Learning Realtime One-Counter Automata[⋆]

Véronique Bruyère[1], Guillermo A. Pérez[2], and Gaëtan Staquet[1,2](✉)

[1] University of Mons, Mons, Belgium
{veronique.bruyere,gaetan.staquet}@umons.ac.be
[2] University of Antwerp – Flanders Make, Antwerp, Belgium
guillermoalberto.perez@uantwerpen.be

Abstract. We present a new learning algorithm for realtime one-counter automata. Our algorithm uses membership and equivalence queries as in Angluin's L^* algorithm, as well as counter value queries and partial equivalence queries. In a partial equivalence query, we ask the teacher whether the language of a given finite-state automaton coincides with a counter-bounded subset of the target language. We evaluate an implementation of our algorithm on a number of random benchmarks and on a use case regarding efficient JSON-stream validation.

Keywords: Realtime one-counter automata · Active learning

1 Introduction

In *active learning*, a *learner* has to infer a model of an unknown machine by interacting with a *teacher*. Angluin's seminal L^* algorithm does precisely this for finite-state automata while using only *membership* and *equivalence queries* [4]. An important application of active learning is to learn black-box models from (legacy) software and hardware systems [17,28]. Though recent works have greatly advanced the state of the art in finite-state automata learning, handling real-world applications usually involves tailor-made abstractions to circumvent elements of the system which result in an infinite state space [1]. This highlights the need for learning algorithms that focus on more expressive models.

One-counter automata (OCAs) are obtained by extending finite-state automata with an integer-valued variable that can be increased, decreased, and tested for equality against zero. The counter allows OCAs to capture the behavior of some infinite-state systems. Additionally, their expressiveness has been shown sufficient to verify programs with lists [10] and validate XML streams [13]. To the best of our knowledge, there is no learning algorithm for general OCAs.

For *visibly* OCAs (that is, when the alphabet is such that letters determine whether the counter is decreased, increased, or not affected), Neider and Löding describe an algorithm in [27]. Besides the usual membership and equivalence queries, they use *partial equivalence queries*: given a finite-state automaton $\mathcal{A}$

⋆ This work was partially supported by the FWO "SAILor" project (G030020N). Gaëtan Staquet is a research fellow (Aspirant) of the F.R.S– FNRS.

and a bound k, does the language of $\mathcal{A}$ correspond to the k-bounded subset of the target language? Additionally, Fahmy and Roos [15] claim to have solved the case of realtime OCA (i.e., when the automaton is assumed to be configuration-deterministic and no ε-transitions are allowed). However, we were unable to understand the algorithm and proofs in that paper due to lack of precise formalization and detailed proofs. We also found an example where the provided algorithm did not produce the expected results. It is noteworthy that Böhm et al. [8] made similar remarks about related works of Roos [6,30].

Our contribution. We present a new learning algorithm for realtime one-counter automata (ROCAs). Our algorithm uses membership, equivalence and partial equivalence queries. It also makes use of *counter value queries*. That is, we make the assumption that we have an executable black box with observable counter values. We prove that our algorithm runs in exponential time and space and that it uses at most an exponential number of queries. Due to lack of space, some proofs have been omitted. We refer the interested reader to the full technical report of this work [12].

In [9], Bollig establishes a connection between OCAs with counter-value observability and visibly OCAs. We expose a similar connection and are thus able to leverage Neider and Löding's learning algorithm for visibly one-counter languages [27] as a sort of sub-routine for ours. Nevertheless, our learning algorithm is more sophisticated due to the fact that the counter values cannot be inferred from a given word. Technically, the latter required us to extend the classical definition of *observation tables* as used in, e.g., [4,27]. Entries in our tables are composed of Boolean language information as well as a counter value or a *wildcard* encoding the fact that we do not (yet) care about the value of the corresponding word. (Our use of wildcards is reminiscent of the work [25] on learning a regular language from an "inexperienced" teacher.) Moreover our tables need two sets of suffixes while only one is necessary in classical tables. Indeed we provide an example showing that making a table closed and consistent leads to an infinite loop when it has only one set of suffixes. Due to these extensions, much work is required to prove that it is always possible to make a table closed and consistent in finite time. Finally, we formulate queries for the teacher in a way which ensures the observation table eventually induces a right congruence refining the classical Myhill-Nerode congruence with counter-value information. Our computations and experiments show that the second set of suffixes is exponential leading to an exponential algorithm (instead of polynomial as in [27]).

We evaluate an implementation of our algorithm on random benchmarks and a use case inspired by [13]. Namely, we learn an ROCA model for a simple JSON schema validator — i.e., a program that verifies whether a JSON document satisfies a given JSON schema. The advantage of having a finite-state model of such a validator is that JSON-stream validation becomes trivially efficient (cf. automata-based parsing [3]).

Related work. Our assumption about counter-value observability means that the system with which we interact is a *gray box*. Several recent works make such assumptions to learn complex languages or ameliorate query-usage bounds. For instance, in [7], the authors assume they have information about the target language L in the form of a superset of it. Similarly, in [2], the authors assume L is obtained as the composition of two languages, one of which they know in advance. In [26], the teacher is assumed to have an executable automaton representation of the (infinite-word) target language and that some properties of this automaton are visible to the learner. Finally, in [16] it is assumed that constraints satisfied along the run of a system can be made visible.

2 Preliminaries

In this section we recall all necessary notions. We give a definition of realtime one-counter automaton adapted from [15,34]. We present the concept of behavior graph of such automata, inspired by the one given in [27] for visibly one-counter automata (VCAs), and state some important properties for our learning task.

An *alphabet* Σ is a non-empty finite set of *symbols*. A *word* is a finite sequence of symbols from Σ, and the *empty word* is denoted by ε. The set of all words over Σ is denoted by Σ^*. The *concatenation* of two words $u, v \in \Sigma^*$ is denoted by uv. A *language* L is a subset of Σ^*. Given a word $w \in \Sigma^*$ and a language $L \subseteq \Sigma^*$, the *set of prefixes of* w is $\mathit{Pref}(w) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*, w = uv\}$ and the *set of prefixes of* L is $\mathit{Pref}(L) = \bigcup_{w \in L} \mathit{Pref}(w)$. Similarly, we have the sets of *suffixes* $\mathit{Suff}(w) = \{u \in \Sigma^* \mid \exists v \in \Sigma^*, w = vu\}$ and $\mathit{Suff}(L) = \bigcup_{w \in L} \mathit{Suff}(w)$. Moreover, L is said to be *prefix-closed* (resp. *suffix-closed*) if $L = \mathit{Pref}(L)$ (resp. $L = \mathit{Suff}(L)$). In this paper, we always work with *non-empty* languages L to avoid having to treat particular cases.

Definition 1. *A* realtime one-counter automaton (ROCA) $\mathcal{A}$ *is a tuple* $\mathcal{A} = (Q, \Sigma, \delta_{=0}, \delta_{>0}, q_0, F)$ *where: (1)* Σ *is an alphabet, (2)* Q *is a non-empty finite set of states, (3)* $q_0 \in Q$ *is the initial state, (4)* $F \subseteq Q$ *is the set of final states, and (5)* $\delta_{=0}$ *and* $\delta_{>0}$ *are two (total) transition functions defined as* $\delta_{=0} : Q \times \Sigma \to Q \times \{0, +1\}$ *and* $\delta_{>0} : Q \times \Sigma \to Q \times \{-1, 0, +1\}$.

The second component of the output of $\delta_{=0}$ and $\delta_{>0}$ gives the counter operation to apply when taking the transition. Notice that it is impossible to decrement the counter when it is already equal to zero.

A *configuration* is a pair $(q, n) \in Q \times \mathbb{N}$, that is, it contains the current state and the current counter value. The *transition relation* $\xrightarrow[\mathcal{A}]{} \subseteq (Q \times \mathbb{N}) \times \Sigma \times (Q \times \mathbb{N})$ contains $(q, n) \xrightarrow[\mathcal{A}]{a} (p, m)$ if and only $\begin{cases} \delta_{=0}(q, a) = (p, c) \land m = n + c & \text{if } n = 0, \\ \delta_{>0}(q, a) = (p, c) \land m = n + c & \text{if } n > 0. \end{cases}$
When the context is clear, we omit $\mathcal{A}$ to simply write $\xrightarrow{a}$. We lift the relation to words in the natural way. Notice that this relation is *deterministic* in the sense that given a configuration (q, n) and a word w, there exists a unique configuration (p, m) such that $(q, n) \xrightarrow{w} (p, m)$.

Given a word w, let $(q_0, 0) \xrightarrow{w} (q, n)$ be the *run* on w. When $n = 0$ and $q \in F$, we say that this run is *accepting*. The *language accepted* by $\mathcal{A}$ is the set

$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, 0) \xrightarrow{w} (q, 0) \text{ with } q \in F\}$. If a language L is accepted by some ROCA, we say that L is a *realtime one-counter language (ROCL).*

Given $w \in \Sigma^*$, we define the *counter value* of w according to $\mathcal{A}$, noted $c_{\mathcal{A}}(w)$, as the counter value n of the configuration (q, n) such that $(q_0, 0) \xrightarrow{w} (q, n)$. We define the *height of w according to $\mathcal{A}$*, noted $h_{\mathcal{A}}(w)$, as the maximal counter value among the prefixes of w, i.e., $h_{\mathcal{A}}(w) = \max_{x \in \mathit{Pref}(w)} c_{\mathcal{A}}(x)$.

We now introduce the concept of behavior graph of an ROCA $\mathcal{A}$, inspired from the one given for VCAs in [27]. It is a (possibly infinite) automaton based on the congruence relation $\equiv$ over Σ^* such that $u \equiv v$ if and only if for all $w \in \Sigma^*$, we have (1) $uw \in L \Leftrightarrow vw \in L$ and (2) $uw, vw \in \mathit{Pref}(L) \Rightarrow c_{\mathcal{A}}(uw) = c_{\mathcal{A}}(vw)$. The equivalence class of u is denoted by $[\![u]\!]_{\equiv}$. This relation $\equiv$ is a refinement of the Myhill-Nerode relation [18]. Its second condition depends on $\mathcal{A}$ and is limited to $\mathit{Pref}(L)$ because even if $\mathcal{A}$ has different counter values for words not in $\mathit{Pref}(L)$, we still require all those words to be equivalent.

Definition 2. *Let $\mathcal{A} = (Q, \Sigma, \delta_{=0}, \delta_{>0}, q_0, F)$ be an ROCA accepting $L \subseteq \Sigma^*$. The* behavior graph of $\mathcal{A}$ *is the automaton $BG(\mathcal{A}) = (Q_{\equiv}, \Sigma, \delta_{\equiv}, q^0_{\equiv}, F_{\equiv})$ where: (1) $Q_{\equiv} = \{[\![u]\!]_{\equiv} \mid u \in \mathit{Pref}(L)\}$ is the set of states, (2) $q^0_{\equiv} = [\![\varepsilon]\!]_{\equiv}$ is the initial state, (3) $F_{\equiv} = \{[\![u]\!]_{\equiv} \mid u \in L\}$ is the set of final states, (4) $\delta_{\equiv} : Q_{\equiv} \times \Sigma \to Q_{\equiv}$ is the transition function defined by: $\delta_{\equiv}([\![u]\!]_{\equiv}, a) = [\![ua]\!]_{\equiv}$, $\forall [\![u]\!]_{\equiv}, [\![ua]\!]_{\equiv} \in Q_{\equiv}$, $\forall a \in \Sigma$.*

Note that $Q_{\equiv} \neq \emptyset$ since $L \neq \emptyset$ by assumption. A straightforward induction shows that $BG(\mathcal{A})$ accepts $L = \mathcal{L}(\mathcal{A})$. By definition, BG_L is trim (each state is reachable and co-reachable) which implies that the transition function is partial.

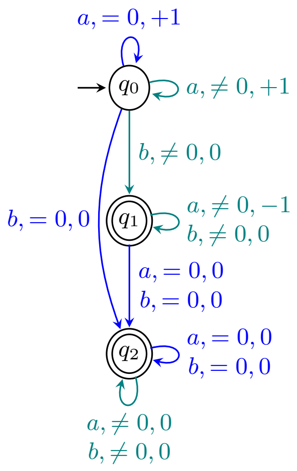


Fig. 1: An ROCA

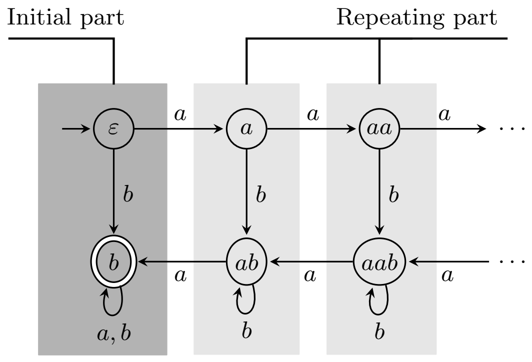


Fig. 2: Behavior graph of the left ROCA

Example 1. A 3-state ROCA $\mathcal{A}$ over $\Sigma = \{a, b\}$ is given in Figure 1. The initial state q_0 is marked by a small arrow and the two final states q_1 and q_2 are double-circled. The transitions give the input symbol, the condition on the counter value, and the counter operation, in this order ($\delta_{=0}$ is indicated in blue while $\delta_{>0}$ is indicated in green). The run on $w = aababaa$ is accepting since it ends with the configuration $(q_2, 0)$. Moreover, $c_{\mathcal{A}}(w) = 0$ and $h_{\mathcal{A}}(w) = 2$. One can verify that $\mathcal{L}(\mathcal{A}) = \{w \in \{a,b\}^* \mid \exists n \geq 0, \exists k_1, \dots, k_n \geq 0, \exists u \in \{a,b\}^*, w = a^n b(b^{k_1}a \cdots b^{k_n}a)u\}$. The behavior graph $BG(\mathcal{A})$ of $\mathcal{A}$ is given in Figure 2. One can check that $b \equiv abba$. Indeed $\forall w \in \Sigma^*, bw \in L \Leftrightarrow abbaw \in L$. Moreover, $\forall w \in \Sigma^*$ such that $bw, abbaw \in \mathit{Pref}(L)$, we have $c_{\mathcal{A}}(bw) = c_{\mathcal{A}}(abbaw)$. However, $ab \not\equiv aab$ since $ab, aab \in \mathit{Pref}(L)$ but $c_{\mathcal{A}}(ab) = 1 \neq c_{\mathcal{A}}(aab) = 2$. □

We finally state two important properties of the behavior graph, useful for the learning of ROCAs. We first establish that $BG(\mathcal{A})$ has a finite representation that relies on the fact that it has an ultimately periodic structure (see Figure 2). Let us introduce some notations. By definition of the states of $BG(\mathcal{A})$, all words in the same class $[\![u]\!]_{\equiv}$ have the same counter value. We thus define the *level* ℓ of $BG(\mathcal{A})$ as the set of states with counter value ℓ. One can easily check that each level has a number of states bounded by $|Q|$. The minimal such bound is called the *width* of BG_L and is denoted by K. This observation allows to enumerate the states in level ℓ using a mapping $\nu_\ell : \{[\![w]\!]_{\equiv} \in Q_{\equiv} \mid c_{\mathcal{A}}(w) = \ell\} \to \{1, \dots, K\}$. Using these enumerations ν_ℓ, $\ell \in \mathbb{N}$, we can encode the transitions of $BG(\mathcal{A})$ as a sequence of mappings $\tau_\ell : \{1, \dots, K\} \times \Sigma \to \{1, \dots, K\} \times \{-1, 0, +1\}$, with $\ell \in \mathbb{N}$, as follows. For all $i \in \{1, \dots, K\}$, $a \in \Sigma$, the mapping τ_ℓ is defined as:

$$\tau_\ell(i, a) = \begin{cases} (j, c) & \text{if } \exists [\![u]\!]_{\equiv}, [\![ua]\!]_{\equiv} \in Q_{\equiv} \text{ such that } c_{\mathcal{A}}(u) = \ell, \\ & c_{\mathcal{A}}(ua) = \ell + c, \nu_\ell([\![u]\!]_{\equiv}) = i, \nu_{\ell+c}([\![ua]\!]_{\equiv}) = j, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In this way, the behavior graph can be encoded as the sequence $\alpha = \tau_0\tau_1\tau_2\dots$, called a *description* of $BG(\mathcal{A})$. The following theorem states that there always exists such a description which is *periodic* (see again Figure 2).

Theorem 1. *Let $\mathcal{A}$ be an ROCA, $BG(\mathcal{A}) = (Q_{\equiv}, \Sigma, \delta_{\equiv}, q^0_{\equiv}, F_{\equiv})$ be the behavior graph of $\mathcal{A}$, and K be the width of $BG(\mathcal{A})$. Then, there exists a sequence of enumerations $\nu_\ell : \{[\![u]\!]_{\equiv} \in Q_{\equiv} \mid c_{\mathcal{A}}(u) = \ell\} \to \{1, \dots, K\}$ such that the corresponding description α of $BG(\mathcal{A})$ is an ultimately periodic word with offset $m > 0$ and period $k \geq 0$, i..e, $\alpha = \tau_0 \dots \tau_{m-1}(\tau_m \dots \tau_{m+k-1})^\omega$.*

This theorem is the counterpart of a similar theorem given in [27] for VCAs. We get this theorem thanks to an isomorphism between the behavior graph of an ROCA $\mathcal{A}$ and that of a suitable VCA constructed from $\mathcal{A}$.

The second major property states that from a periodic description of a behavior graph $BG(\mathcal{A})$, one can construct an ROCA that accepts the same language.

Proposition 1. *Let $\mathcal{A}$ be an ROCA accepting a language $L \subseteq \Sigma^*$, $BG(\mathcal{A})$ be its behavior graph of width K, $\alpha = \tau_0 \dots \tau_{m-1}(\tau_m \dots \tau_{m+k-1})^\omega$ be a periodic description of $BG(\mathcal{A})$ with offset m and period k. Then, from α, one can construct an ROCA $\mathcal{A}_\alpha$ accepting L such that the size of $\mathcal{A}_\alpha$ is polynomial in m, k and K.*

3 Learning ROCAs

The aim of this paper is to design an ROCA-learning algorithm. We suppose that the reader is familiar with the concept of *active learning*, and in particular with Angluin's seminal L^* algorithm for learning finite-state automata (DFAs) [4]. In this section, let us fix a language $L \subseteq \Sigma^*$ and an ROCA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = L$. We here explain how a *learner* will learn L by querying a *teacher*. Our learning algorithm is inspired by the one designed in [27] for VCAs. The idea is to learn an initial fragment of the behavior graph $BG(\mathcal{A})$ up to a fixed counter limit ℓ, to extract every possible periodic description from the fragment, and to construct an ROCA from each of these descriptions. If we find one ROCA accepting L, we are done. Otherwise, we increase ℓ and repeat the process.

Formally, the initial fragment up to ℓ is called the *limited behavior graph* $BG_\ell(\mathcal{A})$. This is the subgraph of $BG(\mathcal{A})$ whose set of states is $\{[\![w]\!]_\equiv \in Q_\equiv \mid h_\mathcal{A}(w) \leq \ell\}$. This DFA accepts the language $L_\ell = \{w \in L \mid \forall x \in \mathit{Pref}(w), 0 \leq c_\mathcal{A}(x) \leq \ell\}$. Notice that $BG_\ell(\mathcal{A})$ is composed of the first $\ell + 1$ levels of $BG(\mathcal{A})$ (from 0 to ℓ) such that each level is restricted to states $[\![w]\!]_\equiv$ with $h_\mathcal{A}(w) \leq \ell$.

During the learning process, the teacher has to answer four different types of queries asked by the learner: (1) *membership query* (does $w \in \Sigma^*$ belong to L?), (2) *counter value query* (given $w \in \mathit{Pref}(L)$, what is $c_\mathcal{A}(w)$?), (3) *partial equivalence query* (does the DFA $\mathcal{B}$ accept L_ℓ?), and (4) *equivalence query* (does the ROCA $\mathcal{B}$ accept L?). In case of negative answer to a (partial) equivalence query, the teacher provides a counterexample $w \in \Sigma^*$ witness of this non-equivalence.

Recall that membership and equivalence queries are used in the L^* algorithm [4]. Additionally, partial equivalence queries are required in the VCA-learning algorithm of [27] to find the basis of a periodic description for the target automaton. However counter-value queries are not necessary because VCAs use a pushdown alphabet and the counter value can be directly inferred from the word. For general alphabets, this is no longer possible and the learner has to ask the teacher for this information. Our main result is the following theorem.

Theorem 2. *Let $\mathcal{A}$ be an ROCA accepting a language $L \subseteq \Sigma^*$. Given a teacher for L, which answers membership, counter value, and (partial) equivalence queries, an ROCA accepting L can be computed in time and space exponential in $|Q|, |\Sigma|$ and t, with Q the set of states of $\mathcal{A}$ and t the length of the longest counterexample returned by the teacher on (partial) equivalence queries. The learner asks $\mathcal{O}(t^3)$ partial equivalence queries, $\mathcal{O}(|Q|t^2)$ equivalence queries and a number of membership and counter value queries exponential in $|Q|, |\Sigma|$ and t.*

In what follows we describe the main steps of our learning algorithm. Given a counter limit ℓ, we first introduce the kind of observation table $\mathscr{O}_\ell$ we use to store the learned information about L_ℓ. Secondly, we explain what are the constraints imposed to $\mathscr{O}_\ell$ to derive a DFA $\mathcal{A}_{\mathscr{O}_\ell}$, candidate for accepting L_ℓ. Thirdly, when a counterexample is returned by the teacher to a partial equivalence query with this DFA, we explain how to update the table. Fourthly, we give the whole learning algorithm such that when $\mathcal{A}_{\mathscr{O}_\ell}$ accepts L_ℓ with ℓ big enough, a periodic description α is finally extracted from $\mathcal{A}_{\mathscr{O}_\ell}$ such that the ROCA $\mathcal{A}_\alpha$ accepts L.

Observation Table and Approximation Sets. As for learning DFAs and VCAs, we use an observation table to store the data gathered during the learning process. This table aims at approximating the equivalence classes of $\equiv$ and therefore stores information about both membership to L and counter values for words known to be in $Pref(L)$. It depends on a counter limit $\ell \in \mathbb{N}$ since we first want to learn $BG_\ell(\mathcal{A})$. We highlight the fact that our table uses two sets of suffixes: $\widehat{S}$ and S (contrarily to the algorithms of [4,27] that use only one set S). Intuitively, we use the set $\widehat{S}$ to store membership information and the set S for counter value information. In [27], the set S is not needed as the counter value of a word can be immediately derived from the word. This is not the case for us, as the teacher's ROCA is required to compute the counter value of a word. Therefore, we need to explicitly store that information.

Definition 3. *Let $\ell \in \mathbb{N}$ be a counter limit and L_ℓ be the language accepted by the limited behavior graph $BG_\ell(\mathcal{A})$ of an ROCA $\mathcal{A}$. An* observation table $\mathscr{O}_\ell$ *up to ℓ is a tuple $(R, S, \widehat{S}, \mathcal{L}_\ell, \mathcal{C}_\ell)$ with: (1) a finite prefix-closed set $R \subseteq \Sigma^*$ of* representatives*, (2) two finite suffix-closed sets $S, \widehat{S}$ of* separators *such that $S \subseteq \widehat{S} \subseteq \Sigma^*$, (3) a function $\mathcal{L}_\ell : (R \cup R\Sigma)\widehat{S} \to \{0, 1\}$, (4) a function $\mathcal{C}_\ell : (R \cup R\Sigma)S \to \{\bot, 0, \dots, \ell\}$.*

Let $Pref(\mathscr{O}_\ell)$ be the set $\{w \in Pref(us) \mid u \in R \cup R\Sigma, s \in \widehat{S}, \mathcal{L}_\ell(us) = 1\}$. Then for all $u \in R \cup R\Sigma$ the following holds:

- *for all $s \in \widehat{S}$, $\mathcal{L}_\ell(us)$ is 1 if $us \in L_\ell$ and 0 otherwise,*
- *for all $s \in S$, $\mathcal{C}_\ell(us)$ is $c_\mathcal{A}(us)$ if $us \in Pref(\mathscr{O}_\ell)$ and $\bot$ otherwise.*

In this definition, the domains of $\mathcal{L}_\ell$ and $\mathcal{C}_\ell$ are different as already mentioned. Notice that $Pref(\mathscr{O}_\ell) \subseteq Pref(L_\ell)$. To compute the values of the table $\mathscr{O}_\ell$, the learner proceeds by asking membership and counter value queries to the teacher.

	ε	a	ba
ε	$0,0$	0	1
a	$0,1$	0	1
ab	$0,1$	1	1
aba	$1,0$	1	1
aa	$0,\bot$	0	0
b	$1,0$	1	1
abb	$0,1$	1	1
$abaa$	$1,0$	1	1
$abab$	$1,0$	1	1
aaa	$0,\bot$	0	0
aab	$0,\bot$	0	0

Fig. 3: An observation table.

	ε
ε	$0,0$
a	$0,1$
ab	$0,1$
aba	$1,0$
b	$1,0$
aa	$0,\bot$
abb	$0,\bot$
$abaa$	$1,0$
$abab$	$1,0$

	ε
ε	$0,0$
a	$0,1$
ab	$0,1$
aba	$1,0$
abb	$0,1$
b	$1,0$
aa	$0,\bot$
$abaa$	$1,0$
$abab$	$1,0$
$abba$	$1,0$
$abbb$	$0,\bot$

	ε
ε	$0,0$
a	$0,1$
ab	$0,1$
aba	$1,0$
abb	$0,1$
$abbb$	$0,1$
b	$1,0$
aa	$0,\bot$
$abaa$	$1,0$
$abab$	$1,0$
$abba$	$1,0$
$abbba$	$1,0$
$abbbb$	$0,\bot$

Fig. 4: Observation tables exposing an infinite loop when using the L^* algorithm.

Example 2. In this example, we give an observation table $\mathscr{O}_\ell$ for the ROCA $\mathcal{A}$ from Figure 1 and the counter limit $\ell = 1$, see Figure 3. Hence we want to learn $BG_\ell(\mathcal{A})$ whose set of states is given by the first two levels from Figure 2.

The first column of $\mathscr{O}_\ell$ contains the elements of $R \cup R\Sigma$ such that the upper part is constituted by $R = \{\varepsilon, a, ab, aba, aa\}$ and the lower part by $R\Sigma \setminus R$. The first row contains the elements of $\widehat{S}$ such that the left part is constituted by $S = \{\varepsilon\}$ and the right part by $\widehat{S} \setminus S$. For each element $us \in (R \cup R\Sigma)S$, we store the two values $\mathcal{L}_\ell(us)$ and $\mathcal{C}_\ell(us)$ in the intersection of row u and column s. For instance, these values are equal to $0, \perp$ for $u = aa$ and $s = \varepsilon$. For each element $us \in (R \cup R\Sigma)(\widehat{S} \setminus S)$, we have only one value $\mathcal{L}_\ell(us)$ to store. Note that $\mathit{Pref}(\mathscr{O}_\ell)$ is a proper subset of $\mathit{Pref}(L_\ell)$. For instance, $aababaa \in \mathit{Pref}(L_\ell) \setminus \mathit{Pref}(\mathscr{O}_\ell)$.

Let us now explain why it is necessary to use the additional set $\widehat{S}$ in Definition 3. Assume that we only use the set S and that the current observation table is the leftmost table $\mathscr{O}_\ell$, with $\ell = 1$, given in Figure 4 for the ROCA from Figure 1. On top of that, assume we are using the classical L^* algorithm [4]. As we can see, the table is not closed since the stored information for abb, that is, $0, \perp$, does not appear in the upper part of the table for any $u \in R$. So, we add abb in this upper part and $abba$ and $abbb$ in the lower part, to obtain the second table of Figure 4. Notice that this shift of abb has changed its stored information, which is now equal to $0, 1$. Indeed the set $\mathit{Pref}(\mathscr{O}_\ell)$ now contains abb as a prefix of $abba \in L_\ell$. Again, the new table is not closed because of $abbb$. After shifting this word in the upper part of the table, we obtain the third table of Figure 4. It is still not closed due to $abbbb$. This process will continue ad infinitum. □

To avoid an infinite loop when making the table closed, as described in the previous example, we modify both the concept of table and how to derive an equivalence relation from that table. Our solution is to introduce the set $\widehat{S}$, as already explained, but also the concept of approximation set to approximate $\equiv$.

Definition 4. *Let $\mathscr{O}_\ell = (R, S, \widehat{S}, \mathcal{L}_\ell, \mathcal{C}_\ell)$ be an observation table up to ℓ. Let $u, v \in R \cup R\Sigma$. Then, $u \in \mathit{Approx}(v)$ if and only if for all $s \in S$, we have $\mathcal{L}_\ell(us) = \mathcal{L}_\ell(vs)$ and $\mathcal{C}_\ell(us) \neq \perp \wedge \mathcal{C}_\ell(vs) \neq \perp \Rightarrow \mathcal{C}_\ell(us) = \mathcal{C}_\ell(vs)$. The set $\mathit{Approx}(v)$ is called an* approximation set.

In this definition, note that we consider $\perp$ values as wildcards and we focus on words with suffixes from S only (and not from $\widehat{S} \setminus S$). Interestingly, such wildcard entries in observation tables also feature in learning from an "inexperienced" teacher [25]. Just like in that work, a crucial part of our learning algorithm concerns how to obtain an equivalence relation from such an observation table (note that Approx does not define an equivalence relation as it is not transitive).

Example 3. Let $\mathscr{O}_\ell$ be the table from Figure 3. Let us compute $\mathit{Approx}(\varepsilon)$ (recall that we only consider $S = \{\varepsilon\}$ and not $\widehat{S} = \{a, ba\}$). We can see that $aba \notin \mathit{Approx}(\varepsilon)$ as $\mathcal{L}_\ell(aba) = 1$ and $\mathcal{L}_\ell(\varepsilon) = 0$. Moreover, $a \notin \mathit{Approx}(\varepsilon)$ since $\mathcal{C}_\ell(a) \neq \perp, \mathcal{C}_\ell(\varepsilon) \neq \perp$, and $\mathcal{C}_\ell(a) \neq \mathcal{C}_\ell(\varepsilon)$. With the same arguments, we also discard $ab, b, abb, abaa, abab$. Thus, $\mathit{Approx}(\varepsilon) = \{\varepsilon, aa, aaa, aab\}$. On the other hand, $\mathit{Approx}(aa) = \{\varepsilon, a, ab, aa, abb, aaa, aab\}$ knowing that $\mathcal{C}_\ell(aa) = \perp$. □

The following notation will be convenient later. Let $\mathscr{O}_\ell$ be an observation table and $u \in R \cup R\Sigma$. If $\mathcal{C}_\ell(u) = \bot$ (which means that $u \notin \mathit{Pref}(\mathscr{O}_\ell)$), then we say that u is a *$\bot$-word*. Let us denote by $\overline{R}$ the set R from which the $\bot$-words have been removed. We define $\overline{R \cup R\Sigma}$ in a similar way. We can now formalize the relation between $\equiv$ and *Approx*.

Proposition 2. *Let $\mathscr{O}_\ell$ be an observation table up to $\ell \in \mathbb{N}$. Then for all $u, v \in \overline{R \cup R\Sigma}$, we have $u \equiv v \Rightarrow u \in \mathit{Approx}(v)$.*

Closed and Consistent Observation Table. As for the L^* algorithm [4], we need to define the constraints a table $\mathscr{O}_\ell$ must respect in order to obtain a congruence relation from *Approx* and then to construct a DFA. This is more complex than for L^*. Namely, the table must be closed, Σ-consistent, and $\bot$-consistent. The first two constraints are close to the ones already imposed by L^*. The last one is new. Crucially, it implies that *Approx* is transitive.

Definition 5. *Let $\mathscr{O}_\ell$ be an observation table up to $\ell \in \mathbb{N}$. We say the table is:*

- closed *if $\forall u \in R\Sigma, \mathit{Approx}(u) \cap R \neq \emptyset$,*
- Σ-consistent *if $\forall u \in R$, $\forall a \in \Sigma$,*

$$ua \in \bigcap_{v \in \mathit{Approx}(u) \cap R} \mathit{Approx}(va),$$

- $\bot$-consistent *if $\forall u, v \in R \cup R\Sigma$ such that $u \in \mathit{Approx}(v)$,*

$$\forall s \in S,\ \mathcal{C}_\ell(us) \neq \bot \Leftrightarrow \mathcal{C}_\ell(vs) \neq \bot.$$

Example 4. Let $\mathscr{O}_\ell$ be the table from Figure 3. We have $\mathit{Approx}(b) \cap R \neq \emptyset$ because $aba \in \mathit{Approx}(b)$. More generally one can check that $\mathscr{O}_\ell$ is closed. However, $\mathscr{O}_\ell$ is not Σ-consistent. Indeed, $\varepsilon b \notin \bigcap_{v \in \mathit{Approx}(\varepsilon) \cap R} \mathit{Approx}(vb)$ since $\mathit{Approx}(\varepsilon) \cap R = \{\varepsilon, aa\}$ and $\varepsilon b \notin \mathit{Approx}(aab)$. Finally, $\mathscr{O}_\ell$ is also not $\bot$-consistent since $aa \in \mathit{Approx}(\varepsilon)$ but $\mathcal{C}_\ell(aa) = \bot$ and $\mathcal{C}_\ell(\varepsilon) = 0$. □

When $\mathscr{O}_\ell$ is closed and consistent, we define the following relation $\equiv_{\mathscr{O}_\ell}$: $\forall u, v \in R \cup R\Sigma$, $u \equiv_{\mathscr{O}_\ell} v \Leftrightarrow u \in \mathit{Approx}(v)$. This relation is a congruence over R from which we can construct a DFA $\mathcal{A}_{\mathscr{O}_\ell}$.

Definition 6. *Let $\mathscr{O}_\ell$ be a closed, Σ- and $\bot$-consistent observation table up to ℓ. From $\equiv_{\mathscr{O}_\ell}$, we define the DFA $\mathcal{A}_{\mathscr{O}_\ell} = (Q_{\mathscr{O}_\ell}, \Sigma, \delta_{\mathscr{O}_\ell}, q^0_{\mathscr{O}_\ell}, F_{\mathscr{O}_\ell})$ with: (1) $Q_{\mathscr{O}_\ell} = \{[\![u]\!]_{\equiv_{\mathscr{O}_\ell}} \mid u \in R\}$, (2) $q^0_{\mathscr{O}_\ell} = [\![\varepsilon]\!]_{\equiv_{\mathscr{O}_\ell}}$, (3) $F_{\mathscr{O}_\ell} = \{[\![u]\!]_{\equiv_{\mathscr{O}_\ell}} \mid \mathcal{L}_\ell(u) = 1\}$, and (4) the (total) transition function $\delta_{\mathscr{O}_\ell}$ is defined by $\delta_{\mathscr{O}_\ell}([\![u]\!]_{\equiv_{\mathscr{O}_\ell}}, a) = [\![ua]\!]_{\equiv_{\mathscr{O}_\ell}}$, for all $[\![u]\!]_{\equiv_{\mathscr{O}_\ell}} \in Q_{\mathscr{O}_\ell}$ and $a \in \Sigma$.*

Note that $\mathcal{A}_{\mathscr{O}_\ell}$ is consistent with the information stored in $\mathscr{O}_\ell$.

Lemma 1. *For all $u \in R \cup R\Sigma$, we have $u \in \mathcal{L}(\mathcal{A}_{\mathscr{O}_\ell}) \Leftrightarrow u \in L_\ell$.*

Making a Table Closed and Consistent. Suppose we have an observation table $\mathscr{O}_\ell$ up to ℓ and we want to make it closed, Σ- and $\perp$-consistent. We here give some intuition on how to proceed.

If the table $\mathscr{O}_\ell$ is not closed or not Σ-consistent, we proceed as in the L^* algorithm [4]. In the first case, this means that $\exists u \in R\Sigma$, $\mathit{Approx}(u) \cap R = \emptyset$. It follows that $u \notin R$ and we thus add u to R and update the table. In the second case, this means that $\exists ua \in R\Sigma, \exists v \in \mathit{Approx}(u) \cap R$, $ua \notin \mathit{Approx}(va)$. We have two cases: there exists $s \in S$ such that either $\mathcal{L}_\ell(uas) \neq \mathcal{L}_\ell(vas)$, or $\mathcal{C}_\ell(uas) \neq \perp \wedge \mathcal{C}_\ell(vas) \neq \perp \wedge \mathcal{C}_\ell(uas) \neq \mathcal{C}_\ell(vas)$. In both cases, we add as to S and to $\widehat{S}$ and we update the table.

Suppose that $\mathscr{O}_\ell$ is not $\perp$-consistent, i.e., $\exists u, v \in R \cup R\Sigma, \exists s \in S$, $u \in \mathit{Approx}(v)$ and $\mathcal{C}_\ell(us) \neq \perp \Leftrightarrow \mathcal{C}_\ell(vs) = \perp$. We call *mismatch* the latter disequality. Let us assume, without loss of generality, that $\mathcal{C}_\ell(us) \neq \perp$ and $\mathcal{C}_\ell(vs) = \perp$. So, $us \in \mathit{Pref}(\mathscr{O}_\ell)$, i.e., there exist $u' \in R \cup R\Sigma$ and $s' \in \widehat{S}$ such that $us \in \mathit{Pref}(u's')$ and $\mathcal{L}_\ell(u's') = 1$. We denote by s'' the word such that $us'' = u's'$. The idea is to add $\mathit{Suff}(s'')$ to one or both sets $S, \widehat{S}$. We have two cases:

- Suppose u' is a prefix of u. We have $s'' \in \widehat{S} \setminus S$ and add $\mathit{Suff}(s'')$ to S.
- Suppose u is a proper prefix of u'. If $vs'' \in L_\ell$ then we add $\mathit{Suff}(s'')$ to $\widehat{S}$, otherwise we add $\mathit{Suff}(s'')$ to both S and $\widehat{S}$.

The difficult task is to prove that it is always possible to make a table closed and consistent in finite time.

Proposition 3. *Given an observation table $\mathscr{O}_\ell$ up to $\ell \in \mathbb{N}$, there exists an algorithm that makes it closed, Σ- and $\perp$-consistent in a finite amount of time.*

Let us give some rough intuition. Notice that R increases only when the table is not closed, and that $S, \widehat{S}$ may increase only when the table is not consistent. Firstly, the number of times the table is not closed is bounded by the number of classes of $\equiv$ up to counter limit ℓ, by Proposition 2. Indeed, when $u \in R\Sigma \setminus R$, witness that $\mathscr{O}_\ell$ is not closed, is added to R, then it becomes the only representative in its new approximation set. Secondly, one can prove that after resolving a case where the table is not consistent, then either the size of an approximation set decreases or a mismatch is eliminated. The number of times an approximation set may decrease is bounded, because there are at most $|R \cup R\Sigma|$ distinct such sets whose size is bounded by $|R \cup R\Sigma|$. Finally, the number of mismatches to eliminate is also bounded. Hard work was necessary to get this result as, when one mismatch is eliminated when solving a case where the table is not consistent, S may increase, inducing the creation of new mismatches.

Handling Counterexamples to Partial Equivalence Queries. Let $\mathcal{A}_{\mathscr{O}_\ell}$ be the DFA constructed from a closed, Σ- and $\perp$-consistent observation table $\mathscr{O}_\ell$. If the teacher's answer to a partial equivalence query over $\mathcal{A}_{\mathscr{O}_\ell}$ is positive, then $\mathcal{A}_{\mathscr{O}_\ell}$ exactly accepts L_ℓ. Otherwise, the teacher returns a counterexample, that is, a word $w \in \Sigma^*$ such that $w \in L_\ell \Leftrightarrow w \notin \mathcal{L}(\mathcal{A}_{\mathscr{O}_\ell})$. In the latter case, we add

Pref (w) to R and update the table. We finally make the new table $\mathscr{O}'_\ell$ closed, Σ- and $\perp$-consistent. We have that $\equiv_{\mathscr{O}'_\ell}$ is a strict refinement of $\equiv_{\mathscr{O}_\ell}$.

Proposition 4. *For all $u, v \in R \cup R\Sigma$, we have $u \equiv_{\mathscr{O}'_\ell} v \Rightarrow u \equiv_{\mathscr{O}_\ell} v$. Furthermore, the index of $\equiv_{\mathscr{O}'_\ell}$ is strictly greater than the index of $\equiv_{\mathscr{O}_\ell}$.*

Since the number of classes of $\equiv$ up to counter limit ℓ is bounded by the width K and the $\ell + 1$ levels of $BG_\ell(\mathcal{A})$, by Propositions 2 to 4, we deduce that after a finite number of steps, we obtain an observation table $\mathscr{O}_\ell$ and its corresponding DFA $\mathcal{A}_{\mathscr{O}_\ell}$ such that $\mathcal{L}(\mathcal{A}_{\mathscr{O}_\ell}) = L_\ell$.

Algorithm 1 Learning an ROCA

```
Require: A teacher knowing an ROCA $\mathcal{A}$
Ensure: An ROCA accepting the same language is returned
 1: Initialize the observation table $\mathscr{O}_\ell$ with $\ell = 0, R = S = \widehat{S} = \{\varepsilon\}$
 2: while true do
 3:     Make $\mathscr{O}_\ell$ closed, $\Sigma$-, and $\perp$-consistent
 4:     Construct the DFA $\mathcal{A}_{\mathscr{O}_\ell}$ from $\mathscr{O}_\ell$
 5:     Ask a partial equivalence query over $\mathcal{A}_{\mathscr{O}_\ell}$
 6:     if the answer is negative then
 7:         Update $\mathscr{O}_\ell$ with the provided counterexample          ▷ $\ell$ is not modified
 8:     else
 9:         Identify all periodic descriptions $\alpha_1, \dots, \alpha_n$ of $\mathcal{A}_{\mathscr{O}_\ell}$
10:         Construct an ROCA $\mathcal{A}_{\alpha_i}$ for each $\alpha_i$
11:         Ask an equivalence query over each $\mathcal{A}_{\alpha_i}$
12:         if the answer is true for an $\mathcal{A}_{\alpha_i}$ then return $\mathcal{A}_{\alpha_i}$
13:         else Select one counterexample and update $\mathscr{O}_\ell$          ▷ $\ell$ is increased
```

Learning Algorithm. We have every piece needed to give the learning algorithm for ROCAs, as presented in Algorithm 1. We initialize the observation table $\mathscr{O}_\ell$ with $\ell = 0, R = S = \widehat{S} = \{\varepsilon\}$. Then, we make the table closed, Σ-, and $\perp$-consistent, construct the DFA $\mathcal{A}_{\mathscr{O}_\ell}$, and ask for a partial equivalence query with $\mathcal{A}_{\mathscr{O}_\ell}$. If the teacher answers positively, we have learned a DFA accepting L_ℓ. Otherwise, we use the provided counterexample to update the table without increasing ℓ. Once the learned DFA $\mathcal{A}_{\mathscr{O}_\ell}$ accepts the language L_ℓ, the next proposition states that the initial fragments (up to a certain counter limit) of both $\mathcal{A}_{\mathscr{O}_\ell}$ and $BG(\mathcal{A})$ are isomorphic. This means that, once we have learned a long enough initial fragment, we can extract a periodic description from $\mathcal{A}_{\mathscr{O}_\ell}$ that is valid for $BG(\mathcal{A})$.

Proposition 5. *Let $BG(\mathcal{A})$ be the behavior graph of an ROCA $\mathcal{A}$, K be its width, and m, k be the offset and the period of a periodic description of $BG(\mathcal{A})$. Let $s = m + (K \cdot k)^4$. Let $\mathscr{O}_\ell$ be a closed, Σ- and $\perp$-consistent observation table up to $\ell > s$ such that $\mathcal{L}(\mathcal{A}_{\mathscr{O}_\ell}) = L_\ell$. Then, the trim parts of the subautomata of $BG(\mathcal{A})$ and $\mathcal{A}_{\mathscr{O}_\ell}$ restricted to the levels in $\{0, \dots, \ell - s\}$ are isomorphic.*

Hence we extract all possible periodic descriptions α from $\mathcal{A}_{\mathscr{O}_\ell}$. By Proposition 1, each description α yields an ROCA $\mathcal{A}_\alpha$ on which we ask for an equivalence query. If the teacher answers positively, we have learned an ROCA accepting L and we are done. Otherwise, we need to increase the counter limit and update the table using some of the counterexamples provided by the teacher.

Extracting every possible periodic description of $\mathcal{A}_{\mathscr{O}_\ell}$ can be performed by identifying an isomorphism between two consecutive subgraphs of $\mathcal{A}_{\mathscr{O}_\ell}$. That is, we fix values for the offset m and period k and see if the subgraphs induced by the levels m to $m+k-1$, and by the levels $m+k$ to $m+2k-1$ are isomorphic (this means considering all pairs (m,k) such that $m+2k-1 \leq \ell$). This can be done by executing two depth-first searches in parallel [27]. Note that multiple periodic descriptions may be found, due to the finite knowledge of the learner.

In case all ROCAs constructed from $\mathcal{A}_{\mathscr{O}_\ell}$ do not accept L, we handle the counterexamples returned by the teacher as follows. If among them, there is one counterexample, say w, such that the height $h_{\mathcal{A}}(w)$ exceeds ℓ, we add $\mathit{Pref}(w)$ to R (as in the case of a negative partial equivalence query) and the new counter limit is updated to $h_{\mathcal{A}}(w)$. If none of the counterexamples have an height exceeding ℓ (this may happen due to the limited knowledge of the learner), we instead use $\mathcal{A}_{\mathscr{O}_\ell}$ directly as an ROCA and ask an equivalence query. Since $\mathcal{L}(\mathcal{A}_{\mathscr{O}_\ell}) = L_\ell$ (as the last partial equivalence query was true), the counterexample returned by the teacher necessarily has a high enough height and we proceed as above.

Complexity of the Algorithm. Let us briefly explain the complexity announced in Theorem 2 for Algorithm 1 in terms of $|Q|$ the number of states of the given ROCA and t the length of the longest counterexample returned by the teacher. The given bound on the number of (partial) equivalence queries is obtained by arguments similar to those of [27]. The number of steps in the main loop of Algorithm 1 is the (polynomial) number of partial equivalence queries. During one step in this loop, by carefully studying how we make the table closed and consistent and handle a counterexample, we get that $R \cup R\Sigma$ (resp. $\widehat{S}$) grows linearly (resp. exponentially) in $|Q|$, $|\Sigma|$, and t. We also get an exponential number of membership and equivalence queries for the whole algorithm.

4 Experiments

We evaluated our algorithm on two types of benchmarks. The first uses randomly generated ROCAs, while the second focuses on a new approach to learn an ROCA that can efficiently check if a JSON document is valid against a given JSON schema. Notice that while there exist several algorithms that infer a JSON schema from a collection of JSON documents (see survey [5]), none are based on learning techniques nor do they yield an automaton-based validation algorithm.

The ROCAs and the learning algorithm were implemented by extending the well-known Java libraries AutomataLib and LearnLib [20]. These modifications can be consulted on [31,33], while the code for the benchmarks is available on [32]. Implementation specific details (such as the libraries) are given alongside the code. The server used for the computations ran Debian 10 over Linux 5.4.73-1-pve with a 4-core Intel® Xeon® Silver 4214R Processor with 16.5M cache, and 64GB of RAM. Moreover, we used OpenJDK version 11.0.12.

4.1 Random ROCAs

We first discuss our benchmarks based on randomly generated ROCAs.

Random Generation of ROCAs. An ROCA with given size $n = |Q|$ is randomly generated such that (1) $\forall q \in Q, q$ has a probability 0.5 of being final, and (2) $\forall q \in Q, \forall a \in \Sigma, \delta_{>0}(q, a) = (p, c)$ with p a random state in Q and c a random counter operation in $\{-1, 0, +1\}$. We define $\delta_{=0}(q, a) = (p, c)$ in a similar way except that $c \in \{0, +1\}$. All random draws are assumed to come from a uniform distribution. Since this generation does not guarantee an ROCA with n reachable states, we generate 100 ROCAs and select the ROCA with a maximal number of reachable states. However, it is still possible the resulting ROCA does not have n (co)-reachable states.

Equivalence of Two ROCAs. The language equivalence problem of ROCAs is known to be decidable and NL-complete [8]. Unfortunately, the algorithm described in [8] is difficult to implement. Instead, we use an "approximate" equivalence oracle for our experiments.[3] Let $\mathcal{A}$ and $\mathcal{B}$ be two ROCAs such that $\mathcal{B}$ is the learned ROCA from a periodic description with period k. The algorithm explores the configuration space of both ROCAs in parallel. If, at some point, it reaches a pair of configurations such that one is accepting and the other not, then we have a counterexample. However, to have an algorithm that eventually stops, we need to bound the counter value of the configurations to explore. Our approach is to first explore up to counter value $|\mathcal{A} \times \mathcal{B}|^2$ (in view of [8, Proposition 18] about shortest accepting runs in an ROCA). If no counterexample is found, we add k to the bound and, with probability 0.5, a new exploration is done up to the new bound. We repeat this whole process until we find a counterexample or until the random draw forces us to stop.

Results. For our random benchmarks, we let the size $|Q|$ of the ROCA vary between one and five, and the size of $|\Sigma|$ of the alphabet between one and four. For each pair $(|Q|, |\Sigma|)$ of sizes, we execute the learning algorithm on 100 ROCAs (generated as explained above). We set a timeout of 20 minutes and a memory limit of 16GB. The number of executions with a timeout is given in Table 1 (we do not give the pairs $(|Q|, |\Sigma|)$ where every execution could finish).

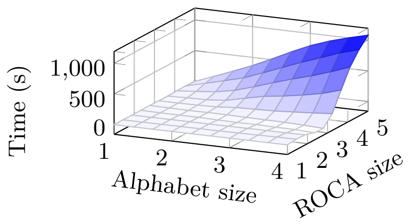
The mean of the total time taken by the algorithm is given in Figure 5a. One can see that it has an exponential growth in both sizes $|Q|$ and $|\Sigma|$. Note that

[3] The teacher might, with some small probability, answer with false positives but never with false negatives.

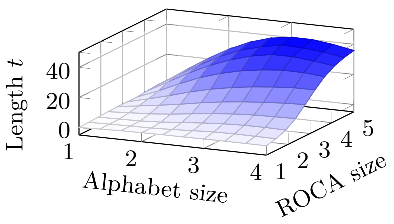
Table 1: Number (over 100) of executions with a timeout (TO). The executions for the missing pairs $(|Q|, |\Sigma|)$ could all finish.

$\|Q\|$	$\|\Sigma\|$	TO (20 min)
4	1	0
4	2	5
4	3	16
4	4	41
5	1	0
5	2	23
5	3	55
5	4	83

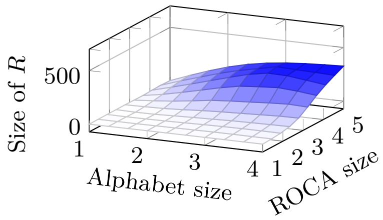
executions with a timeout had their execution time set to 20 minutes, in order to highlight the curve. Let us now drop all the executions with a timeout. The mean length of the longest counterexample provided by the teacher for (partial) equivalence queries is presented in Figure 5b and the final size of the sets R and $\widehat{S}$ is presented in Figures 5c and 5d. Note that the curves go down due to the limited number of remaining executions (for instance, the ones that could finish did not require long counterexamples). We can see that $\widehat{S}$ grows larger than R, which is coherent with the theoretical results stated at the end of Section 3.



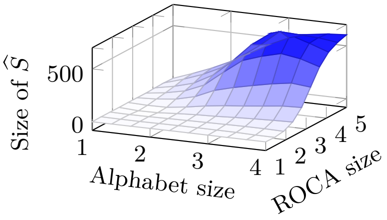
(a) Mean of the total time taken by the learning algorithm.



(b) Mean of the length t of the longest counterexample.



(c) Mean of the final size of R.



(d) Mean of the final size of $\widehat{S}$.

Fig. 5: Results for the benchmarks based on random ROCAs.

4.2 JSON Documents and JSON Schemas

Let us now discuss the second set of benchmarks, which constitutes a proof-of-concept for an efficient validator of *JSON-document* [11] streams and is inspired by [13]. This format is currently the most popular one for exchanging information on the web. Constraints over documents can be described by a *JSON schema* [22] (like DTDs do for XML documents). See [22,21] for a brief overview of JSON documents and schemas.

In our learning process, the learner aims to construct an ROCA that can validate a JSON document, according to the schema. We assume the teacher knows the target schema and the queries are specialized as follows: (1) Membership queries: the learner provides a JSON document and the teacher answers true if the document is valid for the schema. (2) Counter value queries: the learner provides a JSON document and the teacher returns the number of unmatched `{` and `[`. Adding the two values is a heuristic abstraction that allows us to summarize two-counter information into a single counter value. Importantly, the abstraction is a design choice regarding our implementation of a teacher for these experiments and not an assumption made by our learning algorithm. (3) Partial equivalence query: the learner provides a DFA and a counter limit ℓ. The teacher randomly generates an a-priori fixed number of documents with a height not exceeding the counter limit ℓ and checks whether the DFA and the schema both agree on the documents' validity. If a disagreement is noticed, the incorrectly classified document is returned. (4) Equivalence query: the learner provides an ROCA. It is very similar to partial equivalence queries, except that documents are generated without a bound on the height. Note that the randomness of the (partial) equivalence queries implies that the learned ROCA may not completely recognize the same set of documents as for the schema.

In order for an ROCA to be learned in a reasonable time, some abstractions are made mainly to reduce the alphabet size: (1) If an object has a key named `key`, we consider the sequence of characters `"key"` as a single alphabet symbol. (2) Strings, integers, and numbers are abstracted as `"\S"`, `"\I"`, and `"\D"` respectively. Booleans are left unmodified. (3) The symbols `,`, `{`, `}`, `[`, `]`, `:` are all considered as different alphabet symbols. (4) We assume each object is composed of an ordered (instead of unordered) collection of pairs key-value. Note that the learning algorithm can learn without these restrictions but it requires substantially more time, due to a blowup in the state space or in the alphabet.

Moreover, notice that the alphabet is not known at the start of the learning process (due to the fact that keys can be any strings). Therefore we slightly modify the learning algorithm to support growing alphabets. More precisely, the learner's alphabet starts with the symbols `{` and `}` (to guarantee we can at least produce a syntactically valid JSON document for the first partial equivalence query) and is augmented each time a new symbol is seen.

Results. We considered three JSON schemas. The first is a simple document listing all possible values (i.e., it contains an integer, a double, and so on). The

Table 2: Results for JSON benchmarks.

Schema	TO (1h)	Time (s)	t	$\lvert R\rvert$	$\lvert\widehat{S}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\Sigma\rvert$
1	0	16.39	31.00	55.55	32.00	33.00	19.00
2	27	1045.64	12.99	57.84	33.74	44.29	14.70
3	19	922.19	49.49	171.94	50.49	51.16	9.00

second is a real-world JSON schema[4] used by a code coverage tool called Codecov [14]. Finally, the third schema encodes a recursive list, i.e., an object containing a list with at most one object defined recursively. This last example is used to force the behavior graph to be infinite.

Table 2 gives the results of the benchmarks, obtained by fixing the number of random documents by (partial) equivalence query to be 1000. For each schema, 100 executions were ran with a time limit of one hour and a memory limit of 16GB by execution. We can see that real-world JSON schemas and recursively-defined schemas can be both learned by our approach. One last interesting statistics is that $|R|$ is larger than $|\widehat{S}|$, unlike for the random benchmarks.

5 Future Work

As future work, we believe one might be able to remove the use of partial equivalence queries. In this direction, perhaps replacing our use of Neider and Löding's VCA algorithm by Isberner's TTT algorithm [19] for visibly pushdown automata might help. Indeed, the TTT algorithm does not need partial equivalence queries.

Another interesting direction concerns lowering the (query) complexity of our algorithm. In [29], it is proved that L^* algorithm [4] can be modified so that adding a single separator after a failed equivalence query is enough to update the observation table. This would remove the suffix-closedness requirements on the separator sets S and $\widehat{S}$. It is not immediately clear to us whether the definition of $\perp$-consistency presented here holds in that context. Further optimizations, such as discrimination tree-based algorithms (see e.g. Kearns and Vazirani's algorithm [24]), also do not need the separator set to be suffix-closed.

It would also be interesting to directly learn the one-counter language instead of an ROCA. Indeed, our algorithm learns some ROCA that accepts the target language. It would be desirable to learn some canonical representation of the language (e.g. a minimal automaton, for some notion of minimality).

Finally, as far as we know, there currently is no active learning algorithm for deterministic one-counter automata (such that ε-transitions are allowed). We want to study how we can adapt our learning algorithm in this context.

[4] We downloaded the schema from the JSON Schema Store [23]. We modified the file to remove all constraints of type "enum".

References

1. Aarts, F., Jonsson, B., Uijen, J., Vaandrager, F.W.: Generating models of infinite-state communication protocols using regular inference with abstraction. Formal Methods Syst. Des. **46**(1), 1–41 (2015). https://doi.org/10.1007/s10703-014-0216-x, https://doi.org/10.1007/s10703-014-0216-x
2. Abel, A., Reineke, J.: Gray-box learning of serial compositions of mealy machines. In: Rayadurgam, S., Tkachuk, O. (eds.) NASA Formal Methods - 8th International Symposium, NFM 2016, Minneapolis, MN, USA, June 7-9, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9690, pp. 272–287. Springer (2016). https://doi.org/10.1007/978-3-319-40648-0_21, https://doi.org/10.1007/978-3-319-40648-0_21
3. Aho, A.V., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools. Addison-Wesley series in computer science / World student series edition, Addison-Wesley (1986), https://www.worldcat.org/oclc/12285707
4. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6, https://doi.org/10.1016/0890-5401(87)90052-6
5. Baazizi, M.A., Colazzo, D., Ghelli, G., Sartiani, C.: Schemas and types for JSON data. In: Herschel, M., Galhardas, H., Reinwald, B., Fundulaki, I., Binnig, C., Kaoudi, Z. (eds.) Advances in Database Technology - 22nd International Conference on Extending Database Technology, EDBT 2019, Lisbon, Portugal, March 26-29, 2019. pp. 437–439. OpenProceedings.org (2019). https://doi.org/10.5441/002/edbt.2019.39, https://doi.org/10.5441/002/edbt.2019.39
6. Berman, P., Roos, R.: Learning one-counter languages in polynomial time (extended abstract). In: 28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987. pp. 61–67. IEEE Computer Society (1987). https://doi.org/10.1109/SFCS.1987.36, https://doi.org/10.1109/SFCS.1987.36
7. Berthon, R., Boiret, A., Pérez, G.A., Raskin, J.: Active learning of sequential transducers with side information about the domain. In: Moreira, N., Reis, R. (eds.) Developments in Language Theory - 25th International Conference, DLT 2021, Porto, Portugal, August 16-20, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12811, pp. 54–65. Springer (2021). https://doi.org/10.1007/978-3-030-81508-0_5, https://doi.org/10.1007/978-3-030-81508-0_5
8. Böhm, S., Göller, S., Jancar, P.: Bisimulation equivalence and regularity for real-time one-counter automata. J. Comput. Syst. Sci. **80**(4), 720–743 (2014). https://doi.org/10.1016/j.jcss.2013.11.003, https://doi.org/10.1016/j.jcss.2013.11.003
9. Bollig, B.: One-counter automata with counter observability. In: Lal, A., Akshay, S., Saurabh, S., Sen, S. (eds.) 36th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2016, December 13-15, 2016, Chennai, India. LIPIcs, vol. 65, pp. 20:1–20:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2016). https://doi.org/10.4230/LIPIcs.FSTTCS.2016.20, https://doi.org/10.4230/LIPIcs.FSTTCS.2016.20
10. Bouajjani, A., Bozga, M., Habermehl, P., Iosif, R., Moro, P., Vojnar, T.: Programs with lists are counter automata. Formal Methods Syst. Des. **38**(2), 158–192 (2011). https://doi.org/10.1007/s10703-011-0111-7, https://doi.org/10.1007/s10703-011-0111-7

11. Bray, T.: The javascript object notation (JSON) data interchange format. RFC **8259**, 1–16 (2017). https://doi.org/10.17487/RFC8259, https://doi.org/10.17487/RFC8259
12. Bruyère, V., Pérez, G.A., Staquet, G.: Learning realtime one-counter automata. CoRR **abs/2110.09434** (2021), https://arxiv.org/abs/2110.09434
13. Chitic, C., Rosu, D.: On validation of XML streams using finite state machines. In: Amer-Yahia, S., Gravano, L. (eds.) Proceedings of the Seventh International Workshop on the Web and Databases, WebDB 2004, June 17-18, 2004, Maison de la Chimie, Paris, France, Colocated with ACM SIGMOD/PODS 2004. pp. 85–90. ACM (2004). https://doi.org/10.1145/1017074.1017096, https://doi.org/10.1145/1017074.1017096
14. Codecov, https://about.codecov.io/
15. Fahmy, A.F., Roos, R.S.: Efficient learning of real time one-counter automata. In: Jantke, K.P., Shinohara, T., Zeugmann, T. (eds.) Algorithmic Learning Theory, 6th International Conference, ALT '95, Fukuoka, Japan, October 18-20, 1995, Proceedings. Lecture Notes in Computer Science, vol. 997, pp. 25–40. Springer (1995). https://doi.org/10.1007/3-540-60454-5_26, https://doi.org/10.1007/3-540-60454-5_26
16. Garhewal, B., Vaandrager, F.W., Howar, F., Schrijvers, T., Lenaerts, T., Smits, R.: Grey-box learning of register automata. In: Dongol, B., Troubitsyna, E. (eds.) Integrated Formal Methods - 16th International Conference, IFM 2020, Lugano, Switzerland, November 16-20, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12546, pp. 22–40. Springer (2020). https://doi.org/10.1007/978-3-030-63461-2_2, https://doi.org/10.1007/978-3-030-63461-2_2
17. Groce, A., Peled, D.A., Yannakakis, M.: Adaptive model checking. Log. J. IGPL **14**(5), 729–744 (2006). https://doi.org/10.1093/jigpal/jzl007, https://doi.org/10.1093/jigpal/jzl007
18. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation, Second Edition. Addison-Wesley (2000)
19. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: A redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8734, pp. 307–322. Springer (2014). https://doi.org/10.1007/978-3-319-11164-3_26, https://doi.org/10.1007/978-3-319-11164-3_26
20. Isberner, M., Howar, F., Steffen, B.: The open-source learnlib - A framework for active automata learning. In: Kroening, D., Pasareanu, C.S. (eds.) Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9206, pp. 487–495. Springer (2015). https://doi.org/10.1007/978-3-319-21690-4_32, https://doi.org/10.1007/978-3-319-21690-4_32
21. Json.org, https://www.json.org
22. Json schema, https://json-schema.org
23. Json schema store, https://www.schemastore.org/json/
24. Kearns, M.J., Vazirani, U.V.: An Introduction to Computational Learning Theory. MIT Press (1994), https://mitpress.mit.edu/books/introduction-computational-learning-theory
25. Leucker, M., Neider, D.: Learning minimal deterministic automata from inexperienced teachers. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering

Change - 5th International Symposium, ISoLA 2012, Heraklion, Crete, Greece, October 15-18, 2012, Proceedings, Part I. Lecture Notes in Computer Science, vol. 7609, pp. 524–538. Springer (2012). https://doi.org/10.1007/978-3-642-34026-0_39, https://doi.org/10.1007/978-3-642-34026-0_39

26. Michaliszyn, J., Otop, J.: Learning deterministic automata on infinite words. In: Giacomo, G.D., Catalá, A., Dilkina, B., Milano, M., Barro, S., Bugarín, A., Lang, J. (eds.) ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020). Frontiers in Artificial Intelligence and Applications, vol. 325, pp. 2370–2377. IOS Press (2020). https://doi.org/10.3233/FAIA200367, https://doi.org/10.3233/FAIA200367
27. Neider, D., Löding, C.: Learning visibly one-counter automata in polynomial time. Tech. rep., Technical Report AIB-2010-02, RWTH Aachen (January 2010) (2010)
28. Peled, D.A., Vardi, M.Y., Yannakakis, M.: Black box checking. J. Autom. Lang. Comb. **7**(2), 225–246 (2002). https://doi.org/10.25596/jalc-2002-225, https://doi.org/10.25596/jalc-2002-225
29. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. **103**(2), 299–347 (1993). https://doi.org/10.1006/inco.1993.1021, https://doi.org/10.1006/inco.1993.1021
30. Roos, R.S.: Deciding equivalence of deterministic one-counter automata in polynomial time with applications to learning (1988)
31. Staquet, G.: Automatalib fork for rocas, https://github.com/DocSkellington/automatalib
32. Staquet, G.: Code for the benchmarks for roca learning, https://github.com/DocSkellington/LStar-ROCA-Benchmarks
33. Staquet, G.: Learnlib fork for rocas, https://github.com/DocSkellington/Learnlib
34. Valiant, L.G., Paterson, M.: Deterministic one-counter automata. J. Comput. Syst. Sci. **10**(3), 340–350 (1975). https://doi.org/10.1016/S0022-0000(75)80005-5, https://doi.org/10.1016/S0022-0000(75)80005-5