Learning Realtime One-Counter Automata

Submitted at TACAS 2022

Véronique Bruyère Guillermo A. Pérez Gaëtan Staquet

Theoretical computer science
Computer Science Department
Science Faculty
University of Mons

Formal Techniques in Software Engineering
Computer Science Department
Science Faculty
University of Antwerp

November 17, 2021

1. Motivation

2. Learning deterministic finite automata

3. Learning realtime one-counter automata

4. Experimental results

```json
{
    "string": "Hello, world",
    "integer": 42,
    "double": 2.718,
    "array": ["string"],
    "object": {"anything": "correct"}
}
```

Listing 1: A JSON document.

[1] For XML documents, see Chitic and Rosu, "On Validation of XML Streams Using Finite State Machines", 2004

```
{
    "string": "Hello, world",
    "integer": 42,
    "double": 2.718,
    "array": ["string"],
    "object": {"anything": "correct"}
}
```

Listing 1: A JSON document.

How to know whether a JSON document satisfies a given set of constraints?

[1] For XML documents, see Chitic and Rosu, "On Validation of XML Streams Using Finite State Machines", 2004

```json
{
    "string": "Hello, world",
    "integer": 42,
    "double": 2.718,
    "array": ["string"],
    "object": {"anything": "correct"}
}
```

Listing 1: A JSON document.

How to know whether a JSON document satisfies a given set of constraints?

$\hookrightarrow$ Automata-based verification[1].

[1] For XML documents, see Chitic and Rosu, "On Validation of XML Streams Using Finite State Machines", 2004

```json
{
    "string": "Hello, world",
    "integer": 42,
    "double": 2.718,
    "array": ["string"],
    "object": {"anything": "correct"}
}
```

Listing 1: A JSON document.

How to know whether a JSON document satisfies a given set of constraints?

$\hookrightarrow$ Automata-based verification[1].

What kind of automata can be used? How to construct such an automaton?

[1] For XML documents, see Chitic and Rosu, "On Validation of XML Streams Using Finite State Machines", 2004

```json
{
    "string": "Hello, world",
    "integer": 42,
    "double": 2.718,
    "array": ["string"],
    "object": {"anything": "correct"}
}
```

Listing 1: A JSON document.

How to know whether a JSON document satisfies a given set of constraints?

$\hookrightarrow$ Automata-based verification[1].

What kind of automata can be used? How to construct such an automaton?

$\hookrightarrow$ Realtime one-counter automata and our learning algorithm!

[1]For XML documents, see Chitic and Rosu, "On Validation of XML Streams Using Finite State Machines", 2004

1. Motivation

2. Learning deterministic finite automata
   - Deterministic finite automaton
   - Active learning

3. Learning realtime one-counter automata

4. Experimental results

A deterministic finite automaton[2] (DFA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where:

- Q is the set of states,
- Σ is the alphabet,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states, and
- $\delta \subseteq Q \times \Sigma \to Q$ is the transition function.

[2] Hopcroft and Ullman, *Introduction to Automata Theory, Languages and Computation*, 2000.

A deterministic finite automaton[2] (DFA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where:

- Q is the set of states,
- Σ is the alphabet,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states, and
- $\delta \subseteq Q \times \Sigma \to Q$ is the transition function.

The run for the word $w = a_1 \dots a_n \in \Sigma^*$ ($n \in \mathbb{N}$) is the sequence of states

$$q_0 \xrightarrow[\mathcal{A}]{a_1} p_1 \xrightarrow[\mathcal{A}]{a_2} \dots \xrightarrow[\mathcal{A}]{a_n} p_n.$$

If $p_n \in F$, the run is said accepting.

[2] Hopcroft and Ullman, *Introduction to Automata Theory, Languages and Computation*, 2000.

A deterministic finite automaton[2] (DFA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ where:

- Q is the set of states,
- Σ is the alphabet,
- $q_0 \in Q$ is the initial state,
- $F \subseteq Q$ is the set of accepting states, and
- $\delta \subseteq Q \times \Sigma \to Q$ is the transition function.

The run for the word $w = a_1 \dots a_n \in \Sigma^*$ ($n \in \mathbb{N}$) is the sequence of states

$$q_0 \xrightarrow[\mathcal{A}]{a_1} p_1 \xrightarrow[\mathcal{A}]{a_2} \dots \xrightarrow[\mathcal{A}]{a_n} p_n.$$

If $p_n \in F$, the run is said accepting.
The language of $\mathcal{A}$ is the set

$$\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* \mid \exists q \in F, q_0 \xrightarrow[\mathcal{A}]{w} q\}.$$

[2]Hopcroft and Ullman, *Introduction to Automata Theory, Languages and Computation*, 2000.

Let $L \subseteq \Sigma^*$.
We want an algorithm to learn a DFA accepting L.

Let $L \subseteq \Sigma^*$.
We want an algorithm to learn a DFA accepting L.

$\hookrightarrow$ active learning algorithm.

Let $L \subseteq \Sigma^*$.
We want an algorithm to learn a DFA accepting L.

$\hookrightarrow$ active learning algorithm.

Active because the algorithm queries information during the learning process.

Figure 1: Angluin's framework Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

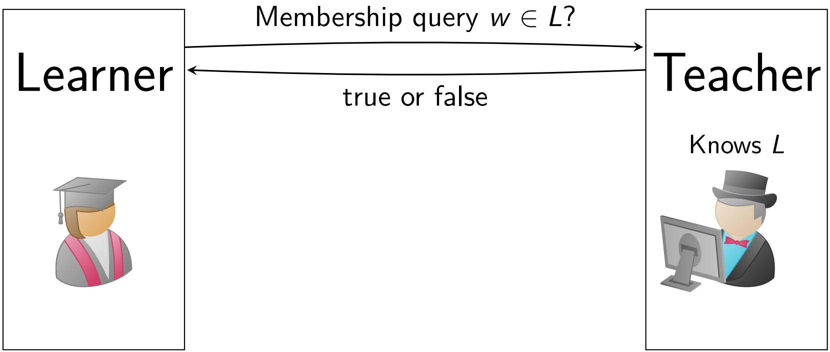


Figure 1: Angluin's framework Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

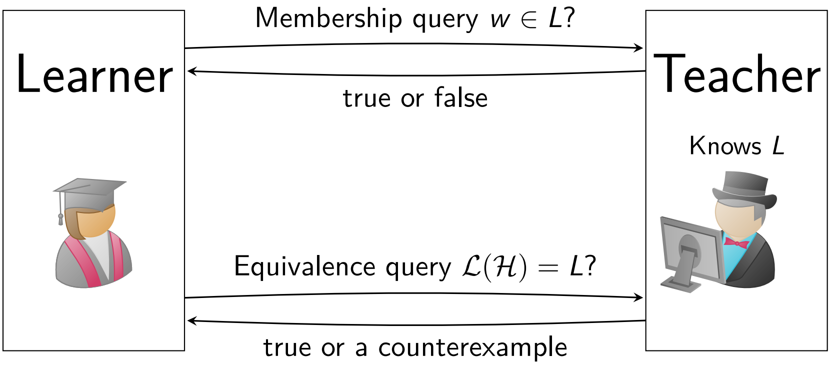


Figure 1: Angluin's framework Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

Algorithm 1 Abstract learner for L^* [Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987]

Require: The target language L
Ensure: A DFA accepting L is returned

Initialize the data structure
Fill the data structure with membership queries
while true **do**
    Make sure the data structure respects some constraints
    Construct the DFA $\mathcal{A}$
    Ask an equivalence query over $\mathcal{A}$
    if the answer is positive **then**
        return $\mathcal{A}$
    else
        Given the counterexample w, refine the data structure
        Fill the data structure with membership queries

Let $L = \{a^n b(b^* a)^m (a|b)^* \mid n, m \geq 0\}$ over $\Sigma = \{a, b\}$.

Let $u \in \Sigma^*$. For all $w \in \Sigma^*$, we look if $uw \in L$.
We construct a table where the rows are indexed by the u and the columns by the w.

Let $L = \{a^n b (b^* a)^m (a|b)^* \mid n, m \geq 0\}$ over $\Sigma = \{a, b\}$.

	ε	a	b	aa	ab	ba	bb	$\dots$
ε	0	0	1	0	1	1	1	$\dots$
a	0	0	1	0	1	1	1	$\dots$
b	1	1	1	1	1	1	1	$\dots$
aa	0	0	1	0	1	1	1	$\dots$
ab	1	1	1	1	1	1	1	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Let $L = \{a^n b(b^* a)^m (a|b)^* \mid n, m \geq 0\}$ over $\Sigma = \{a, b\}$.

	ε	a	b	aa	ab	ba	bb	$\dots$
ε	0	0	1	0	1	1	1	$\dots$
a	0	0	1	0	1	1	1	$\dots$
b	1	1	1	1	1	1	1	$\dots$
aa	0	0	1	0	1	1	1	$\dots$
ab	1	1	1	1	1	1	1	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Let $u, v \in \Sigma^*$ and $L \subseteq \Sigma^*$. We say that $u \sim v$ if and only if[a]

$$\forall w \in \Sigma^*, uw \in L \Leftrightarrow vw \in L.$$

[a] Hopcroft and Ullman, *Introduction to Automata Theory, Languages and Computation*, 2000.

Let $L = \{a^n b (b^* a)^m (a|b)^* \mid n, m \geq 0\}$ over $\Sigma = \{a, b\}$.

	ε	a	b	aa	ab	ba	bb	...
ε	0	0	1	0	1	1	1	...
a	0	0	1	0	1	1	1	...
b	1	1	1	1	1	1	1	...
aa	0	0	1	0	1	1	1	...
ab	1	1	1	1	1	1	1	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Proposition 1

Let L be a language over Σ. Then, there is a DFA accepting L if and only if the index of $\sim$ is finite.

Let $L = \{a^n b(b^* a)^m (a|b)^* \mid n, m \geq 0\}$ over $\Sigma = \{a, b\}$.

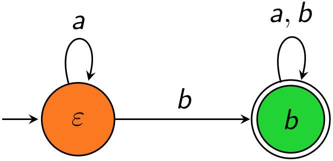
	ε	a	b	aa	ab	ba	bb	$\dots$
ε	0	0	1	0	1	1	1	$\dots$
a	0	0	1	0	1	1	1	$\dots$
b	1	1	1	1	1	1	1	$\dots$
aa	0	0	1	0	1	1	1	$\dots$
ab	1	1	1	1	1	1	1	$\dots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

The Myhill-Nerode congruence encoded in this table has a finite index. We have two equivalence classes: $[\![\varepsilon]\!]_\sim$ and $[\![b]\!]_\sim$.

Let $L = \{a^n b (b^* a)^m (a|b)^* \mid n, m \geq 0\}$ over $\Sigma = \{a, b\}$.

	ε	a	b	aa	ab	ba	bb	$\ldots$
ε	0	0	1	0	1	1	1	$\ldots$
a	0	0	1	0	1	1	1	$\ldots$
b	1	1	1	1	1	1	1	$\ldots$
aa	0	0	1	0	1	1	1	$\ldots$
ab	1	1	1	1	1	1	1	$\ldots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

The Myhill-Nerode congruence encoded in this table has a finite index. We have two equivalence classes: $[\![\varepsilon]\!]_\sim$ and $[\![b]\!]_\sim$.



1. Motivation

2. Learning deterministic finite automata

3. Learning realtime one-counter automata
   - Realtime one-counter automata
   - Behavior graph
   - Learning algorithm

4. Experimental results

A realtime one-counter automaton (ROCA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta_{=0}, \delta_{>0}, q_0, F)$ where Q, q_0, and F are defined as before, and the transition functions $\delta_{=0}$ and $\delta_{>0}$ are defined as:

$$\delta_{=0} : Q \times \Sigma \to Q \times \{0, +1\}$$
$$\delta_{>0} : Q \times \Sigma \to Q \times \{-1, 0, +1\}.$$

A realtime one-counter automaton (ROCA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta_{=0}, \delta_{>0}, q_0, F)$ where Q, q_0, and F are defined as before, and the transition functions $\delta_{=0}$ and $\delta_{>0}$ are defined as:

$$\delta_{=0} : Q \times \Sigma \to Q \times \{0, +1\}$$
$$\delta_{>0} : Q \times \Sigma \to Q \times \{-1, 0, +1\}.$$

A configuration is a pair $(q, n) \in Q \times \mathbb{N}$.

A realtime one-counter automaton (ROCA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta_{=0}, \delta_{>0}, q_0, F)$ where Q, q_0, and F are defined as before, and the transition functions $\delta_{=0}$ and $\delta_{>0}$ are defined as:

$$\delta_{=0} : Q \times \Sigma \to Q \times \{0, +1\}$$
$$\delta_{>0} : Q \times \Sigma \to Q \times \{-1, 0, +1\}.$$

A configuration is a pair $(q, n) \in Q \times \mathbb{N}$.

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 if

$$\begin{cases} \delta_{=0}(q, a) = (p, c) \wedge m = n + c & \text{if } n = 0 \\ \delta_{>0}(q, a) = (p, c) \wedge m = n + c & \text{if } n > 0. \end{cases}$$

A realtime one-counter automaton (ROCA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta_{=0}, \delta_{>0}, q_0, F)$ where Q, q_0, and F are defined as before, and the transition functions $\delta_{=0}$ and $\delta_{>0}$ are defined as:

$$\delta_{=0} : Q \times \Sigma \to Q \times \{0, +1\}$$
$$\delta_{>0} : Q \times \Sigma \to Q \times \{-1, 0, +1\}.$$

A configuration is a pair $(q, n) \in Q \times \mathbb{N}$.
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 if

$$\begin{cases} \delta_{=0}(q, a) = (p, c) \wedge m = n + c & \text{if } n = 0 \\ \delta_{>0}(q, a) = (p, c) \wedge m = n + c & \text{if } n > 0. \end{cases}$$

Let $w \in \Sigma^*$. The counter value of w, according to $\mathcal{A}$, is:

$$c_{\mathcal{A}}(w) = n \Leftrightarrow \exists q \in Q, (q_0, 0) \xrightarrow[\mathcal{A}]{w} (q, n).$$

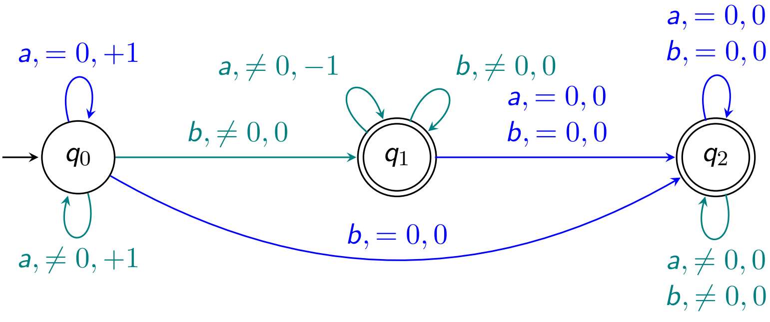


Figure 2: An ROCA $\mathcal{A}$.

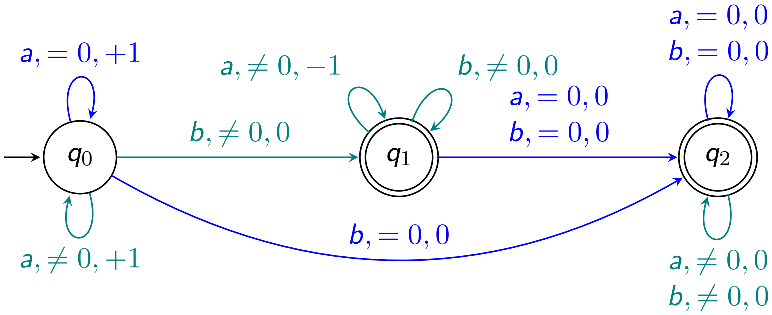


Figure 2: An ROCA $\mathcal{A}$.

$$(q_0, 0) \xrightarrow[\mathcal{A}]{a} (q_0, 1) \xrightarrow[\mathcal{A}]{a} (q_0, 2) \xrightarrow[\mathcal{A}]{b} (q_1, 2) \xrightarrow[\mathcal{A}]{a} (q_1, 1) \xrightarrow[\mathcal{A}]{a} (q_1, 0) \xrightarrow[\mathcal{A}]{a} (q_2, 0).$$

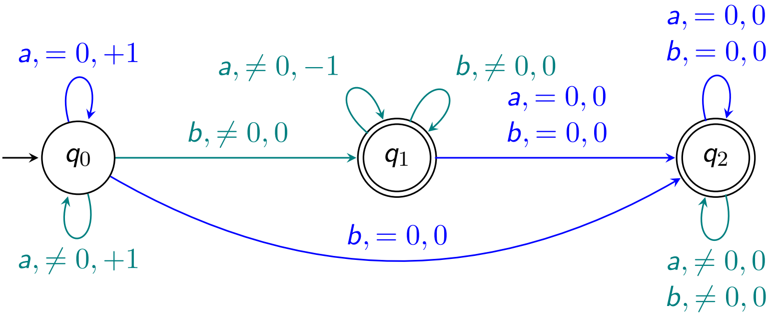


Figure 2: An ROCA $\mathcal{A}$.

$$(q_0, 0) \xrightarrow[\mathcal{A}]{a} (q_0, 1) \xrightarrow[\mathcal{A}]{a} (q_0, 2) \xrightarrow[\mathcal{A}]{b} (q_1, 2) \xrightarrow[\mathcal{A}]{a} (q_1, 1) \xrightarrow[\mathcal{A}]{a} (q_1, 0) \xrightarrow[\mathcal{A}]{a} (q_2, 0).$$

$$\mathcal{L}(\mathcal{A}) = \{a^n b (b^* a)^n (a|b)^* \mid n \geq 0\}.$$

Let $\mathcal{A}$ be an ROCA accepting L. We study the equivalence relation $\equiv$ induced by $\mathcal{A}$ over Σ^*.

Let $u, v \in \Sigma^*$. We say that $u \equiv v$ if and only if

Let $\mathcal{A}$ be an ROCA accepting L. We study the equivalence relation $\equiv$ induced by $\mathcal{A}$ over Σ^*.
Let $u, v \in \Sigma^*$. We say that $u \equiv v$ if and only if

1. $\forall w \in \Sigma^*, uw \in L \Leftrightarrow vw \in L$, and

Let $\mathcal{A}$ be an ROCA accepting L. We study the equivalence relation $\equiv$ induced by $\mathcal{A}$ over Σ^*.

Let $u, v \in \Sigma^*$. We say that $u \equiv v$ if and only if

1. $\forall w \in \Sigma^*, uw \in L \Leftrightarrow vw \in L$, and
2. $\forall w \in \Sigma^*, uw, vw \in \mathit{Pref}(L) \Rightarrow c_{\mathcal{A}}(uw) = c_{\mathcal{A}}(vw)$.

Let $\mathcal{A}$ be an ROCA accepting L. We study the equivalence relation $\equiv$ induced by $\mathcal{A}$ over Σ^*.
Let $u, v \in \Sigma^*$. We say that $u \equiv v$ if and only if

1. $\forall w \in \Sigma^*, uw \in L \Leftrightarrow vw \in L$, and
2. $\forall w \in \Sigma^*, uw, vw \in \mathit{Pref}(L) \Rightarrow c_{\mathcal{A}}(uw) = c_{\mathcal{A}}(vw)$.

For example, let $L = \{a^n b (b^* a)^n (a|b)^* \mid n \geq 0\}$. Then, $b \equiv abba$ but $ab \not\equiv aab$.

Let $\mathcal{A}$ be an ROCA accepting L. Using the relation $\equiv$, we can construct an infinite deterministic automaton accepting L: the behavior graph of $\mathcal{A}$ $BG(\mathcal{A}) = (Q_\equiv, \Sigma, \delta_\equiv, q^0_\equiv, F_\equiv)$ with:

- $Q_\equiv = \{[\![u]\!]_\equiv \mid u \in \mathit{Pref}(L)\}$,
- $q^0_\equiv = [\![\varepsilon]\!]_\equiv$,
- $F_\equiv = \{[\![u]\!]_\equiv \mid u \in L\}$, and
- $\delta_\equiv : Q \times \Sigma \to Q$ such that $\delta([\![u]\!]_\equiv, a) = [\![ua]\!]_\equiv$ with $a \in \Sigma$ and $u, ua \in \mathit{Pref}(L)$.

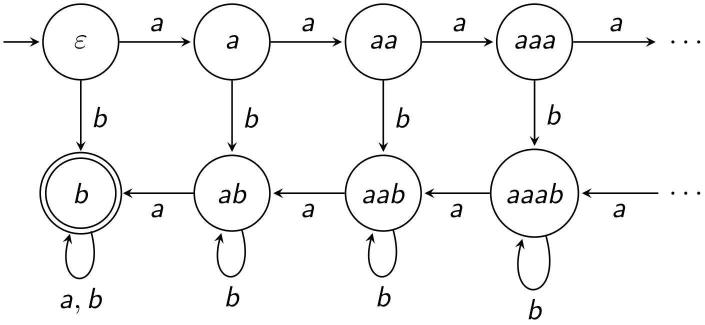


Figure 3: The behavior graph of $\mathcal{A}$.

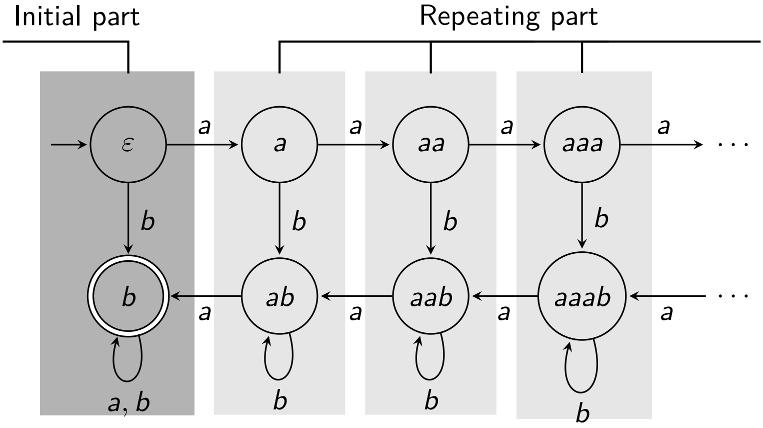


Figure 3: The behavior graph of $\mathcal{A}$.

Theorem 2

Let $\mathcal{A}$ be an ROCA accepting L and $BG(\mathcal{A})$ be its behavior graph. Then, $BG(\mathcal{A})$ is ultimately periodic.

Theorem 2

Let $\mathcal{A}$ be an ROCA accepting L and $BG(\mathcal{A})$ be its behavior graph. Then, $BG(\mathcal{A})$ is ultimately periodic.
Moreover, it is possible to construct an ROCA accepting L from $BG(\mathcal{A})$.

Let $\mathcal{A}$ be an ROCA accepting L.

[3] Based on the algorithm for VCA [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].

Let $\mathcal{A}$ be an ROCA accepting L.

- Rough idea[3]: learn a sufficiently large initial fragment of $BG(\mathcal{A})$ and construct an ROCA from it.

[3] Based on the algorithm for VCA [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].

Let $\mathcal{A}$ be an ROCA accepting L.

- Rough idea[3]: learn a sufficiently large initial fragment of $BG(\mathcal{A})$ and construct an ROCA from it.
- What is an initial fragment?
  $\hookrightarrow$ $BG_\ell(\mathcal{A})$ is a subgraph of $BG(\mathcal{A})$ whose set of states is $\{[\![u]\!]_\equiv \in Q_\equiv \mid \forall x \in Pref(u), 0 \leq c_\mathcal{A}(x) \leq \ell\}$, with $\ell \in \mathbb{N}$. Let $L_\ell = \mathcal{L}(BG_\ell(\mathcal{A}))$.

[3] Based on the algorithm for VCA [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].

Let $\mathcal{A}$ be an ROCA accepting L.

- Rough idea[3]: learn a sufficiently large initial fragment of $BG(\mathcal{A})$ and construct an ROCA from it.
- What is an initial fragment?
  $\hookrightarrow$ $BG_\ell(\mathcal{A})$ is a subgraph of $BG(\mathcal{A})$ whose set of states is $\{[\![u]\!]_\equiv \in Q_\equiv \mid \forall x \in \mathit{Pref}(u), 0 \leq c_\mathcal{A}(x) \leq \ell\}$, with $\ell \in \mathbb{N}$. Let $L_\ell = \mathcal{L}(BG_\ell(\mathcal{A}))$.
- How to construct an ROCA from $BG_\ell(\mathcal{A})$?
  $\hookrightarrow$ Not the focus here but it is possible.

[3] Based on the algorithm for VCA [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].

Let $\mathcal{A}$ be an ROCA accepting L.

- Rough idea[3]: learn a sufficiently large initial fragment of $BG(\mathcal{A})$ and construct an ROCA from it.
- What is an initial fragment?
  $\hookrightarrow$ $BG_\ell(\mathcal{A})$ is a subgraph of $BG(\mathcal{A})$ whose set of states is $\{[\![u]\!]_\equiv \in Q_\equiv \mid \forall x \in \mathit{Pref}(u), 0 \leq c_\mathcal{A}(x) \leq \ell\}$, with $\ell \in \mathbb{N}$. Let $L_\ell = \mathcal{L}(BG_\ell(\mathcal{A}))$.
- How to construct an ROCA from $BG_\ell(\mathcal{A})$?
  $\hookrightarrow$ Not the focus here but it is possible.
- How to learn $BG_\ell(\mathcal{A})$?
  $\hookrightarrow$ $BG_\ell(\mathcal{A})$ is actually a DFA.

[3] Based on the algorithm for VCA [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].

Figure 4: Adaptation of Angluin's framework for ROCAs.

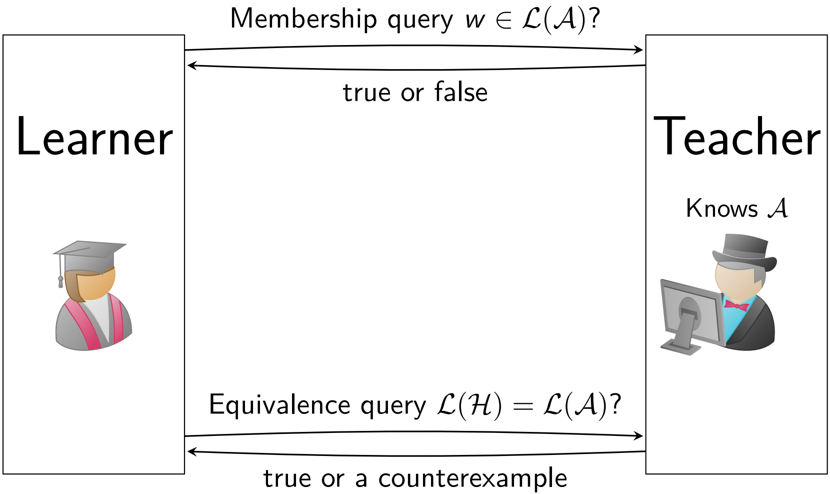


Figure 4: Adaptation of Angluin's framework for ROCAs.

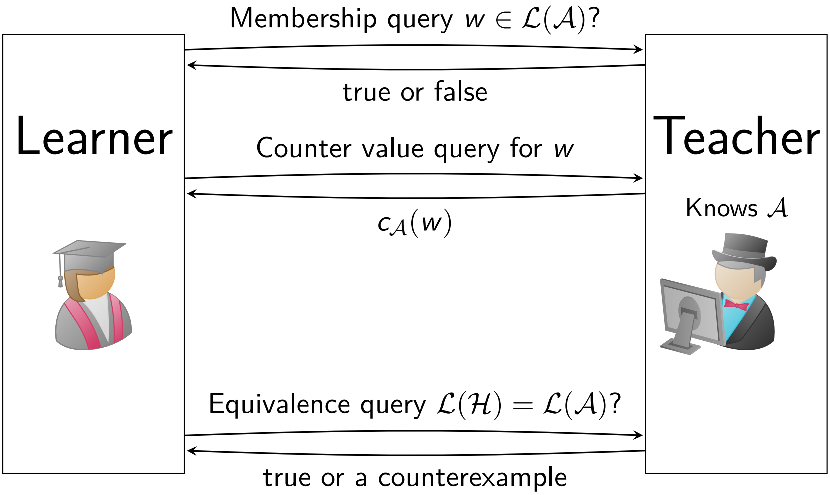


Figure 4: Adaptation of Angluin's framework for ROCAs.

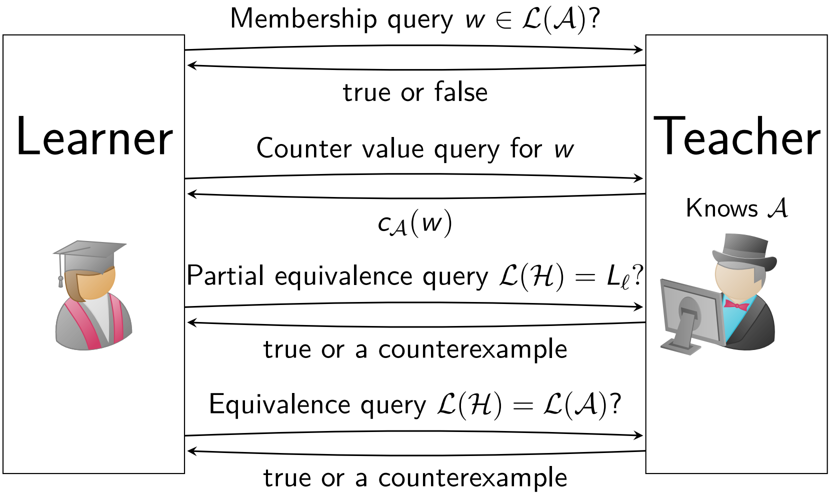


Figure 4: Adaptation of Angluin's framework for ROCAs.

Algorithm 2 Adaptation of L^* for ROCAs.

Require: A teacher knowing an ROCA $\mathcal{A}$

Ensure: An ROCA accepting the same language is returned

```
Initialize the data structure D_ℓ up to ℓ = 0
while true do
    Make D_ℓ respect the needed constraints and construct A_{D_ℓ}
    Ask a partial equivalence query over A_{D_ℓ}
    if the answer is negative then
        Update D_ℓ with the provided counterexample    ▷ ℓ is not modified
    else
        Construct all the possible ROCAs A_1, ..., A_n from A_{D_ℓ}
        Ask an equivalence query over each A_i
        if the answer is true for an A_i then return A_i
        else Select one counterexample and update D_ℓ    ▷ ℓ is increased
```

Let $\mathcal{A}$ be an ROCA accepting $L \subseteq \Sigma^*$.
An observation table up to ℓ is a tuple $\mathscr{O}_\ell = (R, S, \widehat{S}, \mathcal{L}_\ell, \mathcal{C}_\ell)$ with:

- $R \subseteq \Sigma^*$ is the prefix-closed set of representatives,
- $S \subseteq \widehat{S} \subseteq \Sigma^*$ are two suffix-closed sets of separators,
- $\mathcal{L}_\ell : (R \cup R\Sigma)\widehat{S} \to \{0, 1\}$, and
- $\mathcal{C}_\ell : (R \cup R\Sigma)S \to \{0, \dots, \ell\} \cup \{\bot\}$.

Let $\mathcal{A}$ be an ROCA accepting $L \subseteq \Sigma^*$.
An observation table up to ℓ is a tuple $\mathscr{O}_\ell = (R, S, \widehat{S}, \mathcal{L}_\ell, \mathcal{C}_\ell)$ with:

- $R \subseteq \Sigma^*$ is the prefix-closed set of representatives,
- $S \subseteq \widehat{S} \subseteq \Sigma^*$ are two suffix-closed sets of separators,
- $\mathcal{L}_\ell : (R \cup R\Sigma)\widehat{S} \to \{0, 1\}$, and
- $\mathcal{C}_\ell : (R \cup R\Sigma)S \to \{0, \dots, \ell\} \cup \{\bot\}$.

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

- $\forall s \in \widehat{S}, \mathcal{L}_\ell(us) = 1$ if and only if $us \in L_\ell$.
- $\forall s \in S, \mathcal{C}_\ell(us) = \begin{cases} c_{\mathcal{A}}(us) & \text{if } us \in \mathit{Pref}(\mathscr{O}_\ell) \\ \bot & \text{otherwise.} \end{cases}$

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

	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	$0, \perp$
$abaa$	$1, 0$
$abab$	$1, 0$
$abba$	$1, 0$
$abbba$	$1, 0$
$abbbb$	$0, \perp$

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

$\hookrightarrow$ Getting the algorithm to eventually finish is harder than it looks.

Theorem 3

Let $\mathcal{A}$ be an ROCA accepting a language $L \subseteq \Sigma^*$. Given a teacher for L with an automaton $\mathcal{A}$, and t the length of the longest counterexample for (partial) equivalence queries:

Theorem 3

Let $\mathcal{A}$ be an ROCA accepting a language $L \subseteq \Sigma^*$. Given a teacher for L with an automaton $\mathcal{A}$, and t the length of the longest counterexample for (partial) equivalence queries:

- *An ROCA accepting L can be computed in time and space exponential in $|\mathcal{A}|, |\Sigma|$ and t.*

Theorem 3

Let $\mathcal{A}$ be an ROCA accepting a language $L \subseteq \Sigma^*$. Given a teacher for L with an automaton $\mathcal{A}$, and t the length of the longest counterexample for (partial) equivalence queries:

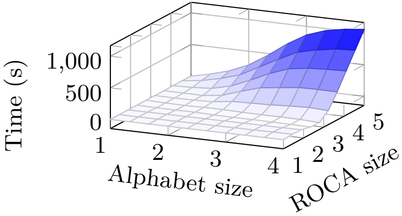
- *An ROCA accepting L can be computed in time and space exponential in $|\mathcal{A}|, |\Sigma|$ and t.*
- *The learner asks:*
  - $\mathcal{O}(t^3)$ *partial equivalence queries*
  - $\mathcal{O}(|\mathcal{A}|t^2)$ *equivalence queries*
  - *An exponential number of membership (resp. counter value) queries in $|\mathcal{A}|, |\Sigma|$, and t.*

1. Motivation

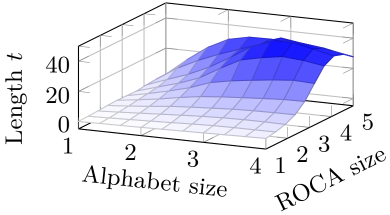
2. Learning deterministic finite automata

3. Learning realtime one-counter automata

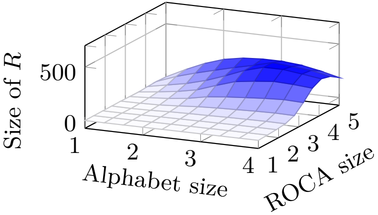
4. Experimental results



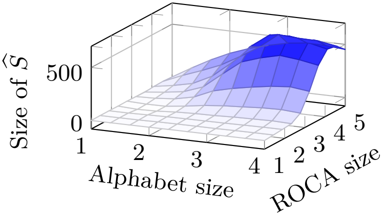
(a) Total time.



(b) Length t of the longest counterexample.



(c) Final size of R.



(d) Final size of $\widehat{S}$.

Figure 5: Experimental results for randomly generated ROCAs.

Angluin, Dana. “Learning Regular Sets from Queries and Counterexamples”. In: *Inf. Comput.* 75.2 (1987), pp. 87–106. DOI: 10.1016/0890-5401(87)90052-6. URL: https://doi.org/10.1016/0890-5401(87)90052-6.

Chitic, Cristiana and Daniela Rosu. “On Validation of XML Streams Using Finite State Machines”. In: *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*. Ed. by Sihem Amer-Yahia and Luis Gravano. ACM, 2004, pp. 85–90. DOI: 10.1145/1017074.1017096. URL: https://doi.org/10.1145/1017074.1017096.

Hopcroft, John E. and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation, Second Edition*. Addison-Wesley, 2000.

References II

Neider, Daniel and Christof Löding. *Learning visibly one-counter automata in polynomial time*. Tech. rep. Technical Report AIB-2010-02, RWTH Aachen (January 2010), 2010.