f}_{\boldsymbol{r}} = \boldsymbol{f}_{\boldsymbol{s}}$,
– Reduced *if* $\forall \boldsymbol{s}, \boldsymbol{s'} \in \boldsymbol{S}$, $\boldsymbol{f}_{\boldsymbol{s}} \neq \boldsymbol{f}_{\boldsymbol{s'}}$, *and*
– Evidence compatible *if* $M(\boldsymbol{s}) = 0$, $\forall \boldsymbol{s} \in \boldsymbol{S}$.

The following result [23] is the natural generalization of the derivation of an automaton from an observation table [1] to the symbolic setting.

Theorem 1 (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 $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu, \hat{\mu})$ 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 $\boldsymbol{f}_{\boldsymbol{r}} = \boldsymbol{f}_{\boldsymbol{s}}$, is well defined. The automaton derived from the table is then $\boldsymbol{\mathcal{A}}_{\boldsymbol{T}} = (\Sigma, \boldsymbol{\Sigma}, \psi, Q, \boldsymbol{\delta}, q_0, F)$, where $Q = \boldsymbol{S}$, $q_0 = \boldsymbol{\varepsilon}$, $F = \{\boldsymbol{s} \in \boldsymbol{S} : \boldsymbol{f}_{\boldsymbol{s}}(\varepsilon) = +\}$, and $\boldsymbol{\delta} : Q \times \boldsymbol{\Sigma} \to Q$ 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, $\boldsymbol{\mathcal{A}}_{\boldsymbol{T}}$ classifies correctly via $\boldsymbol{f}$ the symbolic sample and, due to evidence compatibility, this classification agrees with the characteristic function f on the concrete sample. □

4 The Symbolic Learning Algorithm

In this section we present the symbolic learning algorithm using a high-cardinality bounded subset of $\mathbb{N}$ or $\mathbb{R}$ as an input alphabet. The concrete semantics of each symbolic letter is a sub-interval of the alphabet. We disallow disconnected partition blocks, for example, two subsets of even and odd numbers, respectively. Thus, if two disconnected intervals take the same transition, two symbolic letters will be considered. In this setting, the endpoints of an interval associated with a symbolic letter are such that all evidence points between them have the same residual function, while the nearest points outside the interval have different residuals. The algorithm adapts easily to other alphabet types as we will show in Section 5.

Algorithm 1 A sampling-based symbolic learning algorithm

```
1:  $learned = false$
2:  INITTABLE($\boldsymbol{T}$)
3:  **repeat**
4:      **while** $\boldsymbol{T}$ is not closed **or** not evidence compatible **do**
5:          CLOSE
6:          EVCOMP
7:      **end while**
8:      **if** EQ($\mathcal{A}_{\boldsymbol{T}}$) **then**                        ▷ check hypothesis $\mathcal{A}_{\boldsymbol{T}}$
9:          $learned = true$
10:     **else**                              ▷ a counter-example $w$ is provided
11:         COUNTEREX($\mathcal{A}_{\boldsymbol{T}}, w$)                 ▷ process counter-example
12:     **end if**
13: **until** $learned$
```

Procedure 2 Initialize the table

```
1: **procedure** INITTABLE($\boldsymbol{T}$)
2:     $\boldsymbol{\Sigma}_{\boldsymbol{\varepsilon}} = \{\boldsymbol{a}\}$; $\boldsymbol{S} = \{\boldsymbol{\varepsilon}\}$; $\boldsymbol{R} = \{\boldsymbol{a}\}$; $E = \{\varepsilon\}$          ▷ $\boldsymbol{a}$ is a new symbol
3:     INITSYMBOL($\boldsymbol{a}$)
4:     Ask MQ($u$) for all $u \in \mu(\boldsymbol{a}) \cup \{\varepsilon\}$
5:     $\boldsymbol{f}(\boldsymbol{\varepsilon}) = f(\varepsilon)$; $\boldsymbol{f}(\boldsymbol{a}) = f(\hat{\mu}(\boldsymbol{a}))$
6:     $\boldsymbol{T} = (\Sigma, \boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{R}, \psi, E, \boldsymbol{f}, \mu, \hat{\mu})$
7: **end procedure**
```

The symbolic learning algorithm (Algorithm 1) alternates between two phases. In the first phase it attempts to make the table closed and evidence compatible so as to construct a symbolic automaton. In the second phase, after formulating an equivalence query (EQ), it processes the provided counter-example which renders the table not closed or evidence incompatible. These phases alternate until no counter-example is found. Note that the table, by construction, is always kept reduced. We use MQ as a shorthand for membership queries.

Table Initialization (Procedure 2). The algorithm builds an initial observation table $\boldsymbol{T}$, with $\boldsymbol{\Sigma}_{\varepsilon} = \{\boldsymbol{a}\}$, $\boldsymbol{S} = \{\boldsymbol{\varepsilon}\}$, $\boldsymbol{R} = \{\boldsymbol{a}\}$, $E = \{\varepsilon\}$. The newly introduced symbol $\boldsymbol{a}$ is initialized with concrete semantics, evidence and a representative, via the procedure INITSYMBOL which is invoked each time a new state is introduced. Then membership queries are posed to update $\boldsymbol{f}$ and fill the table.

Symbol Initialization (Procedure 3). For a new symbolic letter $\boldsymbol{a}$ we let $[\![\boldsymbol{a}]\!] = \Sigma$ and as an evidence $\mu(\boldsymbol{a})$ we take a set of k concrete letters, denoted by $sample(\Sigma, k)$. This set can be selected randomly or be the result of a more adaptive process that may depend on the outcome of membership queries. One element of the evidence, denoted by $select(\mu(\boldsymbol{a}))$, is chosen as a representative and will be used to fill table entries

Procedure 3 Initialize new symbol $\boldsymbol{a}$

```
1: procedure InitSymbol($\boldsymbol{a}$)
2:     $[\![\boldsymbol{a}]\!] = \Sigma$
3:     $\mu(\boldsymbol{a}) = sample(\Sigma, k)$
4:     $\hat{\mu}(\boldsymbol{a}) = select(\mu(\boldsymbol{a}))$
5: end procedure
```

for all rows in which $\boldsymbol{a}$ appears. Already at this stage, some elements of $\mu(\boldsymbol{a})$ may behave differently from the representative and this will flag an evidence incompatibility condition to be treated subsequently.

Table Closing (Procedure 4). A table is not closed when there exists some $\boldsymbol{r} \in \boldsymbol{R}$ without any equivalent element $\boldsymbol{s} \in \boldsymbol{S}$ such that $\boldsymbol{f_r} = \boldsymbol{f_s}$. To render the table closed $\boldsymbol{r}$ should be considered as a new state. To this end, $\boldsymbol{r}$ is moved from $\boldsymbol{R}$ to $\boldsymbol{S}$ with alphabet $\boldsymbol{\Sigma_r} = \{\boldsymbol{a}\}$, where $\boldsymbol{a}$ is a new symbol which is initialized. To balance the table a new word $\boldsymbol{r} \cdot \boldsymbol{a}$ is added to $\boldsymbol{R}$, its evidence $\mu(\boldsymbol{r} \cdot \boldsymbol{a})$ and representative $\hat{\mu}(\boldsymbol{r} \cdot \boldsymbol{a})$ are computed following (1) and membership queries are posed to update f and fill the table.

Procedure 4 Close the table

```
1: procedure Close
2:     Given $\boldsymbol{r} \in \boldsymbol{R}$ such that $\forall \boldsymbol{s} \in \boldsymbol{S}$, $\boldsymbol{f_r} \neq \boldsymbol{f_s}$
3:     $\boldsymbol{S} = \boldsymbol{S} \cup \{\boldsymbol{r}\}$                              ▷ declare $\boldsymbol{r}$ a new state
4:     $\boldsymbol{\Sigma_r} = \{\boldsymbol{a}\}$                                ▷ introduce a new symbol $\boldsymbol{a}$
5:     InitSymbol($\boldsymbol{a}$)
6:     $\boldsymbol{R} = (\boldsymbol{R} - \{\boldsymbol{r}\}) \cup \{\boldsymbol{r} \cdot \boldsymbol{a}\}$                   ▷ add new boundary element
7:     Ask MQ($u$) for all $u \in \mu(\boldsymbol{r} \cdot \boldsymbol{a}) \cdot E$
8:     $\boldsymbol{f_{r \cdot a}} = f_{\hat{\mu}(\boldsymbol{r} \cdot \boldsymbol{a})}$
9: end procedure
```

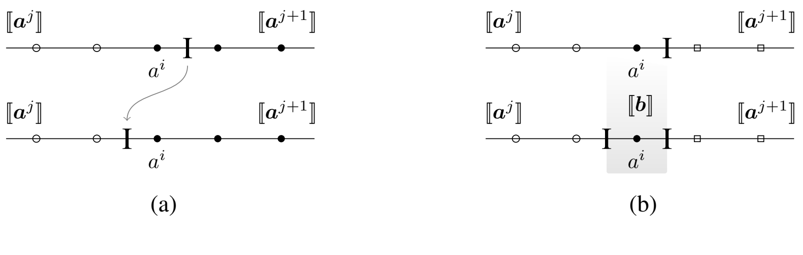


Fig. 2: Evidence incompatibility solved either by (a) boundary modification, or by (b) introducing a new symbol. This depends on the position of the incompatibility instance inside the partition.

Procedure 5 Make evidence compatible

```
 1: procedure EvComp
 2:     Let $\boldsymbol{s} \in \boldsymbol{S}$, for which $M(\boldsymbol{s}) > 0$, where
 3:     $\mu_{\boldsymbol{s}} = \{a^1, \ldots, a^k\}$ such that $a^{i-1} < a^i, \forall i = 2, \ldots, k$
 4:     Let $\boldsymbol{a}^j \in \boldsymbol{\Sigma}_{\boldsymbol{s}}$, $[\![\boldsymbol{a}^j]\!] = [c, c')$, such that $\exists i : f^{i-1} \neq f^i$ for $a^{i-1}, a^i \in \mu(\boldsymbol{a}^j)$
 5:     $p = split(a^{i-1}, a^i)$                                        ▷ new partitioning point
 6:     if $f^i = f^{i+1} = \cdots = f^{i+l+1}$ where $a^i, \ldots, a^{i+l} \in \mu(\boldsymbol{a}^j), a^{i+l+1} \in \mu(\boldsymbol{a}^{j+1})$ then
 7:         $[\![\boldsymbol{a}^j]\!] = [c, p)$; $[\![\boldsymbol{a}^{j+1}]\!] = [p, c') \cup [\![\boldsymbol{a}^{j+1}]\!]$          ▷ change right frontier
 8:         $\mu(\boldsymbol{a}^{j+1}) = (\mu(\boldsymbol{a}^{j+1}) \cup \mu(\boldsymbol{a}^j)) \cap [\![\boldsymbol{a}^{j+1}]\!]$
 9:         $\mu(\boldsymbol{a}^j) = \mu(\boldsymbol{a}^j) \cap [\![\boldsymbol{a}^j]\!]$
10:     else if $f^{i-1} = \cdots = f^{i-l}$ where $a^{i-1}, \ldots, a^{i-l+1} \in \mu(\boldsymbol{a}^j), a^{i-l} \in \mu(\boldsymbol{a}^{j-1})$ then
11:         $[\![\boldsymbol{a}^{j-1}]\!] = [\![\boldsymbol{a}^{j-1}]\!] \cup [c, p)$; $[\![\boldsymbol{a}^j]\!] = [p, c')$           ▷ change left frontier
12:         $\mu(\boldsymbol{a}^{j-1}) = (\mu(\boldsymbol{a}^{j-1}) \cup \mu(\boldsymbol{a}^j)) \cap [\![\boldsymbol{a}^{j-1}]\!]$
13:         $\mu(\boldsymbol{a}^j) = \mu(\boldsymbol{a}^j) \cap [\![\boldsymbol{a}^j]\!]$
14:     else
15:         $\boldsymbol{\Sigma}_{\boldsymbol{s}} = \boldsymbol{\Sigma}_{\boldsymbol{s}} \cup \{\boldsymbol{b}\}$                              ▷ introduce a new symbol $\boldsymbol{b}$
16:         $\boldsymbol{R} = \boldsymbol{R} \cup \{\boldsymbol{s} \cdot \boldsymbol{b}\}$
17:         if $\hat{\mu}(\boldsymbol{a}^j) \leq p$ then
18:             $[\![\boldsymbol{a}^j]\!] = [c, p)$; $[\![\boldsymbol{b}]\!] = [p, c')$
19:         else
20:             $[\![\boldsymbol{b}]\!] = [c, p)$; $[\![\boldsymbol{a}^j]\!] = [p, c')$
21:         end if
22:         $\mu(\boldsymbol{b}) = \mu(\boldsymbol{a}^j) \cap [\![\boldsymbol{b}]\!]$; $\mu(\boldsymbol{a}^j) = \mu(\boldsymbol{a}^j) \cap [\![\boldsymbol{a}^j]\!]$
23:         $\hat{\mu}(\boldsymbol{b}) = select(\mu(\boldsymbol{b}))$
24:         $\boldsymbol{f}_{\boldsymbol{s} \cdot \boldsymbol{b}} = f_{\hat{\mu}(\boldsymbol{s} \cdot \boldsymbol{b})}$
25:     end if
26: end procedure
```

Fixing Evidence Incompatibility (Procedure 5). A table is not evidence compatible when the incompatibility degree of a state in $\boldsymbol{S}$ is greater than zero. Evidence incompatibility appears either after the initialization of a symbol, or after a counter-example treatment. It is resolved by consecutive calls to EvComp where each call reduces $M(\boldsymbol{s})$ until the total incompatibility degree of the observation table becomes zero.

For a state $\boldsymbol{s}$, an incompatibility instance at a indicates either that the partition boundary is imprecise or that a transition (and its corresponding symbol) is missing. In the first case, the incompatible evidence a appears next to the boundary of the interval and its classification matches the classification of a neighboring symbol $\boldsymbol{a}'$. In this situation, modifying the boundary so that a is moved to $[\![\boldsymbol{a}']\!]$ resolves the incompatibility. On the the other hand, when the evidence a is in the interior of an interval, or does not behave like a neighboring symbol, the incompatibility is resolved by adding a new symbol and refining the existing partition. These two cases are illustrated in Figures 2-(a) and 2-(b), respectively.

Formally, let $\boldsymbol{s} \in \boldsymbol{S}$ be a state with positive incompatibility degree $M(\boldsymbol{s}) > 0$, and let $\mu_{\boldsymbol{s}} = \{a^1, \ldots, a^k\} \subset \boldsymbol{S}$ be the set of evidences, ordered such that $a^{i-1} < a^i$ for all i. To simplify notation, f^i denotes the residual $f_{\hat{\mu}(\boldsymbol{s}) \cdot a^i}$ when state $\boldsymbol{s}$ is understood from the context. Moreover, let $\boldsymbol{a}^j$ and $\boldsymbol{a}^{j+1}$ denote symbols in $\boldsymbol{\Sigma_s}$ with adjacent semantics, that is, given any three letters $a, b, c \in \Sigma$, with $a < b < c$, then $a \in [\![\boldsymbol{a}^j]\!] \wedge c \in [\![\boldsymbol{a}^{j+1}]\!]$ implies $b \in [\![\boldsymbol{a}^j]\!] \cup [\![\boldsymbol{a}^{j+1}]\!]$.

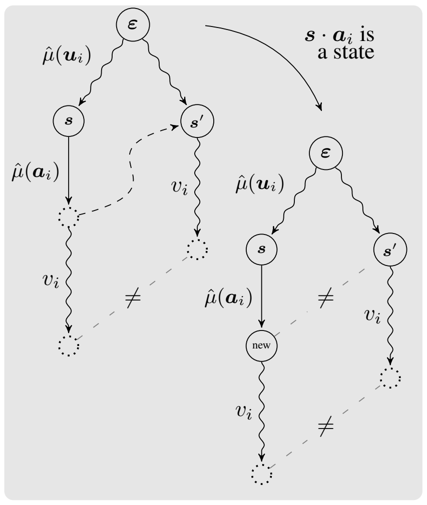
Let $a^{i-1}, a^i \in \mu_{\boldsymbol{s}}$ be two evidences from the same interval that behave differently, $f^{i-1} \neq f^i$, and let $\boldsymbol{a}^j \in \boldsymbol{\Sigma_s}$ be the symbol such that $a^{i-1}, a^i \in \mu(\boldsymbol{a}^j)$ where $[\![\boldsymbol{a}^j]\!] = [c, c')$. Procedure $p = split(a^{i-1}, a^i)$ returns a point $p \in (a^{i-1}, a^i)$ between them. We let $split$ return the middle point, $split(a, a') = (a + a')/2$. One can think of more sophisticated methods, based on binary search, that can be applied instead.

Procedure 5 fixes the incompatibility by separating a^{i-1} and a^i and mapping them to different symbols. The way this separation is realized, with or without introducing a new symbol, depends on the positions of a^{i-1} and a^i in the set of evidences and the residual functions of their neighboring intervals.

1. *Boundary modification*. Suppose the incompatibility instance is at $a^i \in \mu(\boldsymbol{a}^j)$ and that all other evidences $\mu(\boldsymbol{a}^j)$ to the right of a^i behave like $\min \mu(\boldsymbol{a}^{j+1})$. By changing the partition boundaries and moving a^i from $[\![\boldsymbol{a}^j]\!]$ to $[\![\boldsymbol{a}^{j+1}]\!]$, the incompatibility instance at a^i is eliminated. The new boundary between these two intervals is set to p, see Figure 2-(a). The symmetric case, where the incompatibility occurs at $a^{i-1} \in \boldsymbol{a}^j$ with all other evidences of $\mu(\boldsymbol{a}^j)$ on its left behaving like $\max \mu(\boldsymbol{a}^{j-1})$, is treated similarly.
2. *Symbol introduction*. When the above condition does not hold and boundary modification cannot be applied, the incompatibility is solved by refining the partition. The semantics $[\![\boldsymbol{a}^j]\!]$ is split into two intervals $[c, p)$ and $[p, c')$, a new symbol $\boldsymbol{b}$ is introduced and the interval not containing $\hat{\mu}(\boldsymbol{a}^j)$ is moved from $[\![\boldsymbol{a}^j]\!]$ to $[\![\boldsymbol{b}]\!]$ along with the evidences it contains, see Figure 2-(b).

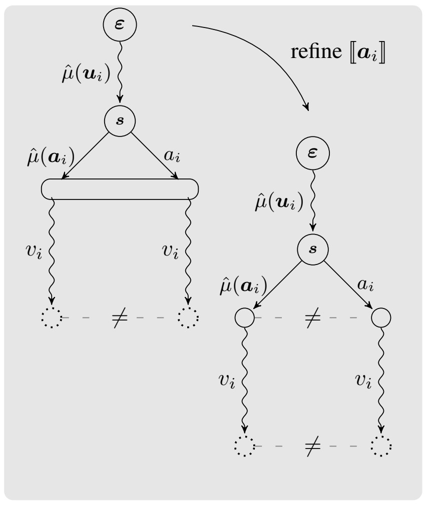
Processing Counter-Examples (Procedure 6). A counter-example is a word w misclassified by the current hypothesis. The automaton should be modified to classify w correctly while remaining compatible with the evidence accumulated so far. These modifications can be of two major types that we call *vertical* and *horizontal*. The first type, which is the only possible modification in concrete learning, involves the discovery of a new state $\boldsymbol{s} \cdot \boldsymbol{a}$. A counter-example which demonstrates that some letter a took a wrong transition $\boldsymbol{\delta}(\boldsymbol{s}, a)$ has a horizontal effect that fixes a transition or adds a new one without creating a new state. The procedure described in the sequel reacts to the counter-example by adding a to the evidence of $\boldsymbol{s}$ and thus modifying the table, which should then be made closed and evidence compatible before we continue with a new hypothesis. The same counter-example is tested again and when it is correctly classified, we proceed by posing a new equivalence query. We treat counter-examples using a symbolic variant of the breakpoint method introduced in [26]. A similar method has been proposed in [16].

Let $\boldsymbol{\mathcal{A}_T}$ be a symbolic automaton derived from a symbolic table $\boldsymbol{T}$, and let $w = a_1 \cdots a_{|w|}$ be a counter-example whose symbolic image is $\boldsymbol{a}_1 \cdots \boldsymbol{a}_{|w|}$. An *i-factorization* of w is $w = u_i \cdot a_i \cdot v_i$ such that $u_i = a_1 \cdots a_{i-1}$ and $v_i = a_{i+1} \cdots a_{|w|}$. For every



ε
$\boldsymbol{s}\cdot\boldsymbol{a}_i$ is a state
$\hat{\mu}(\boldsymbol{u}_i)$
$\boldsymbol{s}$
$\boldsymbol{s}'$
$\hat{\mu}(\boldsymbol{a}_i)$
v_i
$\neq$
new

(a)



ε
refine $[\![\boldsymbol{a}_i]\!]$
$\hat{\mu}(\boldsymbol{u}_i)$
$\boldsymbol{s}$
$\hat{\mu}(\boldsymbol{a}_i)$
a_i
v_i
$\neq$

(b)

Fig. 3: A counter-example expands a hypothesis either (a) vertically, discovering a new state; or (b) horizontally, modifying the alphabet partition in a state.

i-factorization of w, we let $\boldsymbol{u}_i$ be the symbolic image of u_i, and $\boldsymbol{s_i} = \boldsymbol{\delta}(\boldsymbol{\varepsilon}, \boldsymbol{u}_i \cdot \boldsymbol{a}_i)$ be the symbolic state (an element of $\boldsymbol{S}$) reached in $\boldsymbol{\mathcal{A}_T}$ after reading $\boldsymbol{u}_i \cdot \boldsymbol{a}_i$.

Proposition 1 (Symbolic Breakpoint). *If w is a counter-example to $\boldsymbol{\mathcal{A}_T}$ then there exists an i-factorization of w such that either*

$$f(\hat{\mu}(\boldsymbol{s}_{i-1}) \cdot a_i \cdot v_i) \neq f(\hat{\mu}(\boldsymbol{s}_{i-1}) \cdot \hat{\mu}(\boldsymbol{a}_i) \cdot v_i) \tag{2}$$

or

$$f(\hat{\mu}(\boldsymbol{s}_{i-1} \cdot \boldsymbol{a}_i) \cdot v_i) \neq f(\hat{\mu}(\boldsymbol{s}_i) \cdot v_i) \tag{3}$$

Proof. Condition (2) states that a_i is not well represented by $\hat{\mu}(\boldsymbol{a}_i)$ while Condition (3) implies $\boldsymbol{s}_{i-1} \cdot \boldsymbol{a}_i$ is a new state different from $\boldsymbol{s}_i$, see Figure 3. We prove the proposition assuming that none of the above inequalities holds for any i-factorization of w. By using alternatively the negations of (2) and (3) for all values of i, we conclude that $f(\hat{\mu}(\boldsymbol{s}_0) \cdot a_1 \cdot v_1) = f(\hat{\mu}(\boldsymbol{s}_{|w|}))$, where $\hat{\mu}(\boldsymbol{s}_0) \cdot a_1 \cdot v_1$ is the counter-example and $\boldsymbol{s}_{|w|}$ is the state reached in $\boldsymbol{\mathcal{A}_T}$ after reading w. Thus w cannot be a counter-example. □

Procedure 6 iterates over i values and checks whether one of the conditions (2) and (3) holds for some i. We let i take values in a monotonically descending order and keep the suffixes as short as possible. In this case, it suffices to compare $f(\hat{\mu}(\boldsymbol{s}_{i-1} \cdot \boldsymbol{a}_i) \cdot v_i)$ and $f(\hat{\mu}(\boldsymbol{s}_{i-1}) \cdot a_i \cdot v_i)$ with the classification of the counter-example, which is kept in a flag variable. In line 5, Condition (3) is checked and if it holds, adding v_i to E will distinguish between states $\boldsymbol{s}_{i-1} \cdot \boldsymbol{a}_i$ and $\boldsymbol{s}_i$, resulting in a table which is not closed. Otherwise, if Condition (2) holds, which is checked in line 9, the letter a_i is added to the evidence of $\boldsymbol{a}_i$ and new membership queries are posed. These queries will render the table evidence incompatible and will lead to refining $[\![\boldsymbol{a}_i]\!]$. The suffix v_i is added to E in case it is the only witness for the incompatibility. Note that checking the conditions involves supplementary membership queries, based on the suffix of the counter-example w, where the prefix u_i of w is replaced by $\hat{\mu}(\boldsymbol{s}_{i-1})$, the representative of its shortest equivalent symbolic word in the table. Both cases will lead to a new conjectured automaton which might still not classify w correctly. In that case, the procedure should be invoked with the same counter-example and the new hypothesis until $\mathcal{A}_{\boldsymbol{T}}$ classifies w correctly.

Procedure 6 Counter-example treatment

```
 1: procedure CounterEx($\mathcal{A}_{\boldsymbol{T}}, w$)
 2:     flag $= f(\hat{\mu}(\boldsymbol{\delta}(\boldsymbol{\varepsilon}, \boldsymbol{w})))$        ▷ flag $= f(w)$ when iterating on $1, \ldots, |w|$
 3:     for $i = |w|, \ldots, 1$ do
 4:         For an $i$-factorization $w = u_i \cdot a_i \cdot v_i$
 5:         if $f(\hat{\mu}(\boldsymbol{s}_{i-1} \cdot \boldsymbol{a}_i) \cdot v_i) \neq$ flag then        ▷ check (3)
 6:             $E = E \cup \{v_i\}$        ▷ add a new distinguishing word
 7:             Ask MQ$(u)$ for all $u \in \mu(\boldsymbol{S} \cup \boldsymbol{R}) \cdot v_i$
 8:             break
 9:         else if $f(\hat{\mu}(\boldsymbol{s}_{i-1}) \cdot a_i \cdot v_i) \neq$ flag then        ▷ check (2)
10:             $\mu(\boldsymbol{a}_i) = \mu(\boldsymbol{a}_i) \cup \{a_i\}$        ▷ add new evidence
11:             Ask MQ$(u)$ for all $u \in \hat{\mu}(\boldsymbol{s}_{i-1}) \cdot a_i \cdot E$
12:             if $M(\boldsymbol{s}_i) = 0$ then
13:                 $E = E \cup \{v_i\}$        ▷ add distinguishing word
14:                 Ask MQ$(u)$ for all $u \in \mu(\boldsymbol{S} \cup \boldsymbol{R}) \cdot v_i$
15:             end if
16:             break
17:         end if
18:     end for
19: end procedure
```

Example 1. We demonstrate the working of the algorithm in learning a target language L over the alphabet $\Sigma = [0, 100) \subseteq \mathbb{R}$. The observation tables, semantic functions and hypotheses used in this example are shown in Figures 4 and 5.

The table is initialized with $\boldsymbol{S} = \{\boldsymbol{\varepsilon}\}$ and $E = \{\varepsilon\}$. To determine the alphabet partition at the initial state $\boldsymbol{\varepsilon}$, the learner asks membership queries for the randomly selected one-letter words $\{13, 42, 68, 78, 92\}$. All words in this set except 13 are rejected. Consequently, there are at least two distinct intervals that we take $split(13, 42) = 27$ as their boundary. Each interval is represented by a symbolic letter resulting in $\boldsymbol{\Sigma}_{\varepsilon} = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$, $\mu(\boldsymbol{a}_1) = \{13\}$, $\hat{\mu}(\boldsymbol{a}_1) = 13$, $\mu(\boldsymbol{a}_2) = \{42, 68, 78, 92\}$, and $\hat{\mu}(\boldsymbol{a}_2) = 68$. The representatives are randomly chosen from the set of evidences. The semantics, ψ maps all letters smaller than 27 to $\boldsymbol{a}_1$, and maps the rest to $\boldsymbol{a}_2$, that is, $[\![\boldsymbol{a}_1]\!] = [0, 27)$ and $[\![\boldsymbol{a}_2]\!] = [27, 100)$. The table boundary updates to $\boldsymbol{R} = \{\boldsymbol{a}_1, \boldsymbol{a}_2\}$ and the observation table is $\boldsymbol{T}_0$, shown in Figure 4.

Table $\boldsymbol{T}_0$ is not closed and in order to fix this, the learner moves $\boldsymbol{a}_1$ to the set of states $\boldsymbol{S}$. To find the possible partitions of Σ at this new state $\boldsymbol{a}_1$, the learner randomly chooses a sample $\{2, 18, 26, 46, 54\}$ of letters and asks membership queries concerning the words in $\{13 \cdot 2, 13 \cdot 18, 13 \cdot 26, 13 \cdot 46, 13 \cdot 54\}$. Note that the prefix used here is the representative of $\boldsymbol{a}_1$. The teacher classifies all words as rejected. The new table is $\boldsymbol{T}_1$ with $\boldsymbol{\Sigma}_{\boldsymbol{a}_1} = \{\boldsymbol{a}_3\}$, $\mu(\boldsymbol{a}_3) = \{2, 18, 26, 46, 54\}$, $\hat{\mu}(\boldsymbol{a}_3) = 18$, and $[\![\boldsymbol{a}_3]\!] = [0, 100)$. The new table is closed and the first hypothesis $\boldsymbol{\mathcal{A}}_1$ is conjectured.

The hypothesis is tested on a set of words, randomly chosen from some distribution, typically unknown to the learner. After some successful tests, a word $35 \cdot 52 \cdot 11$ is found, which is accepted by $\boldsymbol{\mathcal{A}}_1$ but is outside the target language. The learner takes this word as a counter-example and analyzes it using the symbolic breakpoint method. At iteration $i = 2$ of Procedure 6, condition (3) is violated, in particular $\text{MQ}(\hat{\mu}(\boldsymbol{\varepsilon} \cdot \boldsymbol{a}_2) \cdot 11) = \text{MQ}(68 \cdot 11) \neq flag = +$. Thus, the suffix 11 is added as a distinguishing word to E. The observation table $\boldsymbol{T}_2$ obtained after adding the new suffix is, as expected, not closed. The table is made closed by letting $\boldsymbol{a_2}$ be a new state, resulting in table $\boldsymbol{T}_3$, where $\boldsymbol{\Sigma}_{\boldsymbol{a}_2} = \{\boldsymbol{a}_4, \boldsymbol{a}_5\}$, $\mu(\boldsymbol{a}_4) = \{17, 27\}$, $\hat{\mu}(\boldsymbol{a}_4) = 17$, $[\![\boldsymbol{a}_4]\!] = [0, 45)$, $\mu(\boldsymbol{a}_5) = \{64, 72, 94\}$, $\hat{\mu}(\boldsymbol{a}_5) = 72$ and $[\![\boldsymbol{a}_5]\!] = [45, 100)$. The corresponding new conjecture is $\boldsymbol{\mathcal{A}}_3$.

Automaton $\boldsymbol{\mathcal{A}}_3$ is tested and a counter-example $12 \cdot 73 \cdot 4$ is provided. The breakpoint method discovers that condition (2) is violated, because letter 73 is not part of the semantics of $\boldsymbol{a}_3$. This letter is added as a new evidence to $\mu(\boldsymbol{a}_3)$. The evidence inconsistency is solved by splitting the existing partition into two subintervals. A new symbol $\boldsymbol{a}_6$ is added to $\boldsymbol{\Sigma}_{\boldsymbol{a}_1}$, such that $\mu(\boldsymbol{a}_6) = \{73\}$ and $[\![\boldsymbol{a}_6]\!] = [63, 100)$. The new observation table and hypothesis automaton are $\boldsymbol{T}_4$ and $\boldsymbol{\mathcal{A}}_4$, respectively.

The next counter-example $52 \cdot 47$, also adds a new evidence, this time to symbol $\boldsymbol{a}_5$. The classification of the new evidence matches the classification of $\boldsymbol{a}_4$, which is a neighboring symbol. The boundary between $[\![\boldsymbol{a}_4]\!]$ and $[\![\boldsymbol{a}_5]\!]$ is moved from 45 to 55, thus resolving the evidence incompatibility. The new hypothesis $\boldsymbol{\mathcal{A}}_5$ is successfully tested without discovering any other counter-example and the algorithm terminates while returning $\boldsymbol{\mathcal{A}}_5$ as an answer. $\square$

$\boldsymbol{T}_0$

	ε
$\boldsymbol{\varepsilon}$	$-$
$\boldsymbol{a}_1$	$+$
$\boldsymbol{a}_2$	$-$

$\boldsymbol{T}_1$

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

$\boldsymbol{T}_2$

	ε	11
$\boldsymbol{\varepsilon}$	$-$	$+$
$\boldsymbol{a}_1$	$+$	$-$
$\boldsymbol{a}_2$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_3$	$-$	$+$

$\boldsymbol{T}_3$

	ε	11
$\boldsymbol{\varepsilon}$	$-$	$+$
$\boldsymbol{a}_1$	$+$	$-$
$\boldsymbol{a}_2$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_3$	$-$	$+$
$\boldsymbol{a}_2\boldsymbol{a}_4$	$-$	$-$
$\boldsymbol{a}_2\boldsymbol{a}_5$	$+$	$-$

$\boldsymbol{T}_{4-5}$

	ε	11
$\boldsymbol{\varepsilon}$	$-$	$+$
$\boldsymbol{a}_1$	$+$	$-$
$\boldsymbol{a}_2$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_3$	$-$	$+$
$\boldsymbol{a}_1\boldsymbol{a}_6$	$+$	$-$
$\boldsymbol{a}_2\boldsymbol{a}_4$	$-$	$-$
$\boldsymbol{a}_2\boldsymbol{a}_5$	$+$	$-$

Fig. 4: Observation Tables used in Example 1.

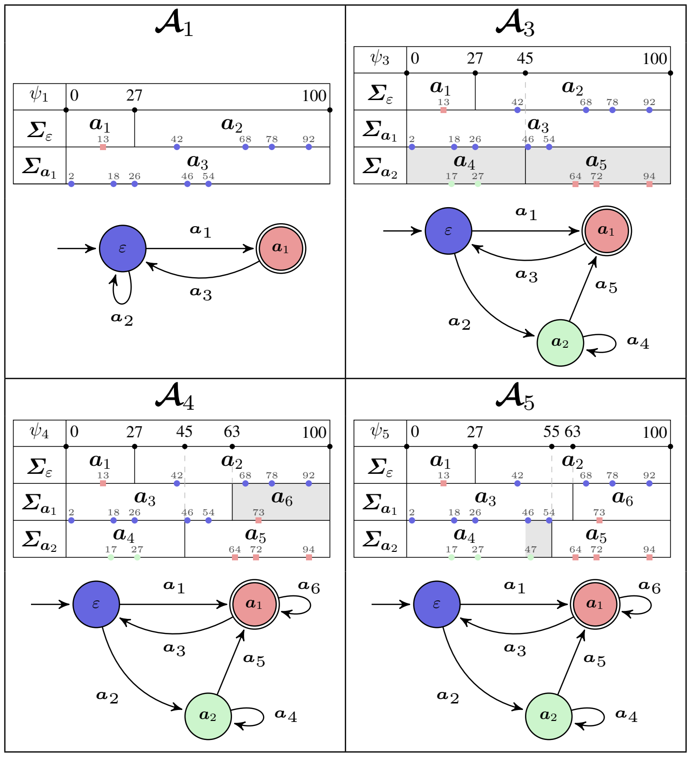


Fig. 5: Symbolic automata and semantics function learned in Example 1.

5 Adaptation to Boolean Alphabets

We demonstrate the versatility of the algorithm by adapting it to languages over the alphabet $\Sigma = \mathbb{B}^n$ of Boolean vectors accessed by variables $\{x_1, \ldots, x_n\}$. All components of the algorithm remain the same except the construction of alphabet partitions and their modification due to evidence incompatibility. These should be adapted to the particular nature of the Boolean hyper-cube. The concrete semantics of the symbolic letters in a state q will be defined by a function $\psi_q : \mathbb{B}^n \to \boldsymbol{\Sigma}_q$. Let $\mu_{\boldsymbol{s}}$ be the set of all evidences for state $\boldsymbol{s}$. At any given moment, the raw data for inducing the alphabet partition at $\boldsymbol{s}$ is the sample $\{(a^i, f^i) : a^i \in \mu_{\boldsymbol{s}}\}$ where for every a^i, $f^i = f_{\hat{\mu}(\boldsymbol{s})\cdot a^i}$ is the residual associated with a^i. Let $\mathcal{F}_{\boldsymbol{s}}$ denote the set of all observed distinct residuals associated with the one-letter successors of $\boldsymbol{s}$. On our way to construct ψ_q, we first derive another function $\psi_{\boldsymbol{s}} : \mathbb{B}^n \to \mathcal{F}_{\boldsymbol{s}}$ associated with any $\boldsymbol{s} \in \boldsymbol{S}$. The function $\psi_{\boldsymbol{s}}$ is compatible with the sample if it agrees with it on the elements of $\mu_{\boldsymbol{s}}$.

We represent both $\psi_{\boldsymbol{s}}$ and ψ_q by isomorphic decision trees [8] whose leaf nodes are labeled by elements of $\mathcal{F}_{\boldsymbol{s}}$ and $\boldsymbol{\Sigma}_q$, respectively. By abuse of notation, we use ψ for the functions and for their associated decision trees. We first build $\psi_{\boldsymbol{s}}$ as a decision

Procedure 7 Make evidence compatible (Boolean alphabets)

```
1: procedure EvComp
2:     Let $\boldsymbol{s} \in \boldsymbol{S}$ be a state for which $M(\boldsymbol{s}) > 0$
3:     Update($\psi_{\boldsymbol{s}}$)                              ▷ build a tree consistent with sample
4:     for all $h \in \mathcal{F}_{\boldsymbol{s}}$ do                              ▷ for all existing residuals
5:         if $\exists \boldsymbol{a} \in \boldsymbol{\Sigma}_{\boldsymbol{s}}$ s.t. $h = f_{\hat{\mu}(\boldsymbol{s})\cdot\hat{\mu}(\boldsymbol{a})}$ then
6:                                      ▷ $h$ is already associated with an existing symbol $\boldsymbol{a}$
7:             $\mu(\boldsymbol{a}) = \{a^i \in \mu_{\boldsymbol{s}} : f^i = h\}$                              ▷ update evidence
8:             $[\![\boldsymbol{a}]\!] = \bigcup\{[\![t]\!] : t \in leaves(\psi_{\boldsymbol{s}})$ and $label(t) = h\}$   ▷ update semantics
9:         else                                  ▷ $h$ does not match any pre-existing residual
10:            $\boldsymbol{\Sigma}_{\boldsymbol{s}} = \boldsymbol{\Sigma}_{\boldsymbol{s}} \cup \{\boldsymbol{b}\}$                              ▷ introduce a new symbol
11:            $\boldsymbol{R} = \boldsymbol{R} \cup \{\boldsymbol{s} \cdot \boldsymbol{b}\}$                              ▷ and a new candidate state
12:            $\mu(\boldsymbol{b}) = \{a^i \in \mu_{\boldsymbol{s}} : f^i = h\}$                              ▷ define evidence
13:            $\hat{\mu}(\boldsymbol{b}) = select(\mu(\boldsymbol{b}))$                              ▷ select representative
14:            $[\![\boldsymbol{b}]\!] = \bigcup\{[\![t]\!] : t \in leaves(\psi_{\boldsymbol{s}})$ and $label(t) = h\}$   ▷ update semantics
15:        end if
16:    end for
17: end procedure
```

tree where all evidences mapped to the same leaf node agree on their residual function. Hence, learning alphabet partitions is an instance of learning decision trees using algorithms such as CART [8], ID3 [25], or ID5 [28] that construct a tree compatible with a labeled sample.

These algorithms work roughly as follows. They start with a tree consisting of a single root node, with which all sample points are associated. A node is said to be *pure* if all its sample points have the same label. For each impure node, two descendants are created and the sample is split among them based on the value of some selected variable x_i. The variable is chosen according to some purity measure, such as information gain, that characterizes the quality of the split based on each variable. The selection is greedy and the algorithm terminates when the tree becomes sample compatible and sends each sample point to a pure leaf node.

Evidence incompatibility in a state $\boldsymbol{s}$ appears when the decision tree $\psi_{\boldsymbol{s}}$ is not compatible with the sample. This may happen in three occasions during the execution of the algorithm, the first being symbol initialization. Recall that when a new state $\boldsymbol{s}$ is introduced, we create a new symbol $\boldsymbol{a}$ and collect evidences for it, which may have different residuals while being associated with the same single root node. The second occasion occurs when new evidence is added to a symbol, making a leaf node in the tree impure. Finally, when some new suffix is added to E, the set $\mathcal{F}_{\boldsymbol{s}}$ of distinct residuals (rows in the table) may increase and the labels of existing evidences may change.

The simplest way to fix a decision tree is to split impure leaf nodes until purification. However, this may lead to very deep trees and it is preferable to reconstruct the tree each time the sample is updated in a way that leads to incompatibility. In the simple (second) case where a new evidence is added, we can use an incremental algorithm such as ID5

[28], which restructures only parts of the tree that need to be modified, leaving the rest of the tree intact. This algorithm produces the same tree as a non-incremental algorithm would, while performing less computation. In the third case, we build the tree from scratch and this is also what we do after initialization where the incremental and non-incremental algorithms coincide.

Once a tree $\psi_{\boldsymbol{s}}$ is made compatible with the sample, the semantics of the symbolic alphabet, expressed via $\psi_{\boldsymbol{s}}$, is updated. This is nothing but mapping the leaves of $\psi_{\boldsymbol{s}}$ to Σ_q. Had we wanted to follow the "convex" partition approach that we used for numerical alphabets, we should have associated a fresh symbol with each leaf node of the tree, thus letting $[\![\boldsymbol{a}]\!]$ be a cube/term for every $\boldsymbol{a} \in \boldsymbol{\Sigma}_q$. We prefer, however, to associate the same symbol with multiple leaf nodes that share the same label, allowing the semantics of a symbol to be a finite union of cubes. This way $|\boldsymbol{\Sigma_s}| = |\mathcal{F}_{\boldsymbol{s}}|$ and there is at most one symbol that labels a transition between any pair of states.

Each time $\psi_{\boldsymbol{s}}$ is restructured, we modify ψ_q as follows. First, with each symbol $\boldsymbol{a}$ that already exists, we re-associate the leaves that agree with the labels of its representative (note that the representative of an existing symbol never changes). Then, in the case where the set $\mathcal{F}_{\boldsymbol{s}}$ of distinct residuals has increased, we introduce a new symbolic letter for each new residual and select its representative. The whole process is described in Procedure 7. We use $[\![t]\!]$ to denote all evidences associated with a leaf node t and $label(t)$ to denote its residual.

$\boldsymbol{T}_0$

	ε
$\boldsymbol{\varepsilon}$	$-$
$\boldsymbol{a}_0$	$-$

$\boldsymbol{T}_1$

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

$\boldsymbol{T}_2$

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

$\boldsymbol{T}_3$

	ε	0000
$\boldsymbol{\varepsilon}$	$-$	$-$
$\boldsymbol{a}_1$	$+$	$-$
$\boldsymbol{a}_0$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2$	$-$	$+$
$\boldsymbol{a}_1\boldsymbol{a}_3$	$+$	$-$

$\boldsymbol{T}_4$

	ε	0000
$\boldsymbol{\varepsilon}$	$-$	$-$
$\boldsymbol{a}_1$	$+$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2$	$-$	$+$
$\boldsymbol{a}_0$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_3$	$+$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_4$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_6$	$+$	$-$

$\boldsymbol{T}_{5-6}$

	ε	0000
$\boldsymbol{\varepsilon}$	$-$	$-$
$\boldsymbol{a}_1$	$+$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2$	$-$	$+$
$\boldsymbol{a}_0$	$-$	$-$
$\boldsymbol{a}_5$	$-$	$+$
$\boldsymbol{a}_1\boldsymbol{a}_3$	$+$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_4$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_6$	$+$	$-$

$\boldsymbol{T}_7$

	ε	0000	1110
$\boldsymbol{\varepsilon}$	$-$	$-$	$+$
$\boldsymbol{a}_1$	$+$	$-$	$+$
$\boldsymbol{a}_1\boldsymbol{a}_2$	$-$	$+$	$-$
$\boldsymbol{a}_0$	$-$	$-$	$-$
$\boldsymbol{a}_5$	$-$	$+$	$-$
$\boldsymbol{a}_0\boldsymbol{a}_7$	$-$	$-$	$+$
$\boldsymbol{a}_0\boldsymbol{a}_8$	$-$	$+$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_3$	$+$	$-$	$+$
$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_4$	$-$	$-$	$-$
$\boldsymbol{a}_1\boldsymbol{a}_2\boldsymbol{a}_6$	$+$	$-$	$+$

Fig. 6: Observation tables generated during the execution of the algorithm on Example 2.

Example 2. We show how the algorithm learns a target language over $\Sigma = \mathbb{B}^4$. All tables encountered during the execution of the algorithm are shown in Figure 6 and the decision trees appear in Figure 7 in the form of Karnaugh maps. The learner starts by initializing the observation table. Like any new state, initial state $\boldsymbol{\varepsilon}$ admits one outgoing transition that represents all concrete letters, that is $\boldsymbol{\Sigma}_{\boldsymbol{\varepsilon}} = \{\boldsymbol{a}_0\}$ and $[\![\boldsymbol{a}_0]\!] = \Sigma$. A set of concrete letters is sampled and is used as the evidence for the new symbol, $\mu(\boldsymbol{a}_0) = \{(0000), (0010), (1011), (1000), (1101)\}$, while $\hat{\mu}(\boldsymbol{a}_0) = (0000)$ is chosen as a representative. At this point, the observation table is $\boldsymbol{T}_0$ and the decision tree is ψ^0_ε, consisting of a single node.

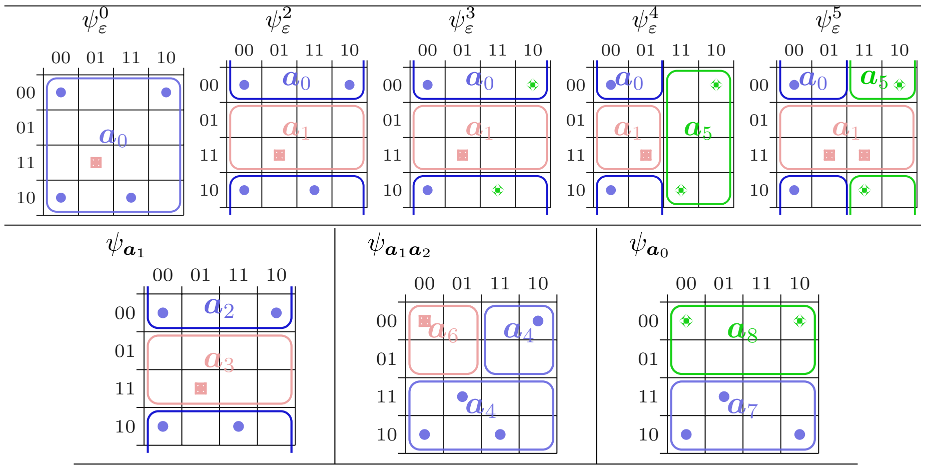


Fig. 7: Semantics functions used in Example 2. We show the evolution of $\psi_{\boldsymbol{\epsilon}}$ over time, while for the other states we show only the final partition. We use symbols such as {●, ■, ◆} to indicate different residuals.

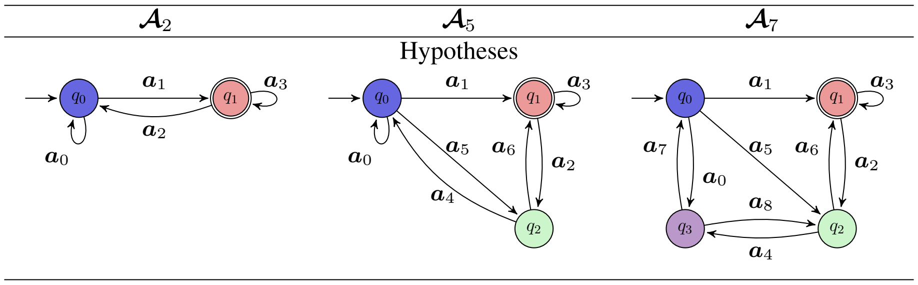


Fig. 8: Intermediate and final conjectured automata for Example 2.

The observation table is not evidence compatible because evidence $(1101) \in \mu(\boldsymbol{a}_0)$ behaves differently, and thus the partition needs refinement. The tree induction algorithm CART, which is used throughout this example, finds that Σ is best split into two blocks based on the values of variable x_2. That is, all letters for which $x_2 = 0$ are mapped to $\boldsymbol{a}_0$ while the others are mapped to a new symbol $\boldsymbol{a}_1$, added to $\boldsymbol{\Sigma}_{\boldsymbol{\epsilon}}$ (see ψ^2_ε). The resulting observation table $\boldsymbol{T}_1$ is made closed by letting $\boldsymbol{a}_1$ be a state. The evidence for the new state/symbol is sampled and, after resolving evidence incompatibility for it, we obtain the table $\boldsymbol{T}_2$ with $\psi = \{\psi^2_{\boldsymbol{\varepsilon}}, \psi_{\boldsymbol{a}_1}\}$.

The first conjectured automaton $\boldsymbol{\mathcal{A}}_2$, shown in Figure 8, is tested. A counter-example $w = (1010) \cdot (0000)$ is found and the learner applies the breakpoint method which adds the distinguishing word (0000) to E. The table is filled in by posing MQ's, resulting in $\boldsymbol{T}_3$ which is neither closed nor evidence compatible.

The table is made closed with $\boldsymbol{a}_1\boldsymbol{a}_2$ becoming a state. The outgoing transitions are defined as before, resulting in table $\boldsymbol{T}_4$ and $\psi_{\boldsymbol{a}_1\boldsymbol{a}_2}$. The added suffix (0000) causes an evidence incompatibility at $\boldsymbol{\epsilon}$ by changing the residual functions of the evidence, see $\psi_{\boldsymbol{\varepsilon}}^3$. The decision tree is reconstructed from scratch to become compatible with the updated sample. Then the symbols are rearranged so as to match the residuals of their representatives and a new symbol $\boldsymbol{a}_5$ is added. The partition is updated to the evidence compatible $\psi_{\boldsymbol{\varepsilon}}^4$ and the corresponding observation table is $\boldsymbol{T}_5$.

The new hypothesis $\boldsymbol{\mathcal{A}}_5$ is tested for equivalence, providing the counter-example (1111). The breakpoint method adds (1111) to $\mu(\boldsymbol{a}_5)$ as a new evidence, causing once more an incompatibility at the initial state, which is fixed by updating the tree $\psi_{\boldsymbol{\varepsilon}}$ into $\psi_{\boldsymbol{\varepsilon}}^5$. Since this incompatibility is due to a new evidence, the tree is updated using an incremental algorithm. Observe that this last counter-example only fixes the partition by rearranging the sub-cubes of $\mathbb{B}^4$ without adding any new transition.

A counter-example to the next hypothesis $\boldsymbol{\mathcal{A}}_6$ is $w = (1000) \cdot (1000) \cdot (0000) \cdot (0000) \cdot (1110)$, which adds the new suffix (1110) to E. The prefix $\boldsymbol{a}_0$ is now identified as a state and after refining $\psi_{\boldsymbol{a}_0}$ to become evidence compatible, the observation table $\boldsymbol{T}_7$ is obtained. The new and last hypothesis $\boldsymbol{\mathcal{A}}_7$ is tested on 855 words with no counter-example. We can conclude (see next section) with 95% confidence that $\boldsymbol{\mathcal{A}}_7$ is correct with a maximum error of 1%. $\square$

6 Theoretical and Empirical Results

We assume a probability distribution D defined over Σ^* which is expressed via a density function when Σ is a sub-interval of $\mathbb{R}$. For any $L \subseteq \Sigma^*$ let $Pr_D(L)$ be the probability of L, obtained by summing up the probabilities of its elements or by integrating densities in the real-valued case. Let L be a target language and let $\boldsymbol{\mathcal{A}}$ be a conjectured automaton accepting the language $L_{\boldsymbol{\mathcal{A}}}$. The quality of $\boldsymbol{\mathcal{A}}$ is defined by the probability of error, that is, the probability of the symmetric difference between L and $L_{\boldsymbol{\mathcal{A}}}$: $d(L, L_{\boldsymbol{\mathcal{A}}}) = Pr_D(L \oplus L_{\boldsymbol{\mathcal{A}}})$.

Definition 5 (PAC Learning [29]). *A learning algorithm learns a language L in a probably-approximately correct (PAC) manner with probability parameters ϵ (accuracy) and δ (confidence) if its output $\boldsymbol{\mathcal{A}}$ satisfies $Pr(d(L, L_{\boldsymbol{\mathcal{A}}}) \leq \epsilon) \geq 1 - \delta$.*

Given that our algorithm implements equivalence checks by comparing membership in L and in $L_{\boldsymbol{\mathcal{A}}}$ for words randomly selected according to D, the following result from [1] applies in a straightforward way to the symbolic case.

Proposition 2. *The symbolic learning algorithm PAC-learns a language L if the i-th equivalence query tests $r_i = \frac{1}{\epsilon}(\ln \frac{1}{\delta} + (i+1) \ln 2)$ random words without finding a counter-example.*

A class of functions or sets is *efficiently* PAC learnable if there is an algorithm that PAC learns it in time (and number of queries) polynomial in $1/\epsilon$, $1/\delta$, and in the size parameters of the learned object. For a target language $L \subset \Sigma^*$, the size is based on the minimal symbolic automaton $\boldsymbol{\mathcal{A}}$ recognizing L which is assumed to have n states and at most m outgoing transitions from every state.

Concerning the size of the observation table and the sample used to learn L, the set of prefixes $\boldsymbol{S}$ is monotonically increasing and reaches the size of at most n elements.

Since the table, by construction, is always kept 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, that play a distinguishing role for the states of the automaton, range between $\log_2 n$ and n. The size of the table ranges between $(n+m)\log_2 n$ and $n(mn+1)$. The size of the symbolic sample follows the size of prefixes and boundary which is at most $\mathcal{O}(mn^2)$, while the concrete sample depends on the number of evidences used in the table and its size is $\sum_{\boldsymbol{s}\in\boldsymbol{S}} |\mu_{\boldsymbol{s}}| \cdot |E|$.

A counter-example improves a hypothesis either by expanding the automaton, discovering a new state or transition, or by modifying the boundaries of already existing transitions. At most $n-1$ counter-examples discover new states and at most $n(m-1)$ introduce new transitions, resulting in at most $\mathcal{O}(mn)$ equivalence queries and counter-examples of this kind. The number of counter-examples that only change the boundaries in a partition is bounded in a probabilistic setting of approximate learning. The probability of finding a non-expansive counter-example ultimately decreases converging to zero. Hence, there exists a hypothesis i for which after r_i tests no counter-example is returned. From this we can conclude that our algorithm terminates, resulting in a symbolic automaton which is a PAC representation of the target language L.

Proposition 3. *The symbolic learning algorithm terminates with probability* 1 *returning a symbolic automaton that is a PAC acceptor of the target language* L*.*

Algorithm 1 and all procedures that appear in the present paper have been implemented in Python. In particular, methods $sample(\Sigma, k)$ in Procedure 3 returns a sample of size k chosen uniformly from Σ. Likewise, method $select(\cdot)$ uses a uniform distribution over the set of evidences to choose one representative. The $split$ method, used in Procedure 5, returns the middle point of the interval. For the case of Boolean alphabets the UPDATE method in Procedure 7 uses the CART algorithm [8] with information gain as the purity measure.

For theoretical results, it is sufficient that the same distribution is assumed for the random queries and the error estimation. For the implementation of random queries and for the empirical evaluation we have to be more concrete. The distribution D that we use is a composition of two distributions: a log-normal distribution, used to select the length of the word, and a uniform distribution over the alphabet, used to choose a letter at each position in the word. The log-normal distribution is chosen so that shorter words are preferred over longer ones.

Once an automaton $\boldsymbol{\mathcal{A}}$ has been learned by our algorithm, its quality can be evaluated as follows. When we have an explicit description of the automaton for the target language L, we can build its product with $\boldsymbol{\mathcal{A}}$ to accept the symmetric difference $L' = L \oplus L_{\boldsymbol{\mathcal{A}}}$. Then for any given k, using techniques similar to the volume computation applied by [2] to timed automata, we can compute the relative volume $|L' \cap \Sigma^k|/|\Sigma|^k$ which gives the probability of error over words of length k. Since the probability becomes negligible beyond some k, this is sufficient to obtain a good approximation of the error. Note that we can use volume because we assume a uniform distribution over Σ. Other distributions can be handled by more complex integration. It is worth mentioning the result of [11] concerning the influence of noise on automata which states that for certain types of automata, even a very small difference in the transition probabilities between a pair of automata may lead to a divergence in their long run behavior as $k \to \infty$. The use of a log-normal distributions protects our evaluation from this effect. An alternative way to evaluate the quality of the approximation, which can be applied also when the target language is represented as a black box, is just to draw words according to D and compare their classification by the target and learned languages.

We have compared our algorithm with three non-symbolic algorithms due to [1], [22] and [26] using the same oracle for membership and equivalence queries. All algorithms were tested on the same target languages defined over a numerical input alphabet Σ, intersected with $\mathbb{N}$ to allow the concrete enumerative algorithms to run as well. We evaluated the behavior of the algorithm in two ways. The first was to keep the structure of the automaton fixed while increasing the size of the alphabet. The second kept the alphabet size fixed and varied the number of states in the (randomly generated) target automaton. Naturally, the symbolic algorithm admits the most modest growth in the total number of membership queries including queries used for testing, in both evaluation scenarios. Not surprisingly, it generates more hypotheses and testing queries, and obtains more counter-examples. Similar results were observed in a case study where the target languages were sets of valid passwords. Here too, the symbolic algorithm required less MQ's on average than any other method in all types of passwords, and the difference increases as the passwords rules become more complicated and the automata require more states and transitions. It is remarkable, however, that the symbolic algorithm managed to discover more states in general. More experimental results will be reported elsewhere.

7 Conclusions and Future Work

We presented an algorithmic scheme for learning languages over large alphabets. The algorithm targets languages acceptable by symbolic automata with a modest number of states and transitions, guarded by simple constraints on the alphabet, which can be arbitrarily large. The new algorithm replaces the helpful teacher of the L^* algorithm by random testing and is thus applicable to more realistic settings. The price of this modification is in the probabilistic relaxation of the correctness criterion and in a more general procedure for handling counter-examples and refining the alphabet partitions. This generality pays off as attested by the easy adaptation of the algorithm to the Boolean domain.

Concerning *related work*, ideas similar to ours have been suggested and explored in a series of papers [4,16,17] that also adapt automaton learning to large alphabets. While some design decisions are similar, for example, to use distinct symbolic alphabets at every state [17], our approach is more rigorous in the way it treats evidence incompatibility and the modification of partition boundaries. We do not consider each modification as a partition refinement, but rather try first just to modify the boundaries without adding a new symbol. As a result, we have the following property whenever we conclude the treatment of evidence incompatibility: the mapping of concrete letters to symbols is always sample-compatible and is well-defined for the whole alphabet, which does not seem to be the case for the scheme presented in [4], which has the potential of generating new symbols indefinitely, or the case in [16,17], which results in a partially-defined hypothesis. Recently, new results presented in [12] in the context of learning symbolic automata give a more general justification for a learning scheme like ours by proving that learnability is closed under product and disjoint union.

Our work on abstract automata should not be confused with work dealing with register automata, another extension of automata to infinite alphabets [18,3,15]. These are automata augmented with additional variables that can store some input letters and newly-read letters. Newly-read letters can be compared with the registers but typically not with constants in the domain. Such automata can express, for example, the requirement that the password at login is the same as the password at sign-up. In the most recent work on learning register automata [9], a strong *tree oracle* is used. Given a concrete prefix and a symbolic prefix, the teacher returns a special type of a register automaton that has a tree structure. This fills in the entries of the observation table and provides the information about the registers and guards in the automaton. This algorithm is efficient only in the presence of shortest counter-examples and, in addition, when applied on a theory of inequalities and extended to use constants, these constants should be known in advance.

We believe that our comprehensive framework for learning languages over large alphabets is unique in employing *all* the following features:

1. It is based on a clean and general definition of the relation between the concrete and symbolic alphabets;
2. It can work without a helpful teacher and replace its counter-examples by random sampling, resulting in counter-examples which are not assumed to be minimal (neither in length nor in lexicographic order);
3. It employs an adaptation of the breakpoint method to analyze in an efficient way the information provided by counter-examples;
4. It treats the modification of alphabet partitions in a rigorous way which guarantees that no superfluous symbols are introduced;
5. It is modular, separating the general aspects from those that are alphabet specific, thus providing for a relatively easy adaptation to new alphabets.

A natural future extension of the algorithm is to consider alphabets which are subsets of $\mathbb{N}^n$ and $\mathbb{R}^n$. Preliminary work in this direction has been reported in [23] but used a very restricted type of monotone partitions in order to keep the notion of a minimal counter-example meaningful. Now that we are not restricted to such counter-examples we can use more general partitions, represented by regression trees, a generalization of decision trees to numerical domains.

We are currently conducting more experiments to assess the scalability of our algorithms, mostly in the Boolean domain. These are mostly synthetic examples which are intended to confirm the sensitivity of the algorithm to the complexity of the partitions (the number of blocks and the number of variables that are involved on their definition, rather on the total number of variables which determines the alphabet size. Once the scalability issue is resolved, it remains to find a convincing class of real-world applications that benefits from such algorithms. In the numerical domain we are rather convinced in the existence of mechanisms, say, in cellular information processing in

Biology [7] where discrete transitions are taken based on threshold crossings of continuous variables without remembering their values. Likewise, in the Boolean domain, we have to find applications in the specification of large complex systems with many components (digital circuit, distributed multi-agent systems). Hopefully such specifications could be expressible by symbolic automata where the complexity can be confined to the alphabet partitions and need not proliferate into states and cause explosion.

References

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
2. Eugene Asarin, Nicolas Basset, and Aldric Degorre. Entropy of regular timed languages. *Information and Computation*, 241:142 – 176, 2015.
3. Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In *AMW*, volume 619 of *CEUR Workshop Proceedings*, 2010.
4. 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.
5. Therese Berg and Harald Raffelt. Model checking. In *Model-Based Testing of Reactive Systems*, volume 3472 of *LNCS*, pages 557–603. Springer, 2004.
6. Matko Botinčan and Domagoj Babić. Sigma*: Symbolic learning of Input-Output specifications. In *POPL*, pages 443–456. ACM, 2013.
7. Dennis Bray. *Wetware: a computer in every living cell*. Yale University Press, 2009.
8. Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.
9. Sofia Cassel, Falk Howar, Bengt Jonsson, and Bernhard Steffen. Active learning for extended finite state machines. *Formal Aspects of Computing*, 28(2):233–263, 2016.
10. Loris D'Antoni and Margus Veanes. Minimization of symbolic automata. In *POPL*, pages 541–554. ACM, 2014.
11. Bernard Delyon and Oded Maler. On the effects of noise and speed on computations. *Theoretical Computer Science*, 129(2):279–291, 1994.
12. Samuel Drews and Loris DAntoni. Learning symbolic automata. In *TACAS*, pages 173–189. Springer, 2017.
13. E. Mark Gold. System identification via state characterization. *Automatica*, 8(5):621–636, 1972.
14. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
15. Falk Howar, Bernhard Steffen, Bengt Jonsson, and Sofia Cassel. Inferring canonical register automata. In *VMCAI*, volume 7148 of *LNCS*, pages 251–266. Springer, 2012.
16. 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.
17. 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.
18. Michael Kaminski and Nissim Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.
19. Harry R Lewis and Christos H Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall PTR, 1997.
20. Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, pages 152–166, 2004.
21. Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of computer science*, pages 475–505. Springer, 2008.
22. Oded Maler and Amir Pnueli. On the learnability of infinitary regular sets. *Information and Computation*, 118(2):316–326, 1995.

23. Irini-Eleftheria Mens and Oded Maler. Learning regular languages over large ordered alphabets. *Logical Methods in Computer Science (LMCS)*, 11(3), 2015.
24. Anil Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.
25. J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.
26. Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.
27. Michael Sipser. *Introduction to the Theory of Computation*. PWS, Boston, 1997.
28. Paul E Utgoff. Incremental induction of decision trees. *Machine learning*, 4(2):161–186, 1989.
29. Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.
30. Gertjan Van Noord and Dale Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.
31. Margus Veanes. Applications of symbolic finite automata. In *International Conference on Implementation and Application of Automata*, pages 16–23. Springer, 2013.
32. Margus Veanes, Nikolaj Bjørner, and Leonardo De Moura. Symbolic automata constraint solving. In *LPAR*, pages 640–654. Springer, 2010.
33. 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.