Learning Realtime One-Counter Automata

Published 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

October 26, 2022

1. Motivation
2. Learning deterministic finite automata
3. Learning realtime one-counter automata
4. Experimental results

1. Motivation

2. Learning deterministic finite automata

3. Learning realtime one-counter automata

4. Experimental results

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```

- An object is an unordered collection of key-value pairs.

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```

- An object is an unordered collection of key-value pairs.
- An array is an ordered collection of values.

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```

- An object is an unordered collection of key-value pairs.
- An array is an ordered collection of values.

Here, let us fix an order on the keys inside an object. That is, we can assume objects are ordered.

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```

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

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

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```

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

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

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

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```


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

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

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

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

```
{ "title": "Learning ROCAs",
  "place": {
    "city": "Brussels",
    "country": "Belgium"
  },
  "authors": ["Bruyère", "Pérez", "Staquet"]
}
```

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

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

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

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

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

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]
  - For VCAs, we deduce the counter value from the word itself.

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]
  - For VCAs, we deduce the counter value from the word itself.
  - For ROCAs, we need an automaton.

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]
  - For VCAs, we deduce the counter value from the word itself.
  - For ROCAs, we need an automaton.
- We showed similarity between ROCAs and VCAs.

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]
  - For VCAs, we deduce the counter value from the word itself.
  - For ROCAs, we need an automaton.
- We showed similarity between ROCAs and VCAs.
  - VCA's algorithm is used as a sort of sub-routine.

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]
  - For VCAs, we deduce the counter value from the word itself.
  - For ROCAs, we need an automaton.
- We showed similarity between ROCAs and VCAs.
  - VCA's algorithm is used as a sort of sub-routine.
- We extend data structure to take into account the counter value.

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]
  - For VCAs, we deduce the counter value from the word itself.
  - For ROCAs, we need an automaton.
- We showed similarity between ROCAs and VCAs.
  - VCA's algorithm is used as a sort of sub-routine.
- We extend data structure to take into account the counter value.
  - Some values are unknown and left as wildcards.

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

Overview:

- Based on learning algorithm for visibly one-counter automata (VCA).[1]
  - For VCAs, we deduce the counter value from the word itself.
  - For ROCAs, we need an automaton.
- We showed similarity between ROCAs and VCAs.
  - VCA's algorithm is used as a sort of sub-routine.
- We extend data structure to take into account the counter value.
  - Some values are unknown and left as wildcards.
  - Obtaining an hypothesis is harder than for VCAs.

[1] Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010.

1. Motivation

2. Learning deterministic finite automata
   - Deterministic finite automaton
   - Active learning
   - Data structure: the observation table

3. Learning realtime one-counter automata

4. Experimental results

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

- Σ is the alphabet,

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

- Σ is the alphabet,
- Q is the set of states,

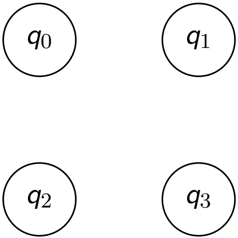


Figure 1: A DFA $\mathcal{A}$.

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

- Σ is the alphabet,
- Q is the set of states,
- $\delta : Q \times \Sigma \to Q$ is the transition function.

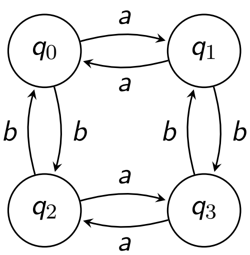


Figure 1: A DFA $\mathcal{A}$.

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

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

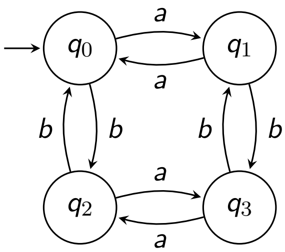


Figure 1: A DFA $\mathcal{A}$.

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

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

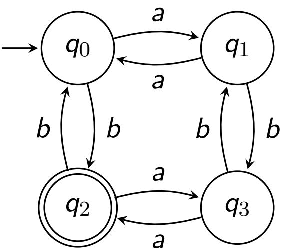


Figure 1: A DFA $\mathcal{A}$.

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

$$p_1 \xrightarrow[\mathcal{A}]{a_1} p_2 \xrightarrow[\mathcal{A}]{a_2} \dots \xrightarrow[\mathcal{A}]{a_n} p_{n+1}$$

such that $p_1 = q_0$ and $\forall i, \delta(p_i, a_i) = p_{i+1}$.

Example 1

Soit $w = ababb$. Its run is

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_2 \xrightarrow{b} q_0 \xrightarrow{b} q_2.$$

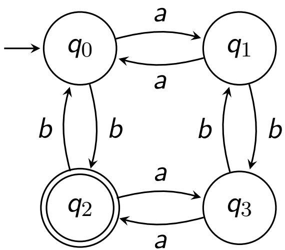


Figure 1: A DFA $\mathcal{A}$.

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

$$p_1 \xrightarrow[\mathcal{A}]{a_1} p_2 \xrightarrow[\mathcal{A}]{a_2} \dots \xrightarrow[\mathcal{A}]{a_n} p_{n+1}$$

such that $p_1 = q_0$ and $\forall i, \delta(p_i, a_i) = p_{i+1}$. If $p_{n+1} \in F$, the run is said accepting.

Example 1

Soit $w = ababb$. Its run is

$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_3 \xrightarrow{a} q_2 \xrightarrow{b} q_0 \xrightarrow{b} q_2,$$

and w is accepted by $\mathcal{A}$.

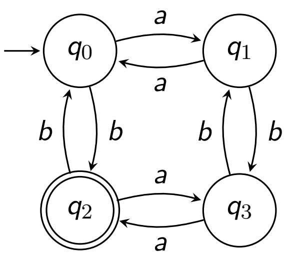


Figure 1: A DFA $\mathcal{A}$.

The language of $\mathcal{A}$ is the set

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

Example 2

The language of $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}) = \{w \mid w \text{ a an even number of } a \text{ and an odd number of } b\}.$$

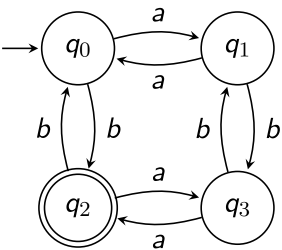


Figure 1: A DFA $\mathcal{A}$.

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.

$\hookrightarrow$ $\underbrace{\text{active}}$ learning algorithm.

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

$\hookrightarrow$ $\underbrace{\text{active}}_{\substack{\text{queries}\\\text{information}}}$ learning algorithm.

Figure 2: Angluin's framework Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

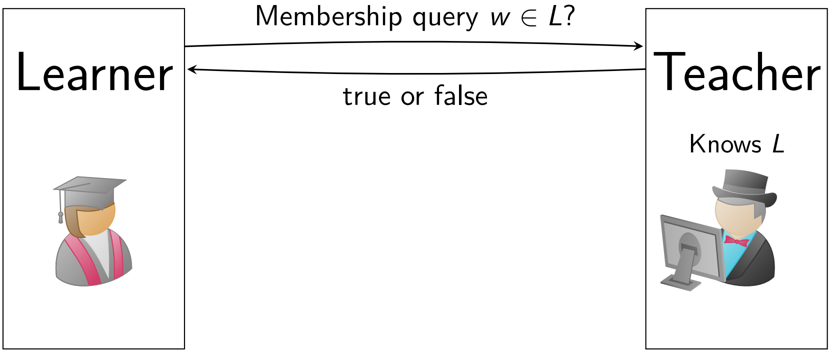


Figure 2: Angluin's framework Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

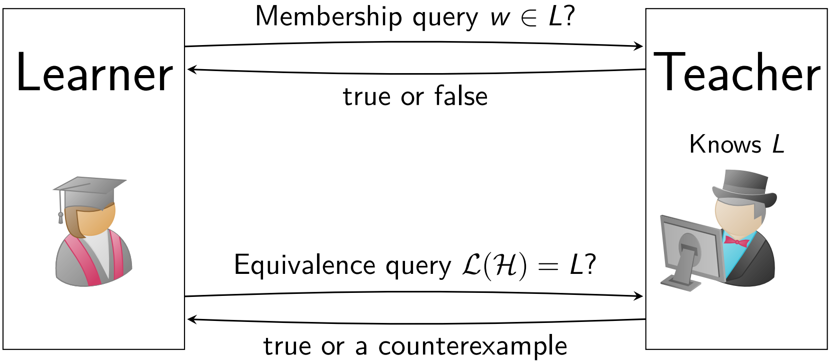


Figure 2: Angluin's framework Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.
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 $L = \{w \in \{a,b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

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

Let $L = \{w \in \{a,b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

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

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 $L = \{w \in \{a,b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

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

Proposition 3

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

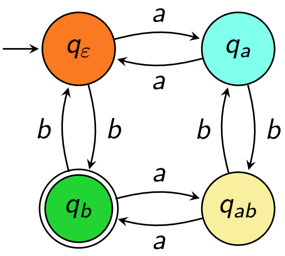
Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

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

The Myhill-Nerode congruence of this table has a finite index.

Let $L = \{w \in \{a, b\}^* \mid$ w has an even number of a and an odd number of $b\}$.

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



The Myhill-Nerode congruence of this table has a finite index.

An observation table[2] is a tuple $\mathscr{O} = (R, S, \mathcal{L})$ where:

- $R \subseteq \Sigma^*$ is a prefix-closed set of representatives (the lines),
- $S \subseteq \Sigma^*$ is a suffix-closed set of separators (the columns),
- $\mathcal{L} : (R \cup R\Sigma)S \to \{1, 0\}$ is such that $\forall u \in R \cup R\Sigma, s \in S, \mathcal{L}(us) = 1 \Leftrightarrow us \in L$.

[2] Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987.

An observation table[2] is a tuple $\mathcal{O} = (R, S, \mathcal{L})$ where:

- $R \subseteq \Sigma^*$ is a prefix-closed set of representatives (the lines),
- $S \subseteq \Sigma^*$ is a suffix-closed set of separators (the columns),
- $\mathcal{L} : (R \cup R\Sigma)S \to \{1, 0\}$ is such that $\forall u \in R \cup R\Sigma, s \in S, \mathcal{L}(us) = 1 \Leftrightarrow us \in L$.

Let $u, v \in R \cup R\Sigma$. We say that $u \sim_{\mathcal{O}} v$ if and only if

$$\forall s \in S, \mathcal{L}(us) = \mathcal{L}(vs).$$

[2] Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987.

An observation table[2] is a tuple $\mathcal{O} = (R, S, \mathcal{L})$ where:

- $R \subseteq \Sigma^*$ is a prefix-closed set of representatives (the lines),
- $S \subseteq \Sigma^*$ is a suffix-closed set of separators (the columns),
- $\mathcal{L} : (R \cup R\Sigma)S \to \{1, 0\}$ is such that $\forall u \in R \cup R\Sigma, s \in S, \mathcal{L}(us) = 1 \Leftrightarrow us \in L$.

Let $u, v \in R \cup R\Sigma$. We say that $u \sim_{\mathcal{O}} v$ if and only if

$$\forall s \in S, \mathcal{L}(us) = \mathcal{L}(vs).$$

The goal is to have a sufficient large finite subset of the infinite table from before.
More precisely, we refine $\sim_{\mathcal{O}}$ until it coincides with $\sim$.

[2] Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987.

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε
ε	0
a	0
b	1

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε
ε	0
a	0
b	1

An observation table is closed if

$$\forall u \in R\Sigma, \exists v \in R, u \sim_{\mathscr{O}} v.$$

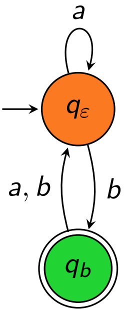
If unclosed due to $u \in R\Sigma$, add u to R.
Here, unclosed due to b.

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε
ε	0
b	1
a	0
ba	0
bb	0

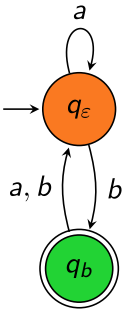
Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε
ε	0
b	1
a	0
ba	0
bb	0



Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε
ε	0
b	1
a	0
ba	0
bb	0



Counterexample: ab.

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε
ε	0
b	1
a	0
ab	0
ba	0
bb	0
aa	0
aba	1
abb	0

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε
ε	0
b	1
a	0
ab	0
ba	0
bb	0
aa	0
aba	1
abb	0

An observation table is Σ-consistent if

$$\forall u, v \in R, \forall a \in \Sigma, u \sim_{\mathscr{O}} v \Rightarrow ua \sim_{\mathscr{O}} va.$$

If Σ-inconsistent due to $u \sim_{\mathscr{O}} v$ but $\mathcal{L}(uaw) \neq \mathcal{L}(vaw)$, add aw to S.
Here, $\varepsilon \sim_{\mathscr{O}} ab$ but $\mathcal{L}(\varepsilon \cdot a \cdot \varepsilon) = 0 \neq \mathcal{L}(ab \cdot a \cdot \varepsilon) = 1$.

Let $L = \{w \in \{a,b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε	a
ε	0	0
b	1	0
a	0	0
ab	0	1
ba	0	1
bb	0	0
aa	0	0
aba	1	0
abb	0	0

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε	a
ε	0	0
b	1	0
a	0	0
ab	0	1
ba	0	1
bb	0	0
aa	0	0
aba	1	0
abb	0	0

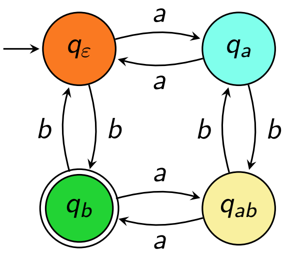
$\varepsilon \sim_{\mathscr{O}} a$ but $\mathcal{L}(\varepsilon \cdot b \cdot \varepsilon) = 1 \neq \mathcal{L}(a \cdot b \cdot \varepsilon) = 0$.

Let $L = \{w \in \{a, b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε	a	b
ε	0	0	1
b	1	0	0
a	0	0	0
ab	0	1	0
ba	0	1	0
bb	0	0	1
aa	0	0	1
aba	1	0	0
abb	0	0	0

Let $L = \{w \in \{a,b\}^* \mid$
w has an even number of a and an odd number of $b\}$.

	ε	a	b
ε	0	0	1
b	1	0	0
a	0	0	0
ab	0	1	0
ba	0	1	0
bb	0	0	1
aa	0	0	1
aba	1	0	0
abb	0	0	0



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 observation table $\mathscr{O}$
Fill $\mathscr{O}$ with membership queries
while true **do**
    Make $\mathscr{O}$ closed and Σ-consistent
    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, add $Pref(w)$ to $\mathscr{O}$
        Fill $\mathscr{O}$ with membership queries

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:

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

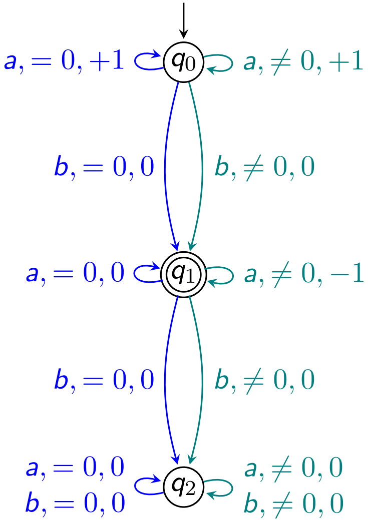


Figure 3: An ROCA $\mathcal{A}$.

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 $(q, n) \in Q \times \mathbb{N}$.

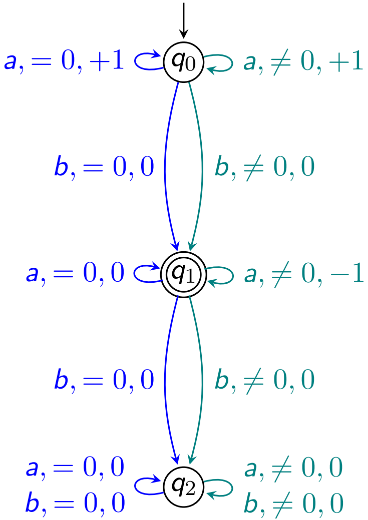


Figure 3: An ROCA $\mathcal{A}$.

The transition relation

$\xrightarrow[\mathcal{A}]{} \subseteq (Q \times \mathbb{N}) \times \Sigma \times (Q \times \mathbb{N})$

contains $(q, n) \xrightarrow[\mathcal{A}]{a} (p, m)$ iff

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

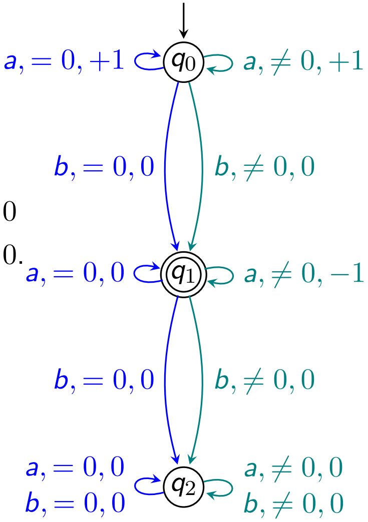


Figure 4: An ROCA $\mathcal{A}$.

The transition relation

$$\xrightarrow[\mathcal{A}]{} \subseteq (Q \times \mathbb{N}) \times \Sigma \times (Q \times \mathbb{N})$$

contains $(q, n) \xrightarrow[\mathcal{A}]{a} (p, m)$ iff

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

Example 4

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

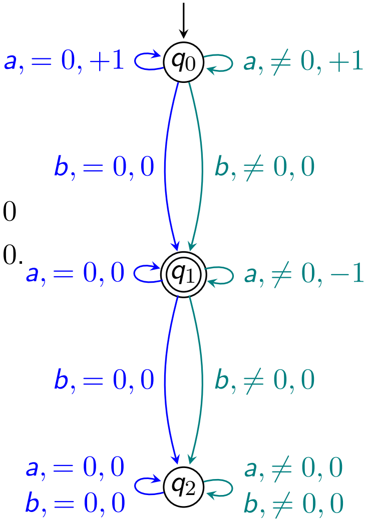


Figure 4: An ROCA $\mathcal{A}$.

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

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

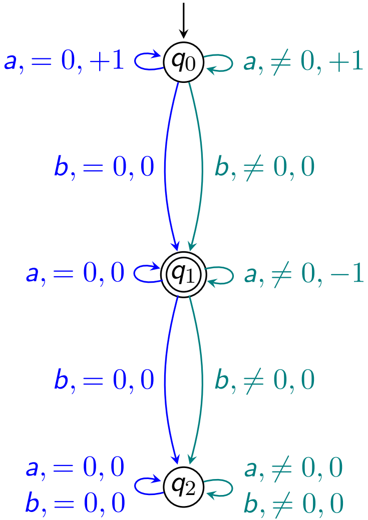


Figure 5: An ROCA $\mathcal{A}$.

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

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

Example 5

Since $(q_0, 0) \xrightarrow[\mathcal{A}]{aabaaa} (q_1, 0)$, $c_{\mathcal{A}}(aabaaa) = 0$.

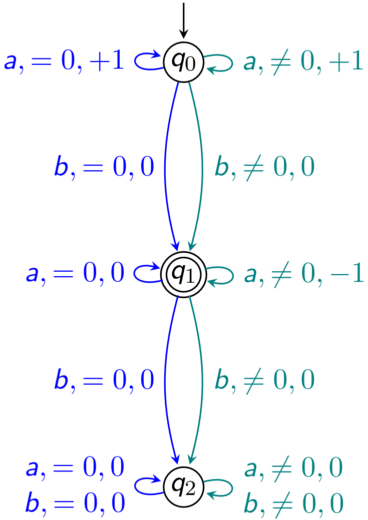


Figure 5: An ROCA $\mathcal{A}$.

For a word w, if we have

$$(q_0, 0) \xrightarrow[\mathcal{A}]{w} (q, 0)$$

with $q \in F$, then $w \in \mathcal{L}(\mathcal{A})$.

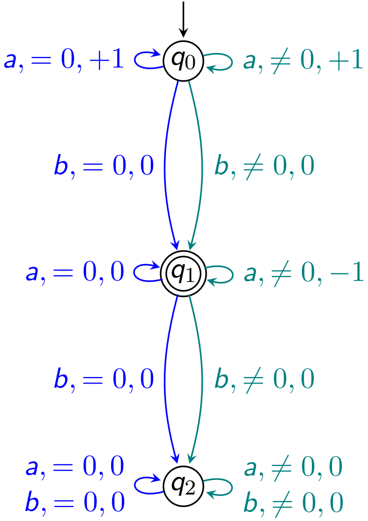


Figure 6: An ROCA $\mathcal{A}$.

For a word w, if we have

$$(q_0, 0) \xrightarrow[\mathcal{A}]{w} (q, 0)$$

with $q \in F$, then $w \in \mathcal{L}(\mathcal{A})$.

Example 6

$\mathcal{L}(\mathcal{A}) = \{a^n b a^m \mid 0 \leq n \leq m\}$.

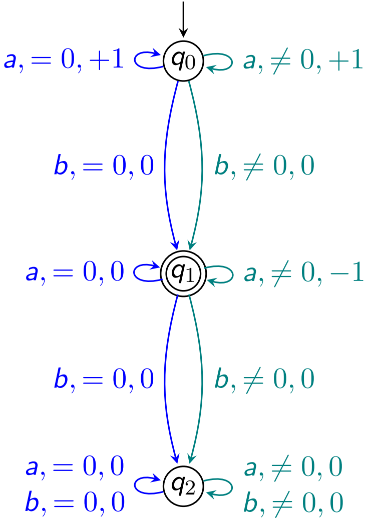


Figure 6: An ROCA $\mathcal{A}$.

Let $\mathcal{A}$ be an ROCA accepting L.
Let $u, v \in \Sigma^*$. We say that $u \equiv v$ iff

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

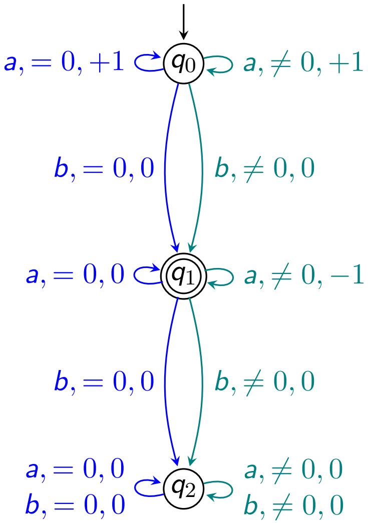


Figure 7: An ROCA $\mathcal{A}$.

Let $\mathcal{A}$ be an ROCA accepting L.
Let $u, v \in \Sigma^*$. We say that $u \equiv v$ iff

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

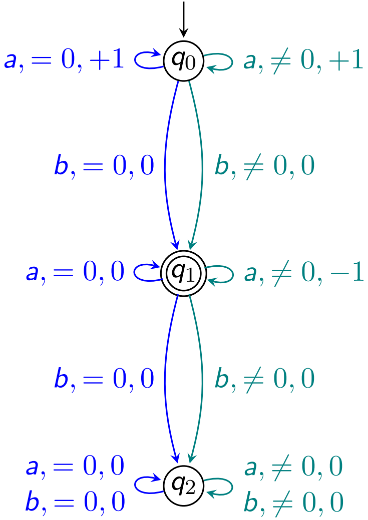


Figure 7: An ROCA $\mathcal{A}$.

Let $\mathcal{A}$ be an ROCA accepting L.
Let $u, v \in \Sigma^*$. We say that $u \equiv v$ iff

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

Example 7

$b \equiv aba$ but $ab \not\equiv aab$.

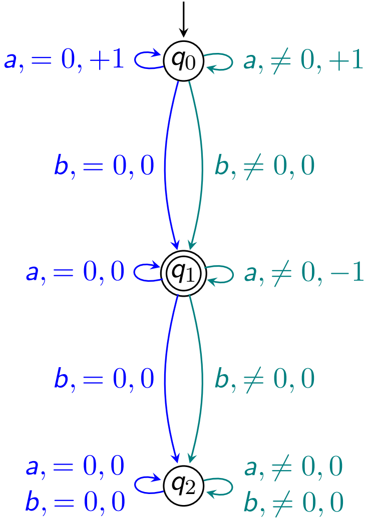


Figure 7: An ROCA $\mathcal{A}$.

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 $u, ua \in \mathit{Pref}(L)$.

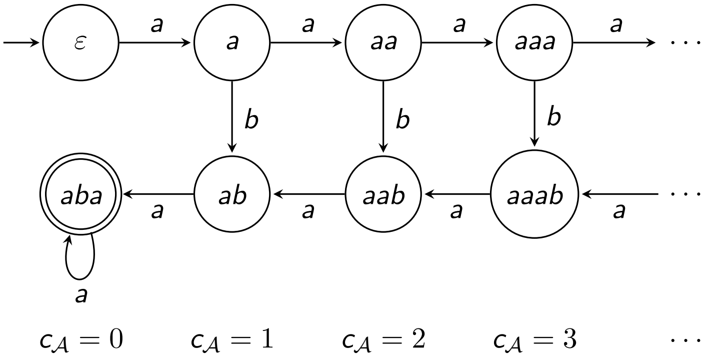


Figure 8: The behavior graph of $\mathcal{A}$.

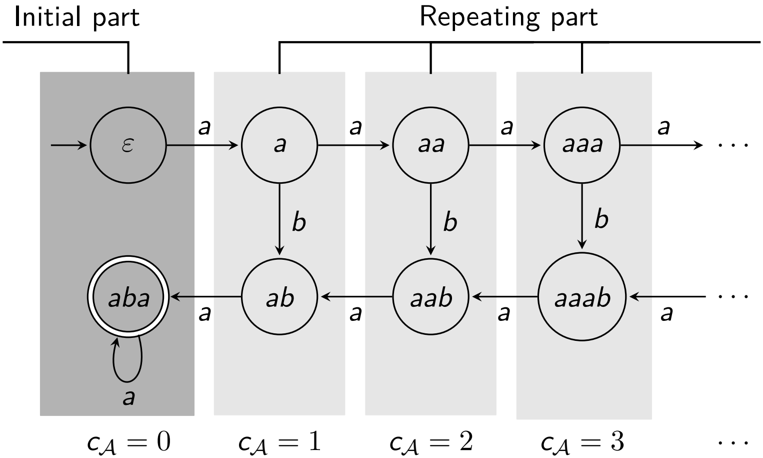


Figure 8: The behavior graph of $\mathcal{A}$.

Theorem 8

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

Theorem 8

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 $\{[\![u]\!]_\equiv \in Q_\equiv \mid \forall x \in \mathit{Pref}(u), 0 \leq c_{\mathcal{A}}(x) \leq \ell\}$, with $\ell \in \mathbb{N}$. Let $L_\ell = \mathcal{L}(BG_\ell(\mathcal{A}))$.

[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 9: Adaptation of Angluin's framework for ROCAs.

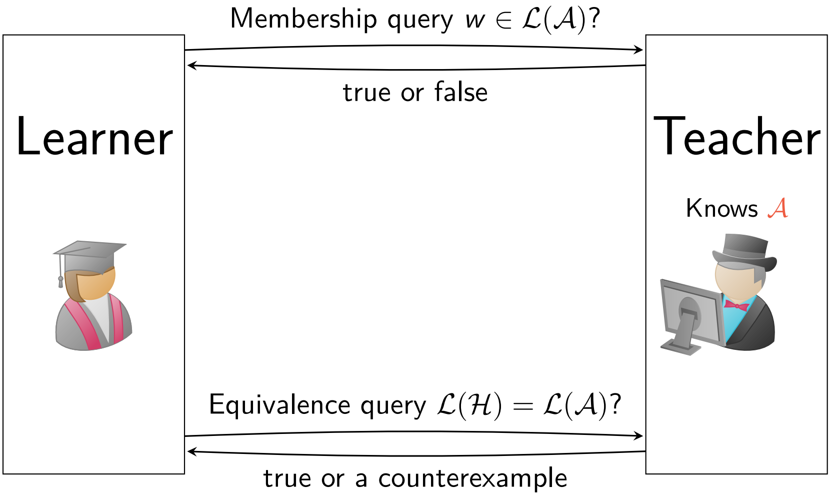


Figure 9: Adaptation of Angluin's framework for ROCAs.

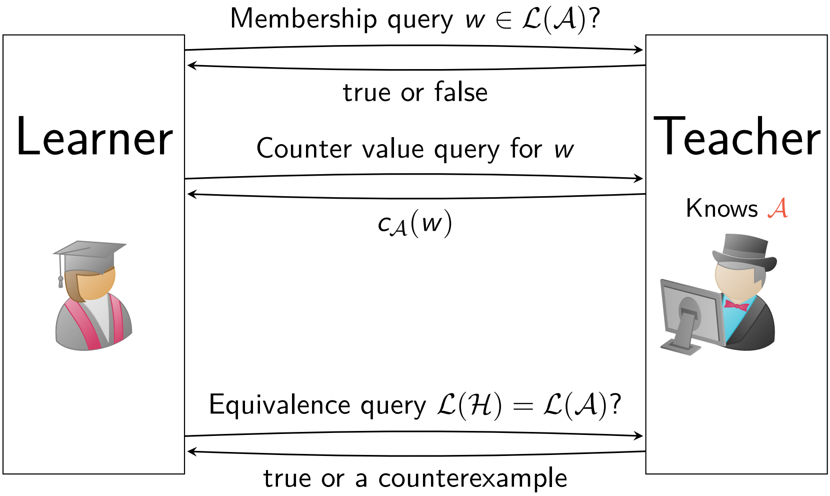


Figure 9: Adaptation of Angluin's framework for ROCAs.

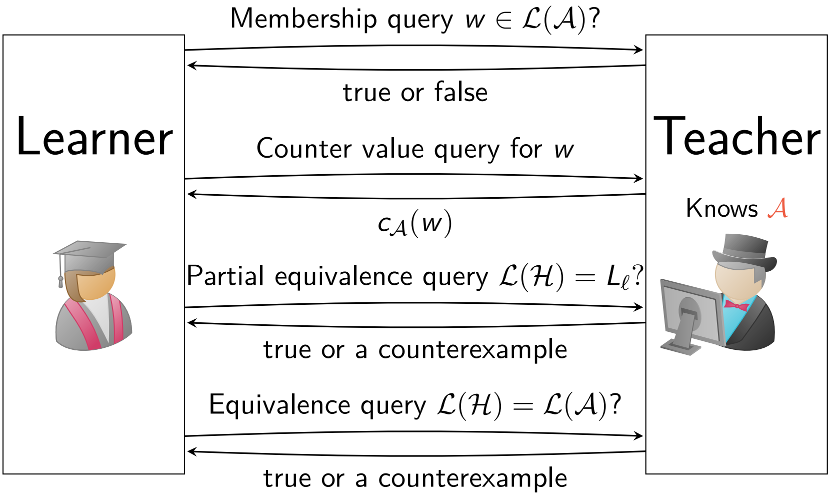


Figure 9: Adaptation of Angluin's framework for ROCAs.

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
- $\mathcal{C}_\ell : (R \cup R\Sigma)S \to \{0, \dots, \ell\} \cup \{\perp\}$.

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

Let $L = \{a^n b a^m \mid 0 \leq n \leq m\}$.

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

Let $L = \{a^n b a^m \mid 0 \leq n \leq m\}$.

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

Let $L = \{a^n b a^m \mid 0 \leq n \leq m\}$.

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

Let $L = \{a^n b a^m \mid 0 \leq n \leq m\}$.

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

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

Rough idea:

- u and v are approximately equivalent if they are equal, up to $\bot$.

Rough idea:

- u and v are approximately equivalent if they are equal, up to $\perp$.
  - If $u \equiv v$, then u and v are approximately equivalent.

Rough idea:

- u and v are approximately equivalent if they are equal, up to $\perp$.
    - If $u \equiv v$, then u and v are approximately equivalent.
    - Not a right-congruence.

Rough idea:

- u and v are approximately equivalent if they are equal, up to $\perp$.
  - If $u \equiv v$, then u and v are approximately equivalent.
  - Not a right-congruence.
- Adapt definitions of closedness and consistency to force right-congruence.

Theorem 9

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 9

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 9

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:

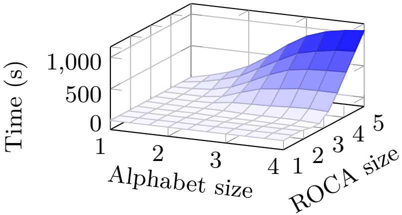
- *An ROCA accepting L can be computed in time and space exponential in $|\mathcal{A}|, |\Sigma|$ and t.*
- *The learner asks:*
    - *$\mathcal{O}(t^3)$ partial equivalence queries*
    - *$\mathcal{O}(|\mathcal{A}|t^2)$ equivalence queries*
    - *An exponential number of membership (resp. counter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

1. Motivation

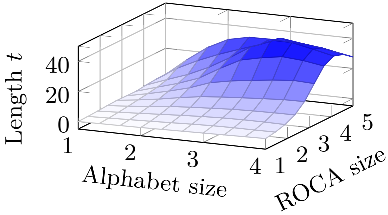
2. Learning deterministic finite automata

3. Learning realtime one-counter automata

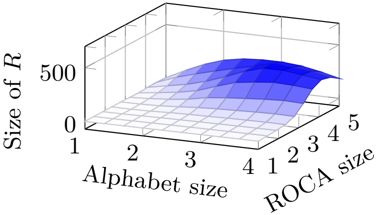
4. Experimental results



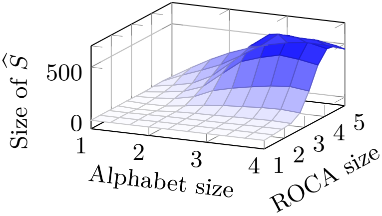
(a) Total time.



(b) Length t of the longest counterexample.



(c) Final size of R.



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

Thank your your attention!
Any questions?

References I

- 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.