Learning Realtime One-Counter Automata

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

April 5, 2022

1. Motivation: model checking
2. Learning a DFA
3. Realtime one-counter automata
4. Learning a realtime one-counter automaton
5. Implementation and future work

Checking that a computer system works as intended is "hard".

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

Checking that a computer system works as intended is “hard”.

$\hookrightarrow$ We must abstract the system into a model.

[1] Angluin, “Learning Regular Sets from Queries and Counterexamples”, 1987.

Checking that a computer system works as intended is “hard”.

$\hookrightarrow$ We must abstract the system into a model.

Constructing the abstraction by hand can lead to more bugs.

[1] Angluin, “Learning Regular Sets from Queries and Counterexamples”, 1987.

Checking that a computer system works as intended is "hard".

$\hookrightarrow$ We must abstract the system into a model.

Constructing the abstraction by hand can lead to more bugs.

$\hookrightarrow$ We want a model that is complex enough to correctly abstract the system and simple enough to be learned, i.e., automatically constructed from the system.

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

Checking that a computer system works as intended is “hard”.

$\hookrightarrow$ We must abstract the system into a model.

Constructing the abstraction by hand can lead to more bugs.

$\hookrightarrow$ We want a model that is complex enough to correctly abstract the system and simple enough to be learned, i.e., automatically constructed from the system.

$\hookrightarrow$ The family of deterministic finite automata (DFAs) which can be learned by an active learning algorithm, such as L^*.[1]

[1] Angluin, “Learning Regular Sets from Queries and Counterexamples”, 1987.

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

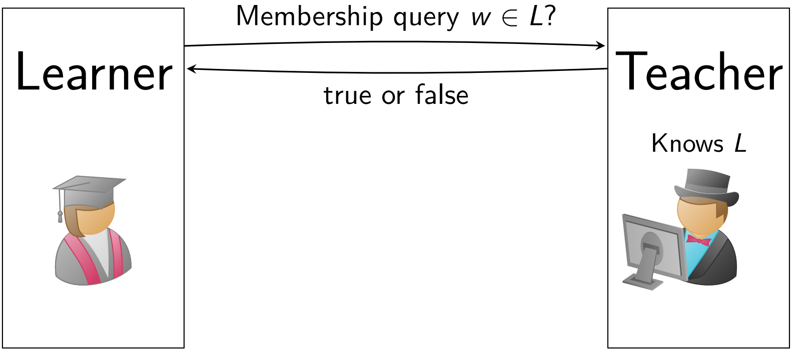


Figure 1: Angluin's framework [Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987].

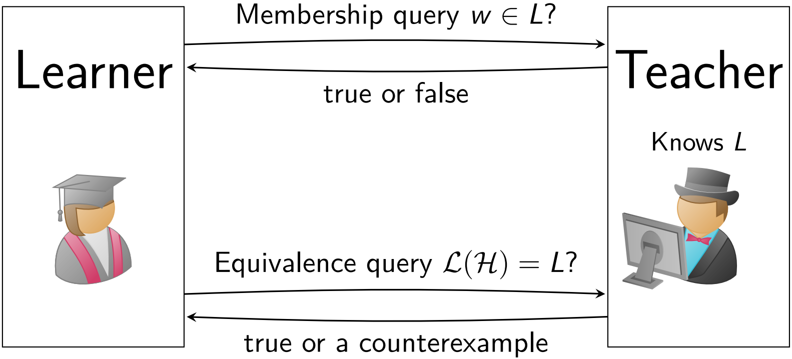


Figure 1: Angluin's framework [Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987].

Main ideas for L^*:

Main ideas for L^*:

- A data structure is built through membership queries.

Main ideas for L^*:

- A data structure is built through membership queries.
- Once it satisfies some properties, we build a DFA $\mathcal{H}$ from it.

Main ideas for L^*:

- A data structure is built through membership queries.
- Once it satisfies some properties, we build a DFA $\mathcal{H}$ from it.
- We ask an equivalence query over $\mathcal{H}$.

Main ideas for L^*:

- A data structure is built through membership queries.
- Once it satisfies some properties, we build a DFA $\mathcal{H}$ from it.
- We ask an equivalence query over $\mathcal{H}$.
- If the answer is true, we are done. Otherwise, we update the data structure and repeat.

Main ideas for L^*:

- A data structure is built through membership queries.
- Once it satisfies some properties, we build a DFA $\mathcal{H}$ from it.
- We ask an equivalence query over $\mathcal{H}$.
- If the answer is true, we are done. Otherwise, we update the data structure and repeat.

While DFAs can be used in practice, they lack expressivity. For instance, a DFA cannot count the number of times a state is seen.

Main ideas for L^*:

- A data structure is built through membership queries.
- Once it satisfies some properties, we build a DFA $\mathcal{H}$ from it.
- We ask an equivalence query over $\mathcal{H}$.
- If the answer is true, we are done. Otherwise, we update the data structure and repeat.

While DFAs can be used in practice, they lack expressivity. For instance, a DFA cannot count the number of times a state is seen.

$\hookrightarrow$ We add a natural counter.

A realtime one-counter automaton (ROCA) is a tuple $\mathcal{A} = (Q, \Sigma, \delta_{=0}, \delta_{>0}, q_0, F)$ with:

- Q is the set of states,
- Σ is the alphabet,
- $\delta_{=0}$ and $\delta_{>0}$ are the transition functions:

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

- q_0 is the initial state, and
- $F \subseteq Q$ is the set of accepting states.

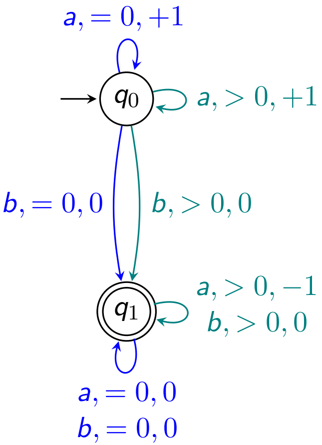


Figure 2: A realtime one-counter automaton.

An ROCA defines a configuration graph where states are $Q \times \mathbb{N}$.

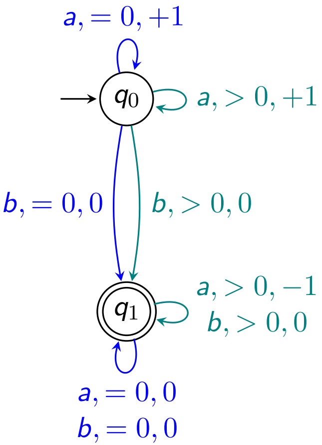


Figure 2: A realtime one-counter automaton.

An ROCA defines a configuration graph where states are $Q \times \mathbb{N}$.
We have the following run for *aba*:

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

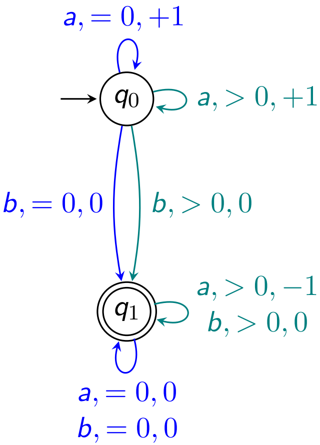


Figure 2: A realtime one-counter automaton.

An ROCA defines a configuration graph where states are $Q \times \mathbb{N}$.
We have the following run for *aba*:

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

The counter value of *ab* according to $\mathcal{A}$ is 1, noted $c_{\mathcal{A}}(ab) = 1$.

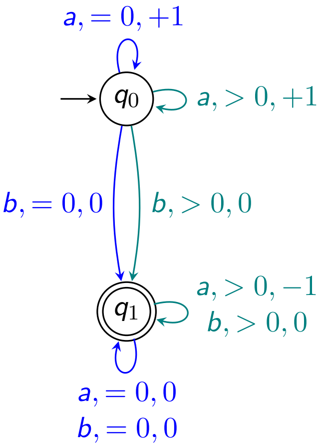


Figure 2: A realtime one-counter automaton.

An ROCA defines a configuration graph where states are $Q \times \mathbb{N}$.
We have the following run for *aba*:

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

The counter value of *ab* according to $\mathcal{A}$ is 1, noted $c_{\mathcal{A}}(ab) = 1$.
Since $q_1 \in F$ and $c_{\mathcal{A}}(aba) = 0$, *aba* is accepted by $\mathcal{A}$.

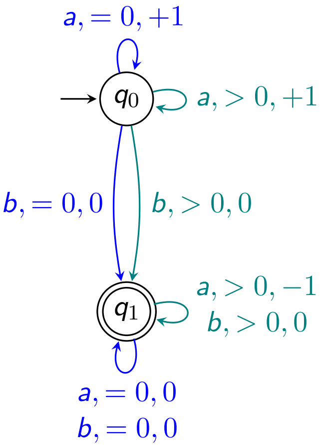


Figure 2: A realtime one-counter automaton.

An ROCA defines a configuration graph where states are $Q \times \mathbb{N}$.
We have the following run for *aba*:

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

The counter value of *ab* according to $\mathcal{A}$ is 1, noted $c_{\mathcal{A}}(ab) = 1$.
Since $q_1 \in F$ and $c_{\mathcal{A}}(aba) = 0$, *aba* is accepted by $\mathcal{A}$.
We can show that the language of $\mathcal{A}$ is

$$\mathcal{L}(\mathcal{A}) = \{a^n b (b^* a)^n \{a, b\}^* \mid n \in \mathbb{N}\}.$$

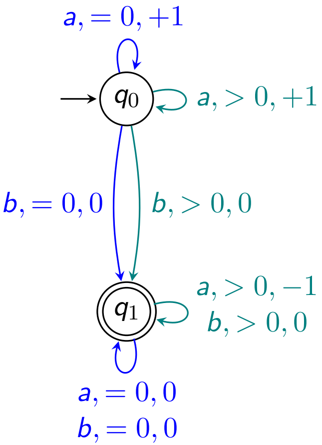


Figure 2: A realtime one-counter automaton.

Theorem 1

Let L be the language of some ROCA $\mathcal{A}$. It is possible to learn an ROCA accepting L in an exponential time and space complexities in $|Q|$ and $|\Sigma|$.

What do we want to learn exactly?

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

What do we want to learn exactly?

For DFAs, we learn an equivalence relation called the Myhill-Nerode congruence, from which we can construct the minimal DFA accepting the target language.

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

What do we want to learn exactly?

For DFAs, we learn an equivalence relation called the Myhill-Nerode congruence, from which we can construct the minimal DFA accepting the target language.

Let $\mathcal{A}$ be an ROCA accepting L, and $u, v \in \Sigma^*$. We say that $u \equiv v$ if and only if $\forall w \in \Sigma^*$, we have[2]

$$uw \in L \Leftrightarrow vw \in L,$$
$$uw, vw \in Pref(L) \Rightarrow c_{\mathcal{A}}(uw) = c_{\mathcal{A}}(vw).$$

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

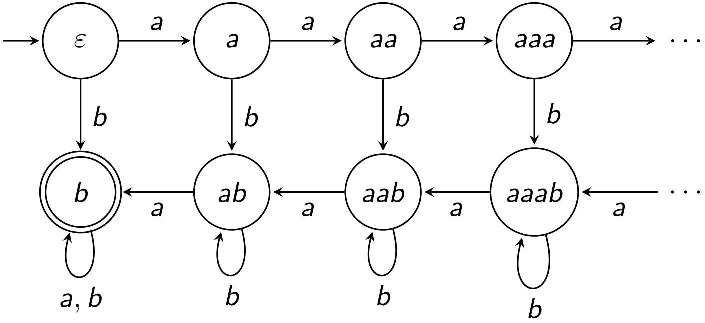


Figure 3: A behavior graph constructed from $\equiv$.

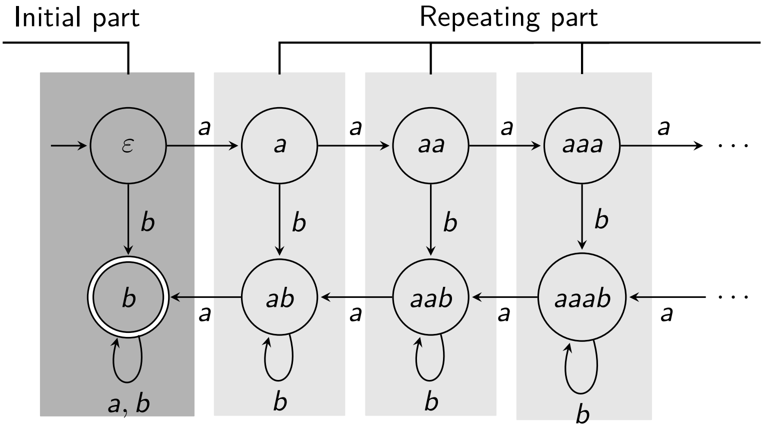


Figure 3: A behavior graph constructed from $\equiv$.

Lemma 2

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

[3] Inspired by Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010

Lemma 2

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

We fix a counter limit ℓ and we learn the minimal DFA that accepts L up to ℓ, denoted by L_ℓ.[3]

[3] Inspired by Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010

Lemma 2

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

We fix a counter limit ℓ and we learn the minimal DFA that accepts L up to ℓ, denoted by L_ℓ.[3]

Lemma 3

Let $\mathcal{A}$ be an ROCA accepting L, $BG(\mathcal{A})$ be its behavior graph, ℓ be a counter limit, and $\mathcal{H}$ be the minimal DFA accepting L_ℓ. Then, if ℓ is large enough, the initial fragments of $BG(\mathcal{A})$ and $\mathcal{H}$ are isomorphic.

[3] Inspired by Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010

Lemma 2

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

We fix a counter limit ℓ and we learn the minimal DFA that accepts L up to ℓ, denoted by L_ℓ.[3]

Lemma 3

Let $\mathcal{A}$ be an ROCA accepting L, $BG(\mathcal{A})$ be its behavior graph, ℓ be a counter limit, and $\mathcal{H}$ be the minimal DFA accepting L_ℓ. Then, if ℓ is large enough, the initial fragments of $BG(\mathcal{A})$ and $\mathcal{H}$ are isomorphic.
Moreover, if ℓ is large enough, it is possible to construct an ROCA accepting L from $\mathcal{H}$.

[3] Inspired by Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010

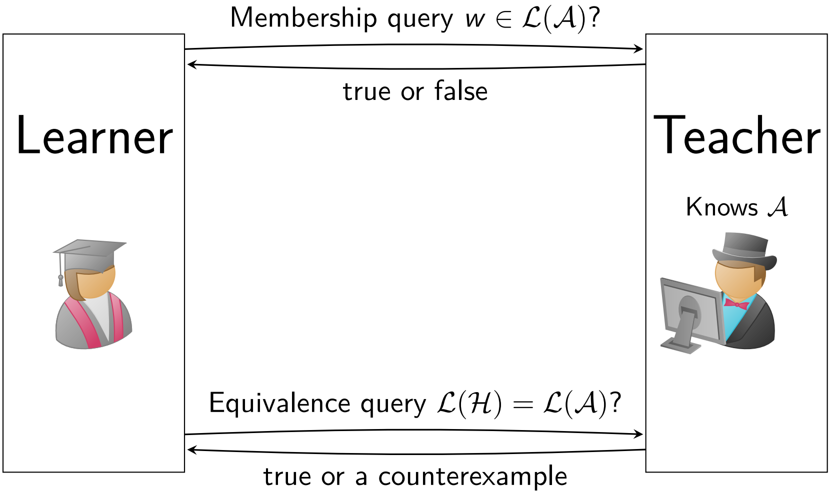


Figure 4: Adaptation of Angluin's framework for ROCAs.

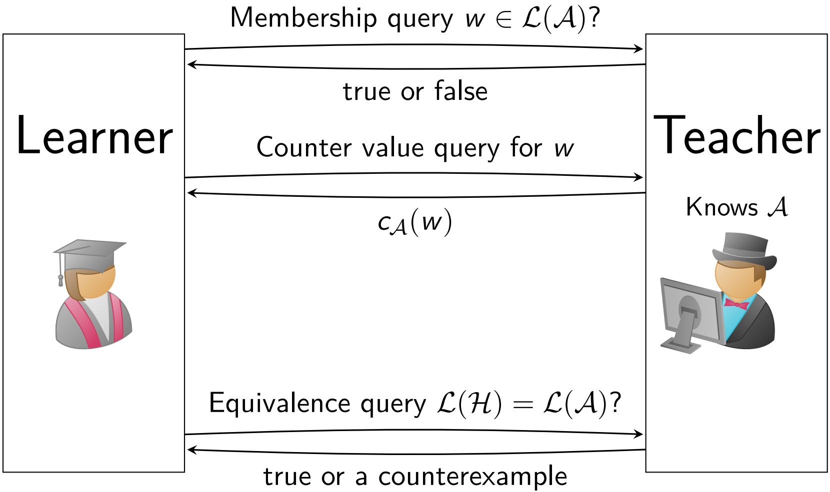


Figure 4: Adaptation of Angluin's framework for ROCAs.

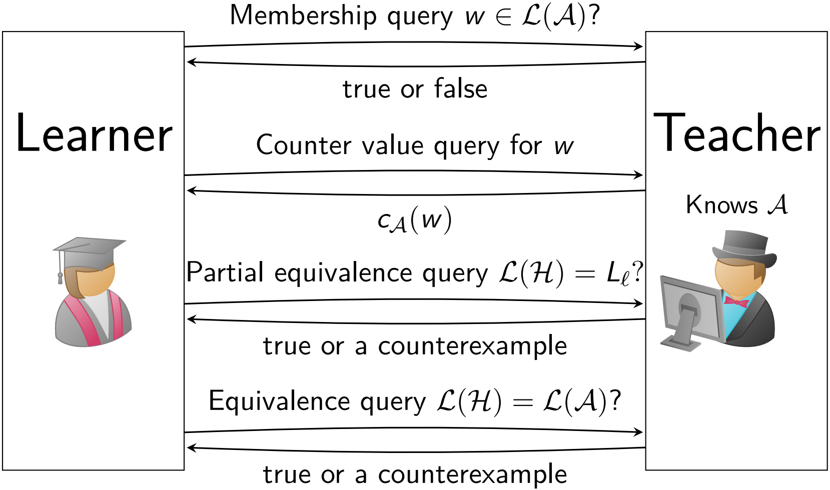


Figure 4: Adaptation of Angluin's framework for ROCAs.

Theorem 4

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, where t is the length of the longest counterexample returned by the teacher on (partial) equivalence queries.

Theorem 4

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, where t is the length of the longest counterexample returned by the teacher on (partial) equivalence queries.

The learner asks

- $\mathcal{O}(t^3)$ *partial equivalence queries,*

Theorem 4

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, where t is 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*

Theorem 4

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, where t is 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 (resp. counter value) queries which is exponential in $|Q|, |\Sigma|$ and t.*

Main difficulties:

Main difficulties:

- Counter value queries are required, unlike in [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].

Main difficulties:

- Counter value queries are required, unlike in [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].
- Unlike in L^*, the data structure must store the counter values and requires two sets of separators.

Main difficulties:

- Counter value queries are required, unlike in [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].
- Unlike in L^*, the data structure must store the counter values and requires two sets of separators.
- Obtaining an hypothesis $\mathcal{H}$ from the data structure is not trivial, due to the counter values and the two sets.

Main difficulties:

- Counter value queries are required, unlike in [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].
- Unlike in L^*, the data structure must store the counter values and requires two sets of separators.
- Obtaining an hypothesis $\mathcal{H}$ from the data structure is not trivial, due to the counter values and the two sets.
- Proving that the structure eventually satisfies the constraints we want is a hard task.

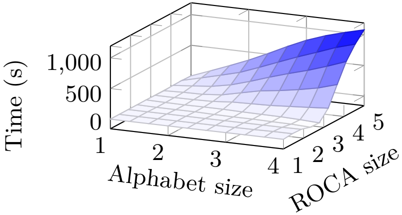
Main difficulties:

- Counter value queries are required, unlike in [Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010].
- Unlike in L^*, the data structure must store the counter values and requires two sets of separators.
- Obtaining an hypothesis $\mathcal{H}$ from the data structure is not trivial, due to the counter values and the two sets.
- Proving that the structure eventually satisfies the constraints we want is a hard task.
  - For instance, the algorithm never stops if we have a single set of separators.

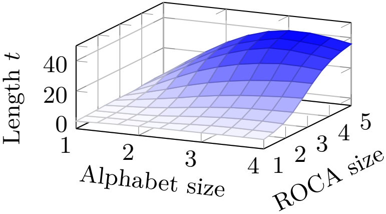
We implemented our algorithm in Java using AutomataLib and LearnLib.

We evaluated the performance on two types of benchmarks:

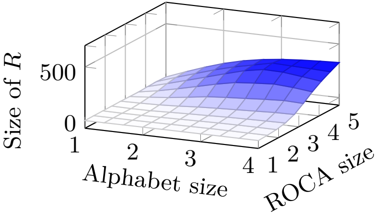
1. On randomly generated ROCAs.
2. On JSON documents.



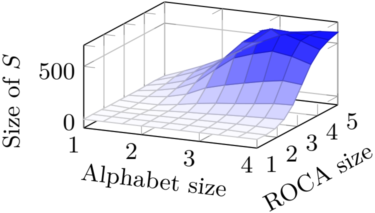
(a) Total time.



(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.

For the JSON based benchmarks, the teacher has a JSON schema which details how a document should be structured.

```
{
    "type": "object",
    "properties": {
        "subList": {
            "type": "array",
            "items": {"$ref": "#"}
        }
    }
}
```

Listing 1: A JSON schema.

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

Table 1: Results for JSON documents.

For future work:

- Remove partial equivalence queries by working with more recent learning algorithms, such as TTT by Isberner et al.[4] or $L^{\#}$ by Vaandrager et al.[5]
- Lowering the complexity.

Currently, we are working on extending the use-case on JSON documents to be usable in practice.

[4] Isberner, Howar, and Steffen, "The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning", 2014.

[5] Vaandrager et al., "A New Approach for Active Automata Learning Based on Apartness", 2021.

- 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.
- Isberner, Malte, Falk Howar, and Bernhard Steffen. "The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning". In: *Runtime Verification - 5th International Conference, RV 2014, Toronto, ON, Canada, September 22-25, 2014. Proceedings*. Ed. by Borzoo Bonakdarpour and Scott A. Smolka. Vol. 8734. Lecture Notes in Computer Science. Springer, 2014, pp. 307–322. DOI: 10.1007/978-3-319-11164-3_26. URL: https://doi.org/10.1007/978-3-319-11164-3%5C_26.
- 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.

Vaandrager, Frits W. et al. “A New Approach for Active Automata Learning Based on Apartness”. In: *CoRR* abs/2107.05419 (2021). arXiv: 2107.05419. URL: https://arxiv.org/abs/2107.05419.