Active Learning of Automata with Resources

Private PhD Defense

Gaëtan Staquet

Theoretical computer science
University of Mons

Formal Techniques in Software Engineering
University of Antwerp

June 19, 2024

Part I – Preliminaries

Base definitions

Question. How to **automatically** construct a model from a **black-box** system?

$\hookrightarrow$ **Active automata learning**.

Question. How to **automatically** construct a model from a **black-box** system?

$\hookrightarrow$ **Active automata learning**.

But simple finite automata are not expressive enough.

$\hookrightarrow$ Extend automata with **resources**.

Question. How to **automatically** construct a model from a **black-box** system?

$\hookrightarrow$ **Active automata learning**.

But simple finite automata are not expressive enough.

$\hookrightarrow$ Extend automata with **resources**.

Goals of the thesis:

- New learning algorithms for automata extended with
  - a **counter** (Part 2),
  - **timers** (Part 4).
- Validation algorithm relying on learning an automaton with a **stack** (Part 3).

Question. How to **automatically** construct a model from a **black-box** system?

$\hookrightarrow$ **Active automata learning**.

But simple finite automata are not expressive enough.

$\hookrightarrow$ Extend automata with **resources**.

Goals of the thesis:

- New learning algorithms for automata extended with
  - a **counter** (Part 2),
  - **timers** (Part 4).
- Validation algorithm relying on learning an automaton with a **stack** (Part 3).

Structure for today:

- Recall L^* learning algorithm.
- In each part, present the main theorem and focus on one property. Also, focus on theory; experimental results are ignored.

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

- Σ is the **alphabet**,

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

- Σ is the **alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,

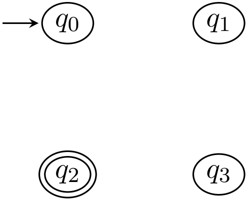


Figure 1: A DFA.

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

- Σ is the **alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,
- $\delta : Q \times \Sigma \to Q$ is the **transition function**.

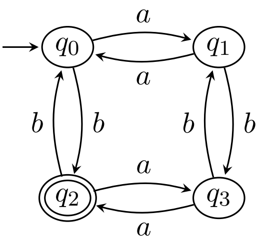


Figure 1: A DFA.

Question. How to construct a DFA from a **black-box** system?

$\hookrightarrow$ L^* algorithm.

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

Question. How to construct a DFA from a **black-box** system?

$\hookrightarrow$ L^* algorithm.

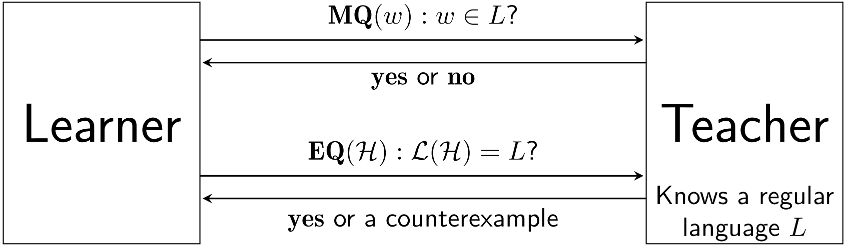


Figure 2: Angluin's framework.[1]

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

An **observation table** for a language L is a tuple $\mathscr{O} = (R, S, T)$ where:

- $R \subsetneq \Sigma^*$ is the **finite** set of **representatives**,
- $S \subsetneq \Sigma^*$ is the **finite** set of **separators**,
- $T : (R \cup R\Sigma) \cdot S \to \{\mathbf{yes}, \mathbf{no}\}$ is such that

$$T(u \cdot s) = \begin{cases} \mathbf{yes} & \text{if } u \cdot s \in L \\ \mathbf{no} & \text{if } u \cdot s \notin L. \end{cases}$$

$\rightsquigarrow$ Membership queries.

	ε	b
ε	**no**	**yes**
b	**yes**	**no**
a	**no**	**no**
ba	**no**	**no**
bb	**no**	**yes**
aa	**no**	**yes**
ab	**no**	**no**

Figure 3: An observation table.

An **observation table** for a language L is a tuple $\mathscr{O} = (R, S, T)$ where:

- $R \subsetneq \Sigma^*$ is the **finite** set of **representatives**,
- $S \subsetneq \Sigma^*$ is the **finite** set of **separators**,
- $T : (R \cup R\Sigma) \cdot S \to \{\mathbf{yes}, \mathbf{no}\}$ is such that

$$T(u \cdot s) = \begin{cases} \mathbf{yes} & \text{if } u \cdot s \in L \\ \mathbf{no} & \text{if } u \cdot s \notin L. \end{cases}$$

$\rightsquigarrow$ Membership queries.

Equivalence relation over the **(extended) representatives**:

$$\forall u, v \in R \cup R\Sigma : u \equiv_{\mathscr{O}} v \Leftrightarrow \forall s \in S : T(u \cdot s) = T(v \cdot s).$$

	ε	b
ε	**no**	**yes**
b	**yes**	**no**
a	**no**	**no**
ba	**no**	**no**
bb	**no**	**yes**
aa	**no**	**yes**
ab	**no**	**no**

Figure 3: An observation table.

Question. When is it possible to construct a DFA from $\equiv_{\mathcal{O}}$?

Question. When is it possible to construct a DFA from $\equiv_{\mathscr{O}}$?

The table must be **closed**:

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

	ε
ε	**no**
a	**no**
b	**yes**

Figure 4: A table that is **not** closed, due to b.

Question. When is it possible to construct a DFA from $\equiv_{\mathscr{O}}$?

The table must be **closed**:

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

	ε
ε	**no**
b	**yes**
a	**no**
ba	**no**
bb	**no**

Figure 5: A closed table.

Question. When is it possible to construct a DFA from $\equiv_{\mathscr{O}}$?

The table must be **closed**:

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

	ε
ε	**no**
b	**yes**
a	**no**
ba	**no**
bb	**no**

Figure 5: A closed table.

The table must be **Σ-consistent**:

$$\forall u, v \in R, a \in \Sigma : u \equiv_{\mathscr{O}} v \Rightarrow u \cdot a \equiv_{\mathscr{O}} v \cdot a.$$

	ε
ε	**no**
b	**yes**
a	**no**
ba	**no**
bb	**no**
aa	**no**
ab	**no**

Figure 6: A table that is **not** Σ-consistent, due to $\varepsilon \equiv_{\mathscr{O}} a$ but $b \not\equiv_{\mathscr{O}} ab$.

Question. When is it possible to construct a DFA from $\equiv_{\mathscr{O}}$?

The table must be **closed**:

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

	ε
ε	**no**
b	**yes**
a	**no**
ba	**no**
bb	**no**

Figure 5: A closed table.

The table must be **Σ-consistent**:

$$\forall u, v \in R, a \in \Sigma : u \equiv_{\mathscr{O}} v \Rightarrow u \cdot a \equiv_{\mathscr{O}} v \cdot a.$$

	ε	b
ε	**no**	**yes**
b	**yes**	**no**
a	**no**	**no**
ba	**no**	**no**
bb	**no**	**yes**
aa	**no**	**yes**
ab	**no**	**no**

Figure 7: A Σ-consistent table.

Theorem 1 (Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987). *Let $\mathcal{A}$ be the* ***minimal*** *DFA accepting the target language L, and ℓ be the length of the* ***longest*** *counterexample provided by the teacher. Then,*

- *the L^* algorithm eventually terminates,*
- *in time and space* ***polynomial*** *in $|\mathcal{A}|, \ell$, and $|\Sigma|$,*
- *with at most $|\mathcal{A}|$ equivalence queries and $\mathcal{O}\left(\ell \cdot |\mathcal{A}|^2\right)$ membership queries.*

Part II – Learning Realtime One-Counter Automata

Bruyère, Pérez, and Staquet, "Learning Realtime One-Counter Automata", TACAS, 2022

A **realtime one-counter automaton** (**ROCA**, for short) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where:

- Σ is the **alphabet**,

A **realtime one-counter automaton** (**ROCA**, for short) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where:

- Σ is the **alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,

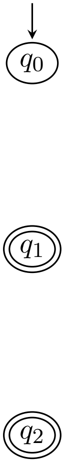


Figure 8: An ROCA.

A **realtime one-counter automaton** (**ROCA**, for short) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where:

- Σ is the **alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,
- $\delta : Q \times \Sigma \times \{=0, >0\} \to Q \times \{+1, -1, 0\}$ is the **transition function**.

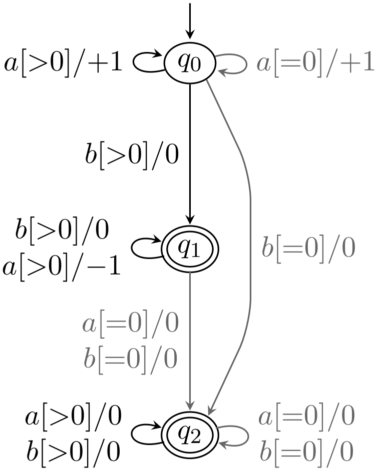


Figure 8: An ROCA.

A **realtime one-counter automaton** (**ROCA**, for short) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where:

- Σ is the **alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,
- $\delta : Q \times \Sigma \times \{=0, >0\} \to Q \times \{+1, -1, 0\}$ is the **transition function**.

$\rightsquigarrow$ **Counted** runs, *e.g.*,

$$(q_0, 0) \xrightarrow{a} (q_0, 1) \xrightarrow{b} (q_1, 1) \xrightarrow{a} (q_1, 0).$$

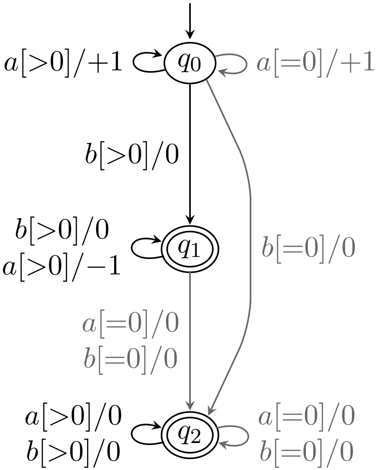


Figure 8: An ROCA.

A **realtime one-counter automaton** (**ROCA**, for short) is a tuple $\mathcal{A} = (\Sigma, Q, q_0, F, \delta)$ where:

- Σ is the **alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,
- $\delta : Q \times \Sigma \times \{=0, >0\} \to Q \times \{+1, -1, 0\}$ is the **transition function**.

$\leadsto$ **Counted** runs, *e.g.*,

$$(q_0, 0) \xrightarrow{a} (q_0, 1) \xrightarrow{b} (q_1, 1) \xrightarrow{a} (q_1, 0).$$

A counted run is **accepting** when last state is in F **and** counter value is 0.

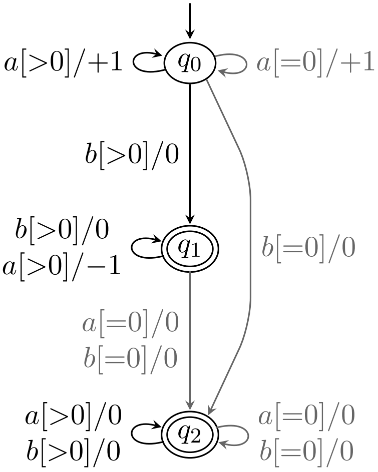


Figure 8: An ROCA.

Definition 2. *Given an ROCA $\mathcal{A}$, two words u, v are* ***equivalent*** *if*

$$\forall w \in \Sigma^* : u \cdot w \in L \Leftrightarrow v \cdot w \in L$$

and

$$\forall w \in \Sigma^* : u \cdot w, v \cdot w \in Pref(\mathcal{L}(\mathcal{A})) \Rightarrow cv^{\mathcal{A}}(u \cdot w) = cv^{\mathcal{A}}(v \cdot w).$$

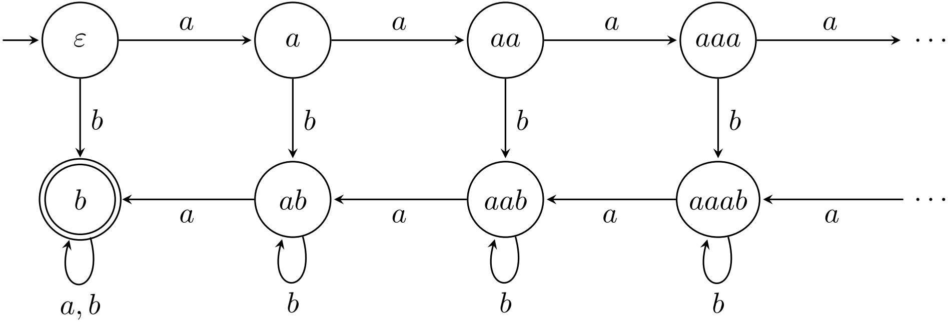


Figure 9: The **behavior graph** of the ROCA.

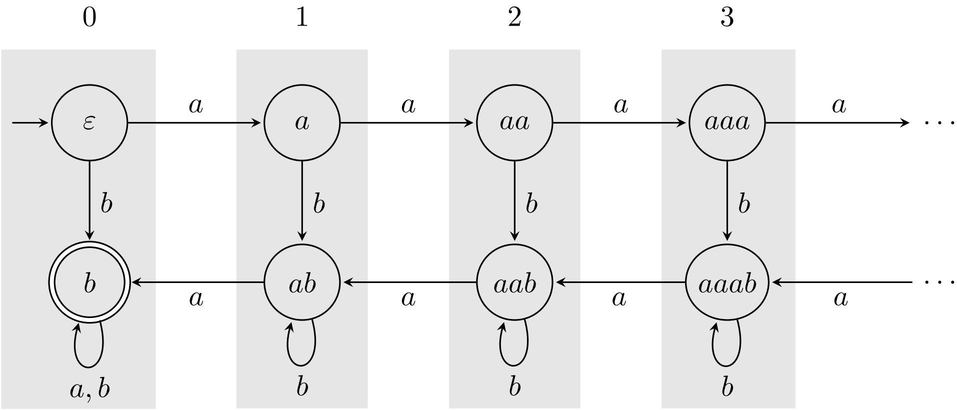


Figure 9: The **behavior graph** of the ROCA.

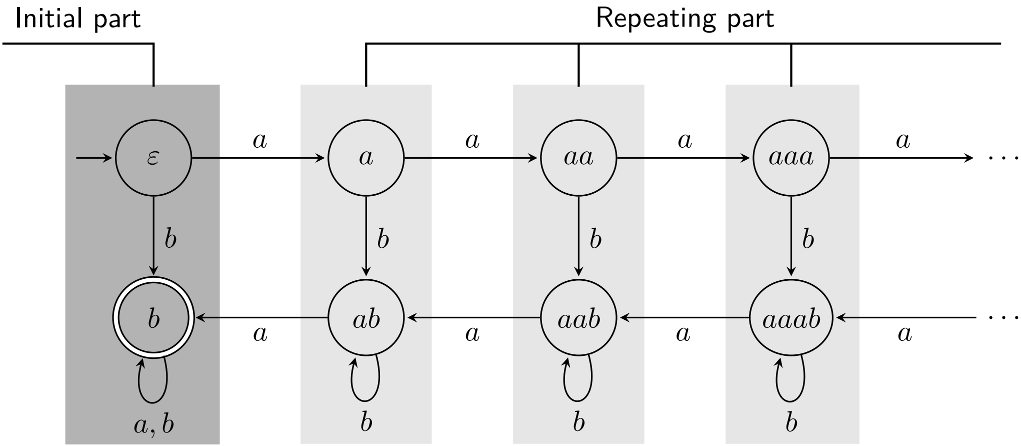


Figure 9: The **behavior graph** of the ROCA.

Theorem 3 (Based on Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010). *For any ROCA $\mathcal{A}$, there always exists an* ***ultimately periodic*** *representation of its behavior graph.*

Theorem 3 (Based on Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010). *For any ROCA $\mathcal{A}$, there always exists an* ***ultimately periodic*** *representation of its behavior graph.*

Proposition 4. *From an* ***ultimately periodic*** *representation of the behavior graph of $\mathcal{A}$, an ROCA $\mathcal{B}$ can be constructed such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

Theorem 3 (Based on Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010). *For any ROCA $\mathcal{A}$, there always exists an* ***ultimately periodic*** *representation of its behavior graph.*

Proposition 4. *From an* ***ultimately periodic*** *representation of the behavior graph of $\mathcal{A}$, an ROCA $\mathcal{B}$ can be constructed such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

Goal of L^*_{ROCA}: learn an **ultimately periodic** representation of the behavior graph.

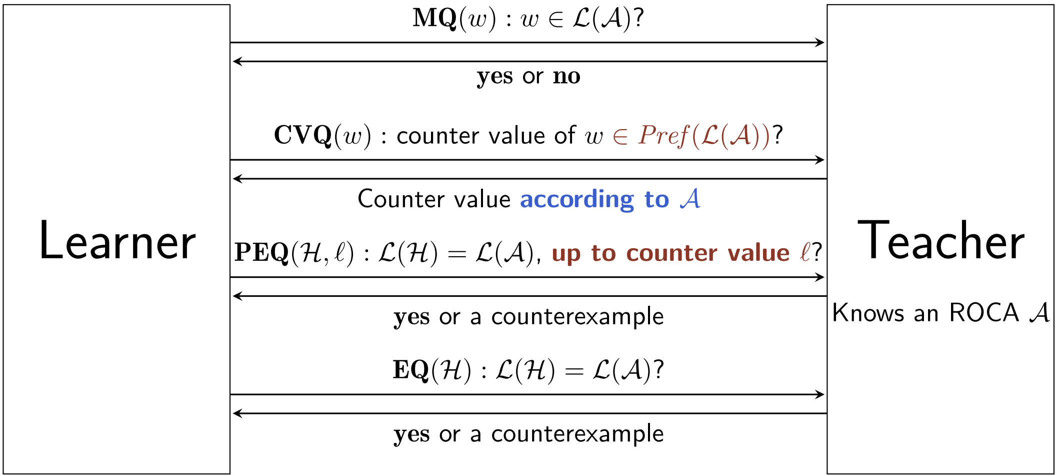


Figure 10: Adaptation of Angluin's framework for ROCAs.

Theorem 5. *Let $\mathcal{A}$ be the ROCA of the teacher and ℓ be the length of the* ***longest*** *counterexample returned by the teacher on (partial) equivalence queries. Then,*

- *the L^*_{ROCA} algorithm eventually terminates,*
- *in time and space* ***exponential*** *in $|\mathcal{A}|, |\Sigma|$ and ℓ, and*
- *asking*
  - *a number of* $\mathbf{PEQ}$ *in* $\mathcal{O}\left(\ell^3\right)$,
  - *a number of* $\mathbf{EQ}$ *in* $\mathcal{O}\left(|\mathcal{A}| \cdot \ell^2\right)$,
  - *and a number of* $\mathbf{MQ}$ *and* $\mathbf{CVQ}$ ***exponential*** *in $|\mathcal{A}|, |\Sigma|$ and ℓ.*

Theorem 5. *Let $\mathcal{A}$ be the ROCA of the teacher and ℓ be the length of the* ***longest*** *counterexample returned by the teacher on (partial) equivalence queries. Then,*

- *the L^*_{ROCA} algorithm eventually terminates,*
- *in time and space* ***exponential*** *in $|\mathcal{A}|, |\Sigma|$ and ℓ, and*
- *asking*
  - *a number of* $\mathbf{PEQ}$ *in* $\mathcal{O}\left(\ell^3\right)$,
  - *a number of* $\mathbf{EQ}$ *in* $\mathcal{O}\left(|\mathcal{A}| \cdot \ell^2\right)$,
  - *and a number of* $\mathbf{MQ}$ *and* $\mathbf{CVQ}$ ***exponential*** *in $|\mathcal{A}|, |\Sigma|$ and ℓ.*

Question. Why the exponential blowup?

Assume that an observation table for L^*_{ROCA} is:

- an observation table as for L^*,
- augmented with counter values for words known to be in $\mathit{Pref}(\mathcal{L}(\mathcal{A}))$:

$$C : (R \cup R\Sigma) \cdot S \to \mathbb{N} \cup \{\bot\}.$$

	ε
ε	$\mathbf{no}, 0$
a	$\mathbf{no}, 1$
ab	$\mathbf{no}, 1$
aba	$\mathbf{yes}, 0$
b	$\mathbf{yes}, 0$
aa	$\mathbf{no}, \bot$
abb	$\mathbf{no}, \bot$
$abaa$	$\mathbf{yes}, 0$
$abab$	$\mathbf{yes}, 0$

Assume that an observation table for L^*_{ROCA} is:

- an observation table as for L^*,
- augmented with counter values for words known to be in $\mathit{Pref}(\mathcal{L}(\mathcal{A}))$:

$$C : (R \cup R\Sigma) \cdot S \to \mathbb{N} \cup \{\bot\}.$$

	ε
ε	**no**, 0
a	**no**, 1
ab	**no**, 1
aba	**yes**, 0
b	**yes**, 0
aa	**no**, $\bot$
abb	**no**, $\bot$
$abaa$	**yes**, 0
$abab$	**yes**, 0

Moreover, assume

$$\forall u, v \in R \cup R\Sigma : u \equiv_{\mathscr{O}} v \Leftrightarrow \forall s \in S : T(u \cdot s) = T(v \cdot s) \wedge C(u \cdot s) = C(v \cdot s).$$

1. $\forall u \in R : abb \not\equiv_{\mathcal{O}} u$.
   $\rightsquigarrow$ Add abb to R.

	ε
ε	$\mathbf{no}, 0$
a	$\mathbf{no}, 1$
ab	$\mathbf{no}, 1$
aba	$\mathbf{yes}, 0$
b	$\mathbf{yes}, 0$
aa	$\mathbf{no}, \perp$
abb	$\mathbf{no}, \perp$
$abaa$	$\mathbf{yes}, 0$
$abab$	$\mathbf{yes}, 0$

1. $\forall u \in R : abb \not\equiv_{\mathcal{O}} u$.
   $\rightsquigarrow$ Add abb to R.

	ε
ε	$\mathbf{no}, 0$
a	$\mathbf{no}, 1$
ab	$\mathbf{no}, 1$
aba	$\mathbf{yes}, 0$
abb	$\mathbf{no}, 1$
b	$\mathbf{yes}, 0$
aa	$\mathbf{no}, \perp$
$abaa$	$\mathbf{yes}, 0$
$abab$	$\mathbf{yes}, 0$
$abba$	$\mathbf{yes}, 0$
$abbb$	$\mathbf{no}, \perp$

1. $\forall u \in R : abb \not\equiv_{\mathcal{O}} u$.
   $\rightsquigarrow$ Add abb to R.
2. $\forall u \in R : abbb \not\equiv_{\mathcal{O}} u$.
   $\rightsquigarrow$ Add $abbb$ to R.

	ε
ε	**no**, 0
a	**no**, 1
ab	**no**, 1
aba	**yes**, 0
abb	**no**, 1
b	**yes**, 0
aa	**no**, $\perp$
$abaa$	**yes**, 0
$abab$	**yes**, 0
$abba$	**yes**, 0
$abbb$	**no**, $\perp$

1. $\forall u \in R : abb \not\equiv_{\mathcal{O}} u$.
   $\rightsquigarrow$ Add abb to R.
2. $\forall u \in R : abbb \not\equiv_{\mathcal{O}} u$.
   $\rightsquigarrow$ Add $abbb$ to R.

	ε
ε	$\mathbf{no}, 0$
a	$\mathbf{no}, 1$
ab	$\mathbf{no}, 1$
aba	$\mathbf{yes}, 0$
abb	$\mathbf{no}, 1$
$abbb$	$\mathbf{no}, 1$
b	$\mathbf{yes}, 0$
aa	$\mathbf{no}, \perp$
$abaa$	$\mathbf{yes}, 0$
$abab$	$\mathbf{yes}, 0$
$abba$	$\mathbf{yes}, 0$
$abbba$	$\mathbf{yes}, 0$
$abbbb$	$\mathbf{no}, \perp$

1. $\forall u \in R : abb \not\equiv_{\mathscr{O}} u$.
   $\rightsquigarrow$ Add abb to R.
2. $\forall u \in R : abbb \not\equiv_{\mathscr{O}} u$.
   $\rightsquigarrow$ Add $abbb$ to R.
3. $\forall u \in R : abbbb \not\equiv_{\mathscr{O}} u$.
   $\rightsquigarrow$ Add $abbbb$ to R.
4. Repeat ad infinitum.

	ε
ε	**no**, 0
a	**no**, 1
ab	**no**, 1
aba	**yes**, 0
abb	**no**, 1
$abbb$	**no**, 1
b	**yes**, 0
aa	**no**, $\perp$
$abaa$	**yes**, 0
$abab$	**yes**, 0
$abba$	**yes**, 0
$abbba$	**yes**, 0
$abbbb$	**no**, $\perp$

We thus need **two** types of separators:

- for **membership**: $\widehat{S}$,
- for **counter values**: $S \subseteq \widehat{S}$.

We thus need **two** types of separators:

- for **membership**: $\widehat{S}$,
- for **counter values**: $S \subseteq \widehat{S}$.

	ε	a
ε	$\textbf{no}, 0$	$\textbf{no}$
a	$\textbf{no}, 1$	$\textbf{no}$
ab	$\textbf{no}, 1$	$\textbf{yes}$
aba	$\textbf{yes}, 0$	$\textbf{yes}$
aa	$\textbf{no}, \bot$	$\textbf{no}$
b	$\textbf{yes}, 0$	$\textbf{yes}$
abb	$\textbf{no}, 1$	$\textbf{yes}$
$abaa$	$\textbf{yes}, 0$	$\textbf{yes}$
$abab$	$\textbf{yes}, 0$	$\textbf{yes}$
aaa	$\textbf{no}, \bot$	$\textbf{no}$
aab	$\textbf{no}, \bot$	$\textbf{no}$

We thus need **two** types of separators:

- for **membership**: $\widehat{S}$,
- for **counter values**: $S \subseteq \widehat{S}$.

We **approximate** the equivalence relation:
$\forall u, v \in R \cup R\Sigma$, $u \in Approx(v)$ if and only if

- for all $s \in S$, $T(us) = T(vs)$, and
- for all $s \in S$, if $C(us) \neq \bot$ and $C(vs) \neq \bot$, then $C(us) = C(vs)$.

	ε	a
ε	**no**, 0	**no**
a	**no**, 1	**no**
ab	**no**, 1	**yes**
aba	**yes**, 0	**yes**
aa	**no**, $\bot$	**no**
b	**yes**, 0	**yes**
abb	**no**, 1	**yes**
$abaa$	**yes**, 0	**yes**
$abab$	**yes**, 0	**yes**
aaa	**no**, $\bot$	**no**
aab	**no**, $\bot$	**no**

We thus need **two** types of separators:

- for **membership**: $\widehat{S}$,
- for **counter values**: $S \subseteq \widehat{S}$.

We **approximate** the equivalence relation:
$\forall u, v \in R \cup R\Sigma$, $u \in Approx(v)$ if and only if

- for all $s \in S$, $T(us) = T(vs)$, and
- for all $s \in S$, if $C(us) \neq \bot$ and $C(vs) \neq \bot$, then $C(us) = C(vs)$.

Not necessarily transitive: $\varepsilon \in Approx(aa)$ and $aa \in Approx(a)$ but $\varepsilon \notin Approx(a)$.
Ensuring transitivity requires an **exponential** number of steps.

	ε	a
ε	**no**, 0	**no**
a	**no**, 1	**no**
ab	**no**, 1	**yes**
aba	**yes**, 0	**yes**
aa	**no**, $\bot$	**no**
b	**yes**, 0	**yes**
abb	**no**, 1	**yes**
$abaa$	**yes**, 0	**yes**
$abab$	**yes**, 0	**yes**
aaa	**no**, $\bot$	**no**
aab	**no**, $\bot$	**no**

Part III – Validating JSON Documents

Bruyère, Pérez, and Staquet, “Validating Streaming JSON Documents with Learned VPAs”, TACAS, 2023

```
{
  "title": "Active Learning of Automata with Resources",
  "details": {
    "pages": 341,
    "chapters": 11
  },
  "nesting": { "inside": { ... } }
}
```

```
{
  "title": "Active Learning of Automata with Resources",
  "details": {
    "pages": 341,
    "chapters": 11
  },
  "nesting": { "inside": { ... } }
}
```

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

There are also arrays (**ordered** collections of values); we mostly ignore them here.

A **JSON document** is composed of **nested** objects and arrays.

```
{
  "title": "Active Learning of Automata with Resources",
  "details": {
    "pages": 341,
    "chapters": 11
  },
  "nesting": { "inside": { ... } }
}
```

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

There are also arrays (**ordered** collections of values); we mostly ignore them here.

A **JSON document** is composed of **nested** objects and arrays.

We want to verify that the document satisfies some **constraints**:

- `"title"` $\mapsto$ string
- `"details"` $\mapsto$ object such that
  - `"pages"` $\mapsto$ integer
  - `"chapters"` $\mapsto$ integer
- And so on.

Classical validation algorithm:

1. Explore the JSON document and the constraints in parallel;
2. If the current value does not match the sub-constraints, stop;
3. Otherwise, repeat recursively.

Classical validation algorithm:

1. Explore the JSON document and the constraints in parallel;
2. If the current value does not match the sub-constraints, stop;
3. Otherwise, repeat recursively.

The constraints can contain Boolean operations.
$\hookrightarrow$ The **same** value must be processed **multiple** times.

Assume we are in a **streaming context**.
$\hookrightarrow$ We receive the document one fragment at a time.

Assume we are in a **streaming context**.
$\hookrightarrow$ We receive the document one fragment at a time.

The classical algorithm must wait for the **whole** document before starting.

Our approach is based on **learning** an automaton from the constraints and then use it for **validation**.

Question. Which kind of automaton?

Question. How to use it to validate documents while receiving them?

We **abstract** the values:

```
{
  "title": "Active Learning of Automata with Resources",
  "details": {
    "pages": 341,
    "chapters": 11
  }
}
```

We **abstract** the values:

```
{
  "title": s,
  "details": {
    "pages": i,
    "chapters": i
  }
}
```

We **abstract** the values:

```
{
  "title": s,
  "details": {
    "pages": i,
    "chapters": i
  }
}
```

Question. How to remember the **nesting** of objects and arrays?

$\hookrightarrow$ A **stack**.

We **abstract** the values:

```
{
  "title": s,
  "details": {
    "pages": i,
    "chapters": i
  }
}
```

Question. How to remember the **nesting** of objects and arrays?

$\hookrightarrow$ A **stack**.

Question. Which kind of automaton?

$\hookrightarrow$ A **(visibly) pushdown automaton** (**VPA**).

Theorem 6. *Let $\mathcal{C}$ be a set of constraints describing JSON documents. There* ***always*** *exists a VPA $\mathcal{A}$ whose language is the set of documents that are* ***valid*** *for $\mathcal{C}$.*

Theorem 6. *Let $\mathcal{C}$ be a set of constraints describing JSON documents. There* ***always*** *exists a VPA $\mathcal{A}$ whose language is the set of documents that are* ***valid*** *for $\mathcal{C}$.*

Theorem 7 (Isberner, "Foundations of active automata learning: an algorithmic perspective", 2015). *Let L be a language accepted by some VPA. The TTT_{VPA} can learn a VPA accepting L with a* ***polynomial*** *number of membership and equivalence queries.*

Theorem 6. *Let $\mathcal{C}$ be a set of constraints describing JSON documents. There* ***always*** *exists a VPA $\mathcal{A}$ whose language is the set of documents that are* ***valid*** *for $\mathcal{C}$.*

Theorem 7 (Isberner, "Foundations of active automata learning: an algorithmic perspective", 2015). *Let L be a language accepted by some VPA. The TTT_{VPA} can learn a VPA accepting L with a* ***polynomial*** *number of membership and equivalence queries.*

Question. How to deal with the **exponential** number of permutations of the **(unordered)** keys?

Fix an **order** over the set of keys.

Theorem 8. *Let $\mathcal{C}$ be a set of constraints over keys Σ_{key} and $\mathcal{A}$ be a **VPA** that recognizes $\mathcal{C}$, **with a fixed order** over Σ_{key}.*
Then, checking whether a JSON document J satisfies $\mathcal{C}$

- *is in time **polynomial** in $|J|$ and $|\mathcal{A}|$ and **exponential** in $|\Sigma_{\text{key}}|$,*
- *and uses an amount of memory **polynomial** in $|\mathcal{A}|$, $|\Sigma_{\text{key}}|$, and $d(J)$.*

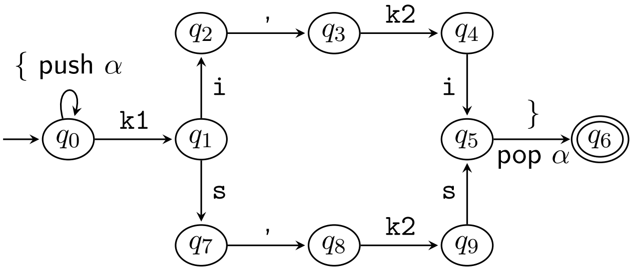
For a JSON document J, $d(J)$ denotes its **depth**: number of nested objects and arrays.

Theorem 8. *Let $\mathcal{C}$ be a set of constraints over keys Σ_{key} and $\mathcal{A}$ be a* ***VPA*** *that recognizes $\mathcal{C}$,* ***with a fixed order*** *over Σ_{key}.*
Then, checking whether a JSON document J satisfies $\mathcal{C}$

- *is in time* ***polynomial*** *in $|J|$ and $|\mathcal{A}|$ and* ***exponential*** *in $|\Sigma_{\text{key}}|$,*
- *and uses an amount of memory* ***polynomial*** *in $|\mathcal{A}|$, $|\Sigma_{\text{key}}|$, and $d(J)$.*

For a JSON document J, $d(J)$ denotes its **depth**: number of nested objects and arrays.

Question. How to use the VPA to validate JSON documents whose objects do **not** follow the **fixed order**?



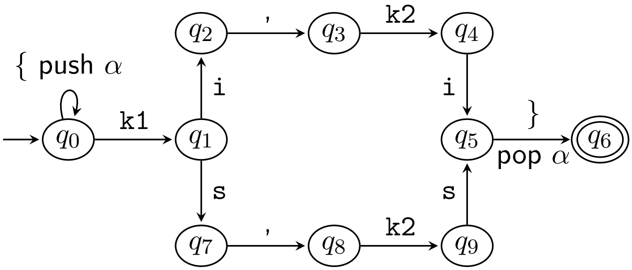
Valid documents:

{ k1 i , k2 i }

{ k2 i , k1 i }

{ k1 s , k2 s }

{ k2 s , k1 s }



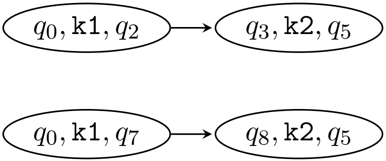
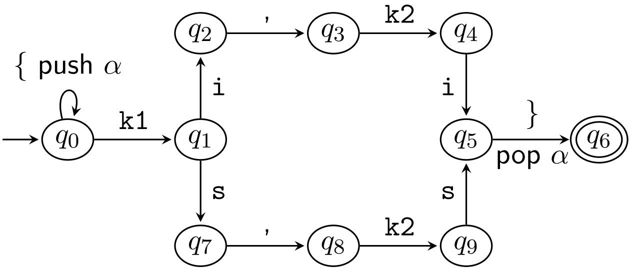


Figure 11: The **key graph**.



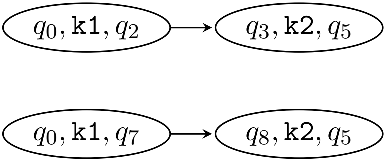
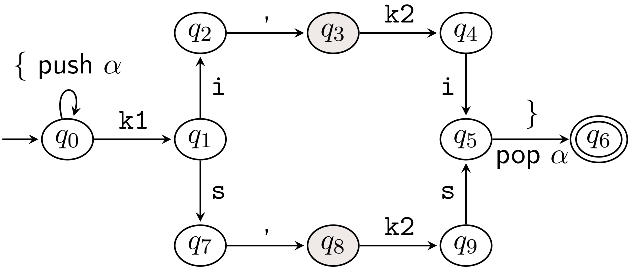


Figure 11: The **key graph**.

We read { `k2` `i` , `k1` `i` }.



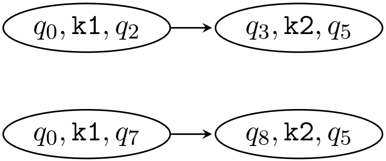
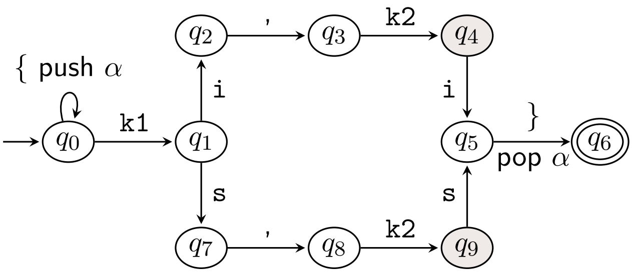


Figure 11: The **key graph**.

We read { `k2 i , k1 i` }.

Potential states for `k2 i`: $\{q_3, q_8\}$.



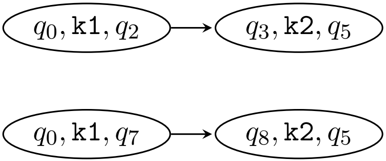
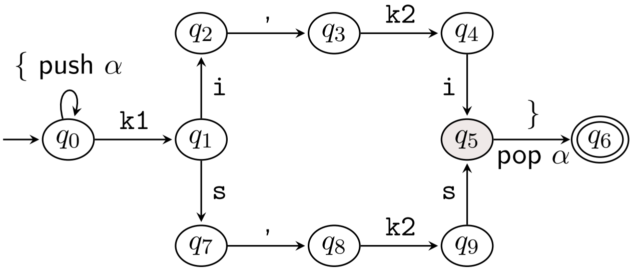


Figure 11: The **key graph**.

We read { `k2 i , k1 i` }.

Potential states for `k2 i`: $\{q_3, q_8\}$.
After reading `k2`: $\{q_4, q_9\}$.



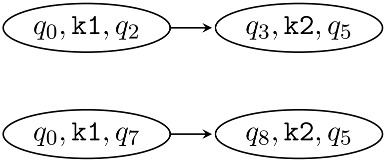
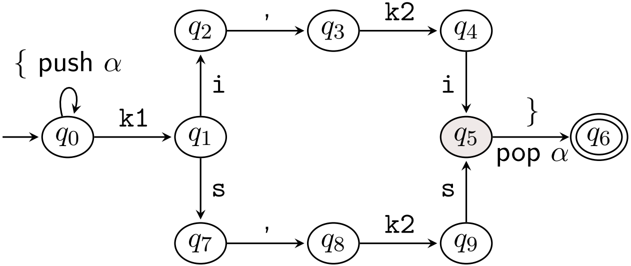


Figure 11: The **key graph**.

We read $\{$ `k2 i , k1 i` $\}$.

Potential states for `k2 i`: $\{q_3, q_8\}$.
After reading `k2`: $\{q_4, q_9\}$.
After reading `k2 i`: $\{q_5\}$.



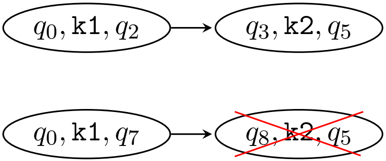
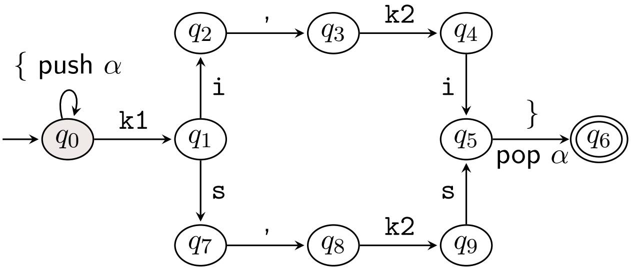


Figure 11: The **key graph**.

We read { `k2` `i` , `k1` `i` }.

Potential states for `k2 i`: $\{q_3, q_8\}$.
After reading `k2`: $\{q_4, q_9\}$.
After reading `k2 i`: $\{q_5\}$.
Not $q_8 \xrightarrow{\texttt{k2 i}} q_5$.



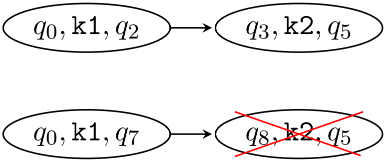
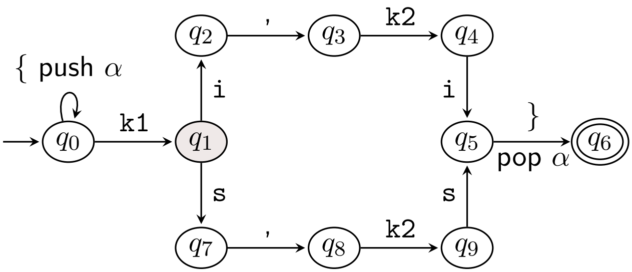


Figure 11: The **key graph**.

We read { k2 i , k1 i }.

Potential states for k2 i: $\{q_3, q_8\}$.
After reading k2: $\{q_4, q_9\}$.
After reading k2 i: $\{q_5\}$.
Not $q_8 \xrightarrow{\texttt{k2 i}} q_5$.

Potential states for k1 i: $\{q_0\}$.



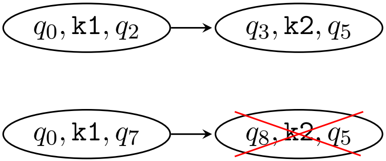
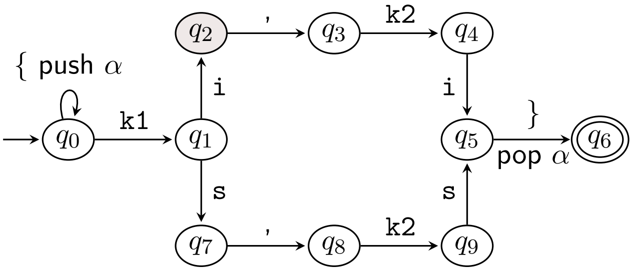


Figure 11: The **key graph**.

We read $\{$ `k2` `i` `,` `k1` `i` $\}$.

Potential states for `k2 i`: $\{q_3, q_8\}$.
After reading `k2`: $\{q_4, q_9\}$.
After reading `k2 i`: $\{q_5\}$.
Not $q_8 \xrightarrow{\texttt{k2 i}} q_5$.

Potential states for `k1 i`: $\{q_0\}$.
After reading `k1`: $\{q_1\}$.



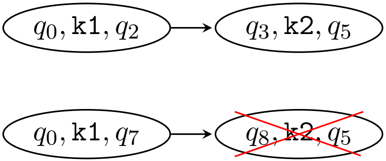
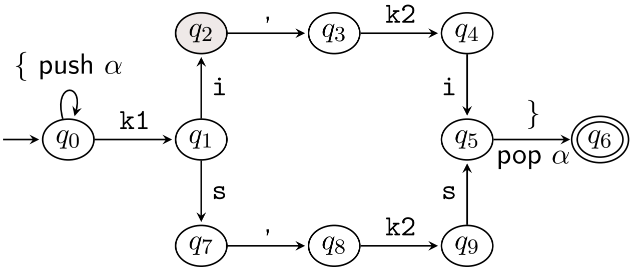


Figure 11: The **key graph**.

We read { k2 i , k1 i }.

Potential states for `k2 i`: $\{q_3, q_8\}$.
After reading `k2`: $\{q_4, q_9\}$.
After reading `k2 i`: $\{q_5\}$.
Not $q_8 \xrightarrow{\texttt{k2 i}} q_5$.

Potential states for `k1 i`: $\{q_0\}$.
After reading `k1`: $\{q_1\}$.
After reading `k1 i`: $\{q_2\}$.



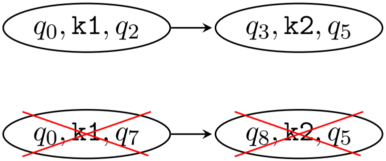
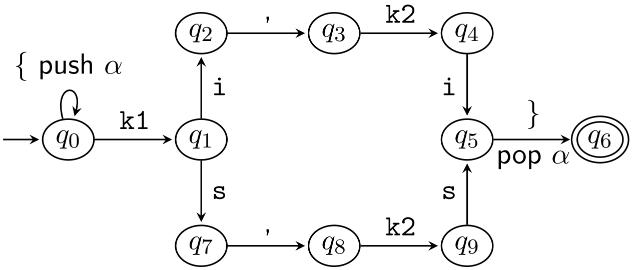


Figure 11: The **key graph**.

We read $\{$ `k2 i , k1 i` $\}$.

Potential states for `k2 i`: $\{q_3, q_8\}$.
After reading `k2`: $\{q_4, q_9\}$.
After reading `k2 i`: $\{q_5\}$.
Not $q_8 \xrightarrow{\texttt{k2 i}} q_5$.

Potential states for `k1 i`: $\{q_0\}$.
After reading `k1`: $\{q_1\}$.
After reading `k1 i`: $\{q_2\}$.
Not $q_0 \xrightarrow{\texttt{k1 i}} q_7$.



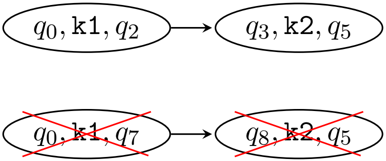


Figure 11: The **key graph**.

We read $\{$ $\mathtt{k2}$ $\mathtt{i}$, $\mathtt{k1}$ $\mathtt{i}$ $\}$. $\rightsquigarrow$ **Valid document**.

Potential states for $\mathtt{k2\ i}$: $\{q_3, q_8\}$.
After reading $\mathtt{k2}$: $\{q_4, q_9\}$.
After reading $\mathtt{k2\ i}$: $\{q_5\}$.
Not $q_8 \xrightarrow{\mathtt{k2\ i}} q_5$.

Potential states for $\mathtt{k1\ i}$: $\{q_0\}$.
After reading $\mathtt{k1}$: $\{q_1\}$.
After reading $\mathtt{k1\ i}$: $\{q_2\}$.
Not $q_0 \xrightarrow{\mathtt{k1\ i}} q_7$.

Part IV – Mealy Machines with Timers

Bruyère, Pérez, Staquet, and Vaandrager, “Automata with Timers”, FORMATS, 2023
Bruyère, Garhewal, et al., “Active Learning of Mealy Machines with Timers”, 2024

A **Mealy machine with timers** (**MMT**, for short) is a tuple $\mathcal{M} = (I, O, X, Q, q_0, \chi, \delta)$ where

- X is the set of **timers**;
- I is the set of **inputs**; the set of all **actions** is:

$$I \cup \{to[x] \mid x \in X\};$$

- O is the set of **outputs**;

A **Mealy machine with timers** (**MMT**, for short) is a tuple $\mathcal{M} = (I, O, X, Q, q_0, \chi, \delta)$ where

- X is the set of **timers**;
- I is the set of **inputs**; the set of all **actions** is:

$$I \cup \{to[x] \mid x \in X\};$$

- O is the set of **outputs**;
- Q is the finite set of **states**;
- $q_0 \in Q$ is the **initial state**;
- $\chi : Q \to 2^X$ gives the **active timers** of each state;

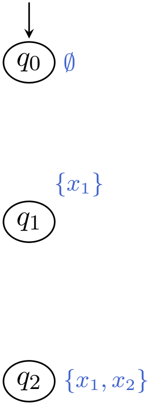


Figure 12: An MMT.

A **Mealy machine with timers** (**MMT**, for short) is a tuple $\mathcal{M} = (I, O, X, Q, q_0, \chi, \delta)$ where

- X is the set of **timers**;
- I is the set of **inputs**; the set of all **actions** is:

$$I \cup \{to[x] \mid x \in X\};$$

- O is the set of **outputs**;
- Q is the finite set of **states**;
- $q_0 \in Q$ is the **initial state**;
- $\chi : Q \to 2^X$ gives the **active timers** of each state;
- δ is the **transition function**.

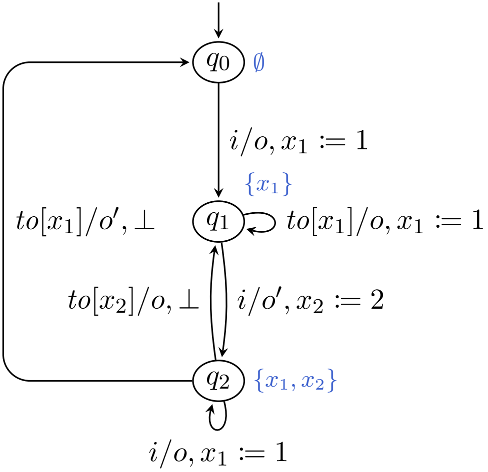
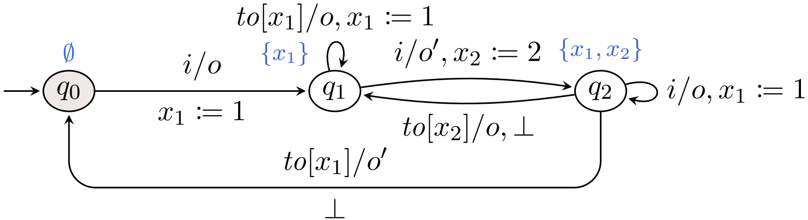
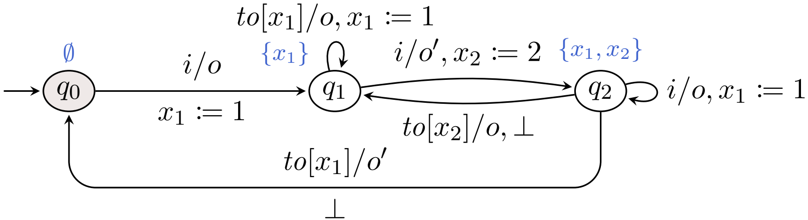


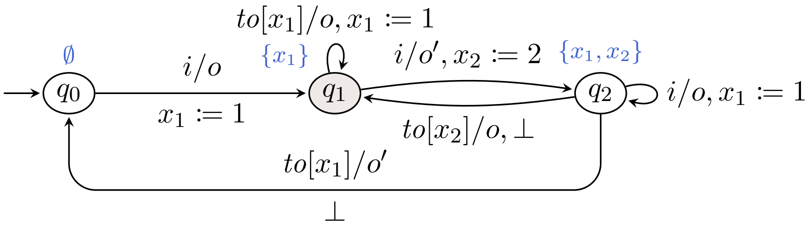
Figure 12: An MMT.


$to[x_1]/o, x_1 := 1$
$\emptyset$
$\{x_1\}$
$i/o', x_2 := 2$
$\{x_1, x_2\}$
i/o
q_0
q_1
q_2
$i/o, x_1 := 1$
$x_1 := 1$
$to[x_2]/o, \perp$
$to[x_1]/o'$
$\perp$

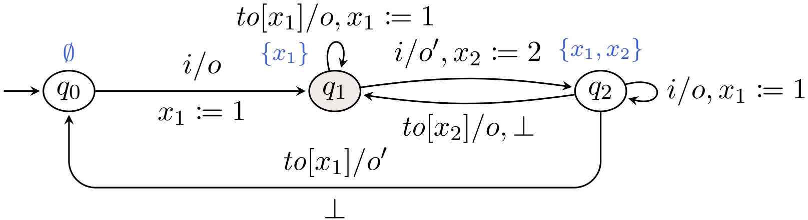
$(q_0, \emptyset)$



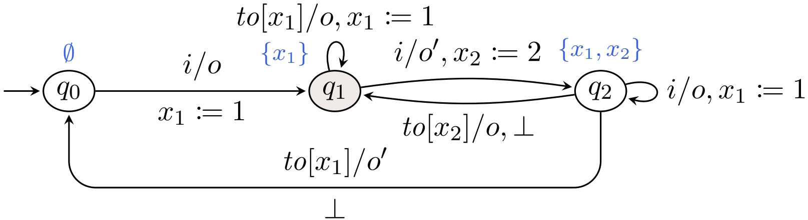
$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset)$$



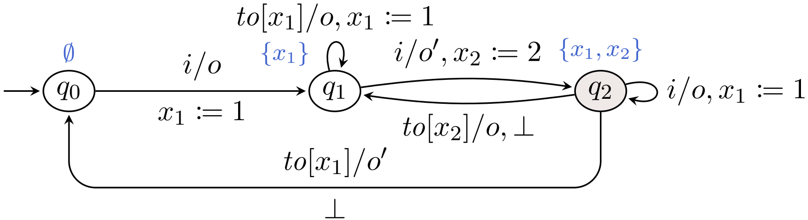
$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1)$$



$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0)$$

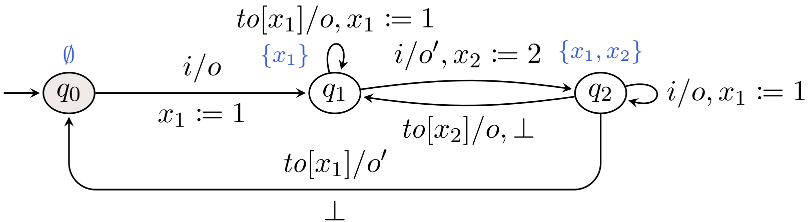


$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0)$$
$$\xrightarrow{to[x_1]/o} (q_1, x_1 = 1)$$



$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0)$$

$$\xrightarrow{to[x_1]/o} (q_1, x_1 = 1) \xrightarrow{0} (q_1, x_1 = 1) \xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2)$$



$$
\begin{aligned}
(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1) \xrightarrow{1} (q_1, x_1 = 0) \\
&\xrightarrow{to[x_1]/o} (q_1, x_1 = 1) \xrightarrow{0} (q_1, x_1 = 1) \xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2) \\
&\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1) \xrightarrow{to[x_1]/o'} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).
\end{aligned}
$$

We adapt $L^{\#}$ (active learning algorithm for Mealy machines[2]) to MMTs: $L^{\#}_{\mathsf{MMT}}$.

[2] Vaandrager et al., "A New Approach for Active Automata Learning Based on Apartness", 2022.

We adapt $L^{\#}$ (active learning algorithm for Mealy machines[2]) to MMTs: $L^{\#}_{\mathsf{MMT}}$.

Theorem 9. *Let $\mathcal{M}$ be a **good** MMT and ℓ be the length of the longest counterexample returned by the teacher. Then,*

- *the $L^{\#}_{\mathsf{MMT}}$ algorithm eventually terminates*
- *in time and number of queries **polynomial** in $|\mathcal{M}|, |I|$, and ℓ, and **exponential** in $|X|$.*

[2] Vaandrager et al., "A New Approach for Active Automata Learning Based on Apartness", 2022.

We adapt $L^{\#}$ (active learning algorithm for Mealy machines[2]) to MMTs: $L^{\#}_{\text{MMT}}$.

Theorem 9. *Let $\mathcal{M}$ be a **good** MMT and ℓ be the length of the longest counterexample returned by the teacher. Then,*

- *the $L^{\#}_{\text{MMT}}$ algorithm eventually terminates*
- *in time and number of queries **polynomial** in $|\mathcal{M}|, |I|$, and ℓ, and **exponential** in $|X|$.*

Question. When is an MMT **good**?

[2] Vaandrager et al., "A New Approach for Active Automata Learning Based on Apartness", 2022.

Question. When is an MMT **good**?

- Timeouts are **observed** via their **outputs**.

Question. When is an MMT **good**?

- Timeouts are **observed** via their **outputs**.
- For **every** untimed sequence of transitions, there exists a timed run using **exactly** this sequence of transitions...

Question. When is an MMT **good**?

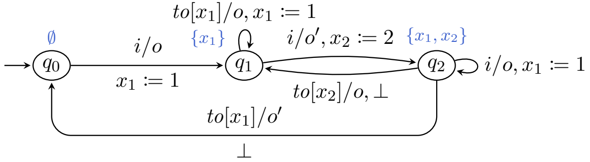
- Timeouts are **observed** via their **outputs**.
- For **every** untimed sequence of transitions, there exists a timed run using **exactly** this sequence of transitions...
- with **all delays** > 0 and there is **at most** one timer that times out at any time.
  $\hookrightarrow$ **Deterministic** behavior.

Question. When is an MMT **good**?

- Timeouts are **observed** via their **outputs**.
- For **every** untimed sequence of transitions, there exists a timed run using **exactly** this sequence of transitions...
- with **all delays** > 0 and there is **at most** one timer that times out at any time.
  $\hookrightarrow$ **Deterministic** behavior.

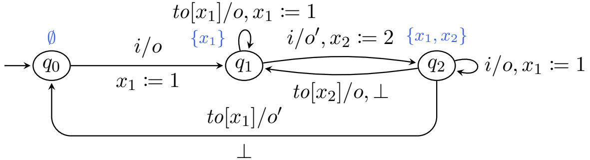
The last condition does **not** always hold.

Question. When can we guarantee a deterministic behavior?



$$\begin{aligned}(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1)\\ &\xrightarrow{1} (q_1, x_1 = 0)\\ &\xrightarrow{to[x_1]/o} (q_1, x_1 = 1)\\ &\xrightarrow{0} (q_1, x_1 = 1)\\ &\xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2)\\ &\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1)\\ &\xrightarrow{to[x_1]/o'} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).\end{aligned}$$

$$\begin{aligned}(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1)\\ &\xrightarrow{1} (q_1, x_1 = 0)\\ &\xrightarrow{i/o'} (q_2, x_1 = 0, x_2 = 2)\\ &\xrightarrow{0} (q_2, x_1 = 0, x_2 = 2)\\ &\xrightarrow{to[x_1]/o'} (q_0, \emptyset)\\ &\xrightarrow{1.5} (q_0, \emptyset).\end{aligned}$$



$$
\begin{aligned}
(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x_1 = 1) \\
&\xrightarrow{1} (q_1, x_1 = 0) \\
&\xrightarrow{to[x_1]/o} (q_1, x_1 = 1) \\
&\xrightarrow{0} (q_1, x_1 = 1) \\
&\xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2) \\
&\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1) \\
&\xrightarrow{to[x_1]/o'} (q_0, \emptyset) \xrightarrow{0.5} (q_0, \emptyset).
\end{aligned}
$$

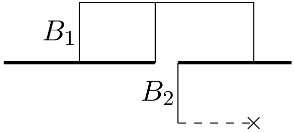


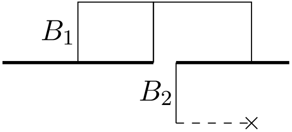
Figure 13: The **blocks** of the timed run on the left.

Definition 10. *We have a **race** when two actions happen **simultaneously**.*

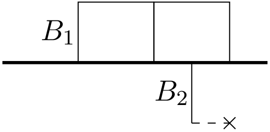
Question. Can we avoid a race, while seeing the **same untimed sequence of transitions**?

Definition 10. *We have a **race** when two actions happen **simultaneously**.*

Question. Can we avoid a race, while seeing the **same untimed sequence of transitions**?

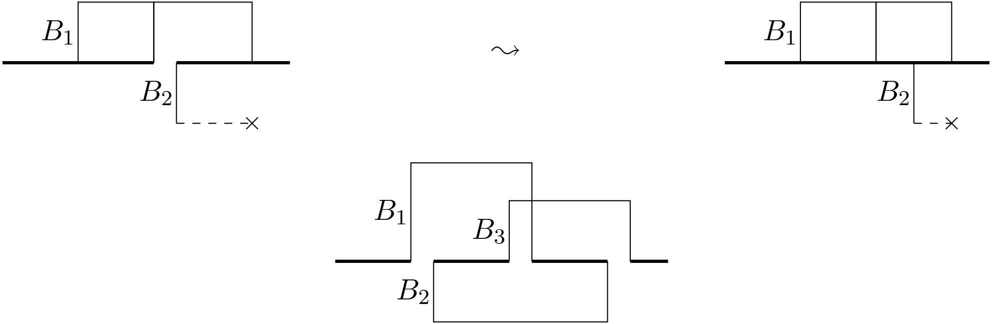


$\rightsquigarrow$



Definition 10. *We have a **race** when two actions happen **simultaneously**.*

Question. Can we avoid a race, while seeing the **same untimed sequence of transitions**?



Theorem 11. *Given an MMT $\mathcal{M}$,* ***deciding*** *whether every untimed sequence of $\mathcal{M}$ can be observed via a timed run in which there is* ***no race*** *is* PSPACE*-hard and in* 3EXP*.*

It is in PSPACE *if the inputs I and the timers X are fixed.*

Part V – Conclusion

Goals of the thesis:

- New learning algorithms for automata extended with
  - a **counter** (Part 2),
  - **timers** (Part 4).
- Validation algorithm relying on learning an automaton with a **stack** (Part 3).

Thank you!

References I

Angluin, Dana. “Learning Regular Sets from Queries and Counterexamples”. In: 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.

Bruyère, Véronique, Bharat Garhewal, et al. “Active Learning of Mealy Machines with Timers”. In: *CoRR* abs/2403.02019 (2024). DOI: 10.48550/ARXIV.2403.02019. arXiv: 2403.02019. URL: https://doi.org/10.48550/arXiv.2403.02019.

References II

Bruyère, Véronique, Guillermo A. Pérez, and Gaëtan Staquet. "Learning Realtime One-Counter Automata". In: *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*. Ed. by Dana Fisman and Grigore Rosu. Vol. 13243. Springer, 2022, pp. 244–262. DOI: 10.1007/978-3-030-99524-9_13. URL: https://doi.org/10.1007/978-3-030-99524-9%5C_13.

References III

Bruyère, Véronique, Guillermo A. Pérez, and Gaëtan Staquet. "Validating Streaming JSON Documents with Learned VPAs". In: *Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I*. Ed. by Sriram Sankaranarayanan and Natasha Sharygina. Vol. 13993. Springer, 2023, pp. 271–289. DOI: 10.1007/978-3-031-30823-9_14. URL: https://doi.org/10.1007/978-3-031-30823-9%5C_14.

References IV

Bruyère, Véronique, Guillermo A. Pérez, Gaëtan Staquet, and Frits W. Vaandrager. "Automata with Timers". In: *Formal Modeling and Analysis of Timed Systems - 21st International Conference, FORMATS 2023, Antwerp, Belgium, September 19-21, 2023, Proceedings*. Ed. by Laure Petrucci and Jeremy Sproston. Vol. 14138. Springer, 2023, pp. 33–49. DOI: 10.1007/978-3-031-42626-1_3. URL: https://doi.org/10.1007/978-3-031-42626-1%5C_3.

Isberner, Malte. "Foundations of active automata learning: an algorithmic perspective". PhD thesis. Technical University Dortmund, Germany, 2015. URL: https://hdl.handle.net/2003/34282.

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.

References V

Vaandrager, Frits W. et al. "A New Approach for Active Automata Learning Based on Apartness". In: *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*. Ed. by Dana Fisman and Grigore Rosu. Vol. 13243. Springer, 2022, pp. 223–243. DOI: 10.1007/978-3-030-99524-9_12. URL: https://doi.org/10.1007/978-3-030-99524-9%5C_12.

Part VI – Appendix

Appendix

Definition 12 (Myhill-Nerode congruence). *Two words u, v are **L-equivalent**, noted $u \sim_L v$, if*

$$\forall w \in \Sigma^* : u \cdot w \in L \Leftrightarrow v \cdot w \in L.$$

A **Mealy machine** (**MM**, for short) is a tuple $\mathcal{A} = (I, O, Q, q_0, \delta)$ where:

- I is the set of **inputs**,
- O is the set of **outputs**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $\delta : Q \times I \to Q \times O$ is the **transition function**.

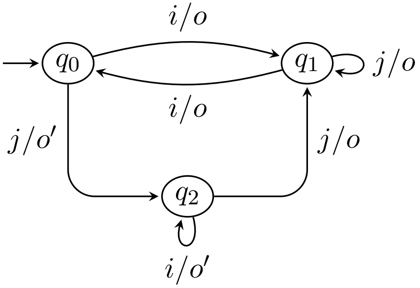


Figure 14: An MM.

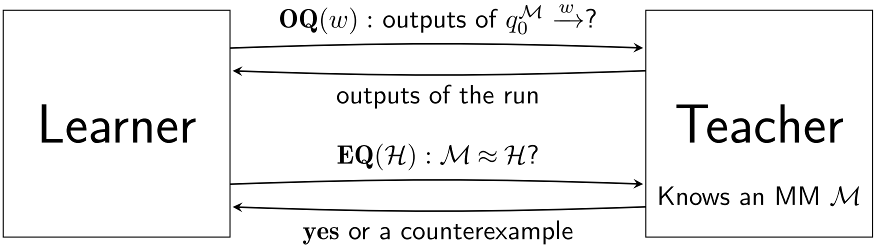


Figure 15: Adaptation of Angluin's framework for Mealy machines.

Theorem 13 (Vaandrager et al., "A New Approach for Active Automata Learning Based on Apartness", 2022). *Let n be the size of a **minimal** MM equivalent to the teacher's MM, and ℓ be the length of the **longest** counterexample provided by the teacher. Then,*

- *the $L^{\#}$ algorithm eventually terminates,*
- *in time and space **polynomial** in n and ℓ,*
- *with at most $n - 1$ equivalence queries and $\mathcal{O}\left(n^2 + n \log \ell\right)$ membership queries.*

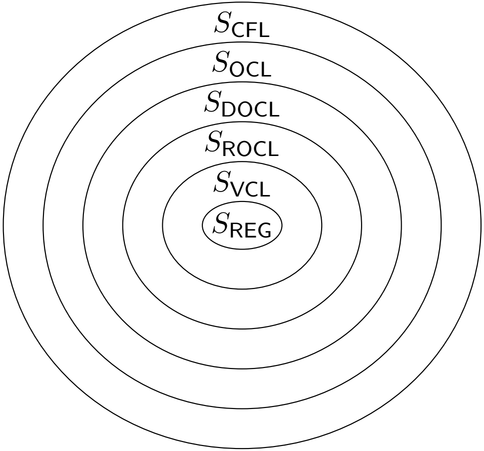


Figure 16: Hierarchy of one-counter languages.

A **pushdown alphabet**, noted $\widetilde{\Sigma} = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$, is the union of three **disjoint** alphabets:

- Σ_c: **calls**,
- Σ_r: **returns**,
- Σ_{int}: **internal symbols**.

The **sign** of a symbol $a \in \widetilde{\Sigma}$ is:

$$sign(a) = \begin{cases} 1 & \text{if } a \in \Sigma_c \\ -1 & \text{if } a \in \Sigma_r \\ 0 & \text{if } a \in \Sigma_{int}. \end{cases}$$

The **counter value** of a word $w = a_1 \cdots a_n$ is

$$cv(w) = \sum_{\ell=0}^{n} sign(a_\ell).$$

The **height** of w is

$$height(w) = \max_{u \in Pref(w)} cv(u).$$

A **visibly one-counter automaton** (**VCA**, for short) is a tuple $\mathcal{A} = (\widetilde{\Sigma}, Q, q_0, F, \delta)$ where:

- $\widetilde{\Sigma}$ is the **pushdown alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,
- $\delta : Q \times \widetilde{\Sigma} \times \{=0, >0\} \to Q$ is the **transition function**.

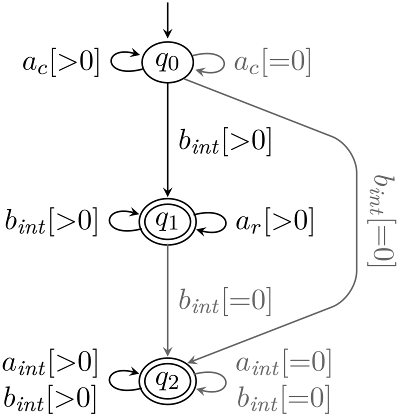


Figure 17: A VCA.

Definition 14 (Myhill-Nerode congruence). *Two words u, v are **L-equivalent**, noted $u \sim_L v$, if*

$$\forall w \in \Sigma^* : u \cdot w \in L \Leftrightarrow v \cdot w \in L.$$

Proposition 15 (**Not in** Neider and Löding, *Learning visibly one-counter automata in polynomial time*, 2010). *Let L be a language accepted by some VCA, and $u, v \in Pref(L)$ such that $u \sim_L v$. Then, $cv(u) = cv(v)$.*

The **behavior graph** of L is constructed from the equivalence classes of $\sim_L$, restricted to co-reachable states.

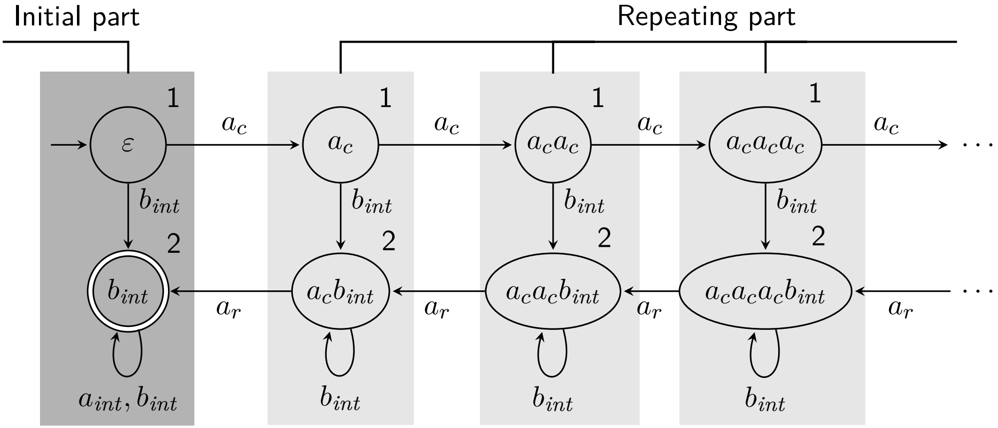


Figure 18: A **behavior graph** $BG(L)$.

Question. How to encode the behavior graph in a finite representation?

Definition 16. *The **level** ℓ of $BG(L)$ is:*

$$level(BG(L), \ell) = \{[\![w]\!]_{\sim_L} \mid cv(w) = \ell\}.$$

*The **width** of $BG(L)$ is:*

$$width(BG(L)) = \max_{\ell \in \mathbb{N}} level(BG(L), \ell).$$

Proposition 17. *The width of $BG(L)$ is always **bounded**.*

Question. How to encode the behavior graph of a VCL in a finite representation?

Enumerate the states in level ℓ:

$$\nu_\ell : level(BG(L), \ell) \to \{1, \ldots, K\},$$

with $K = width(BG(L))$.

Encode the transitions of $BG(L)$:

$$\tau_\ell : \{1, \ldots, K\} \times \widetilde{\Sigma} \to \{1, \ldots, K\}.$$

That encoding is unique, given the enumerations ν_ℓ.

Theorem 18. *For any VCL L, there exists an enumeration*

$$\nu_\ell : level(BG(L), \ell) \to \{1, \ldots, width(BG(L))\}$$

*such that $\tau_0\tau_1 \cdots$ is **ultimately periodic**, i.e., there are an **offset** $m > 0$ and a **period** $k \leq 0$ such that $\tau_0 \cdots \tau_{m-1}(\tau_m \cdots \tau_{m+k-1})^\omega$.*

Theorem 19. *For any ROCL L, there exists a VCL $\widetilde{L}$ such that $BG(L)$ and $BG(\widetilde{L})$ are isomorphic (up to a change of alphabet).*
The isomorphism respects the counter values and both offset and period of ultimately periodic descriptions.

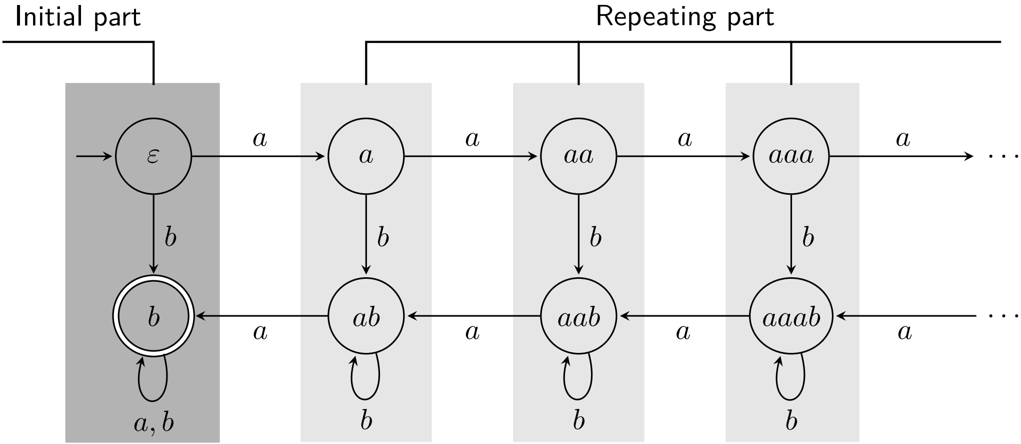


Figure 19: A behavior graph $BG(L)$.

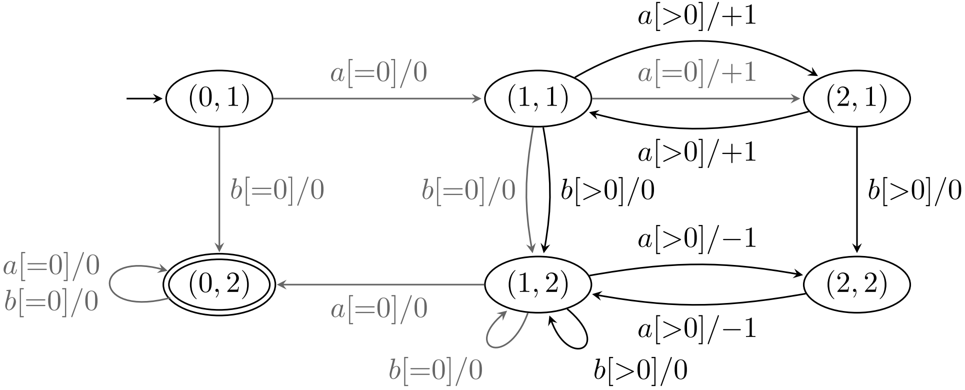


Figure 20: An ROCA constructed from $BG(L)$.

Question. When is it possible to construct an ROCA from the observation table?

The table must be **closed**:

$$\forall u \in R\Sigma : Approx(u) \cap R \neq \emptyset.$$

If not, add u to R.

The table must be **Σ-consistent**:

$$\forall u \in R, a \in \Sigma, v \in Approx(u) \cap R : \\ u \cdot a \in Approx(v \cdot a).$$

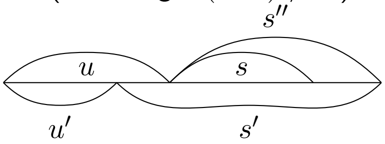
If not, add $a \cdot s$ to S, with $s \in S$ the culprit of $u \cdot a \notin Approx(v \cdot a)$.

Question. When is it possible to construct an ROCA from the observation table?

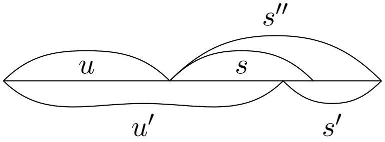
The table must be **$\perp$-consistent**:

$$\forall u \in R \cup R\Sigma, v \in Approx(u), s \in S : C(u \cdot s) = \perp \Leftrightarrow C(v \cdot s) = \perp.$$

If not (assuming $C(u \cdot s) \neq \perp$):

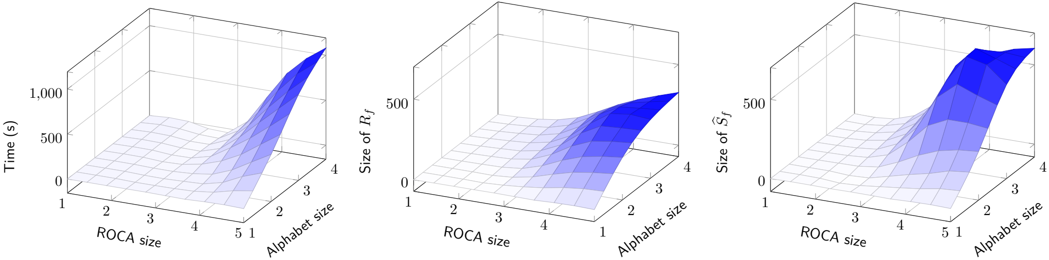


If u' is a prefix of u, add all suffixes of s'' to S.



If u is a proper prefix of u':

- $v \cdot s'' \in \mathcal{L}_{\leq \ell}(L) \rightsquigarrow$ add all suffixes of s'' to $\widehat{S}$;
- $v \cdot s'' \notin \mathcal{L}_{\leq \ell}(L) \rightsquigarrow$ add all suffixes of s'' to $\widehat{S}$ and S.



(a) Mean of the total time.

(b) Mean of the final size of representatives.

(c) Mean of the final size of separators.

Figure 21: Results for the benchmarks based on random ROCAs.

A **visibly pushdown automaton** (**VPA**, for short) is a tuple $\mathcal{A} = (\widetilde{\Sigma}, \Gamma, Q, q_0, F, \delta)$ where:

- $\widetilde{\Sigma}$ is the **pushdown alphabet**,
- Q is the **finite, non-empty** set of **states**,
- $q_0 \in Q$ is the **initial state**,
- $F \subseteq Q$ is the set of **final states**,
- $\delta = \delta_c \cup \delta_r \cup \delta_{int}$ is the **transition relation**:
  - $\delta_c \subseteq (Q \times \Sigma_c) \times (Q \times \Gamma)$,
  - $\delta_r \subseteq (Q \times \Sigma_r \times \Gamma) \times Q$,
  - $\delta_{int} \subseteq (Q \times \Sigma_{int}) \times Q$.

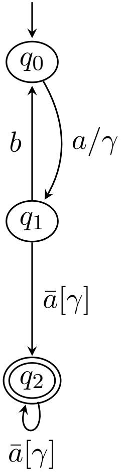


Figure 22: A VPA.

The **key graph** $\mathrm{G}_{\mathcal{A}}$ of a VPA $\mathcal{A}$ has:

- the vertices (p, k, p') with $p, p' \in Q^{\mathcal{A}}$ and $k \in \Sigma_{\text{key}}$ if there exists a stacked run

  $$(p, \varepsilon) \xrightarrow{k \cdot v} (p', \varepsilon) \in \mathit{sruns}(\mathcal{A})$$

  with

  $$v \in \Sigma_{\text{pVal}} \cup \{a \cdot u \cdot \bar{a} \mid a \in \Sigma_c, u \in \mathrm{WM}(\widetilde{\Sigma}_{\text{JSON}})\},$$

  and
- the edges $((p_1, k_1, p_1'), (p_2, k_2, p_2'))$ if there exists an internal transition $p_1' \xrightarrow{\#} p_2$.

Lemma 20. *In a key graph $\mathrm{G}_{\mathcal{A}}$, there exists a path*

$$((p_1, k_1, p'_1)(p_2, k_2, p'_2) \dots (p_n, k_n, p'_n))$$

with $p_1 = q_0^{\mathcal{A}}$ if and only if there exist

- *a word $u = k_1 v_1 \# k_2 v_2 \# \dots \# k_n v_n$ such that each $k_i v_i$ is a key-value pair and u is a factor of a word in $\mathcal{L}_{<}(\mathcal{G})$, and*
- *a path $(q_0^{\mathcal{A}}, \varepsilon) \xrightarrow{u} (p'_n, \varepsilon) \in \mathit{sruns}(\mathcal{A})$ that decomposes as follows:*

$$\forall i \in \{1, \dots, n\} : (p_i, \varepsilon) \xrightarrow{k_i v_i} (p'_i, \varepsilon)$$

and

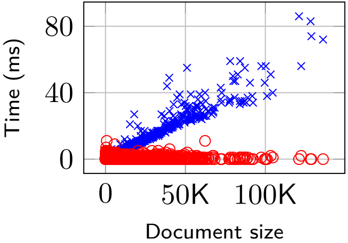
$$\forall i \in \{1, \dots, n-1\} : (p'_i, \varepsilon) \xrightarrow{\#} (p_{i+1}, \varepsilon).$$

Time	Membership	Equivalence	$\lvert Q \rvert$
9590.3 s	4246085.0	36.4	150.0

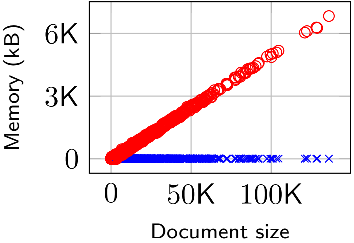
(a) Learning.

Time	Computation	Storage	Size
1715 s	11827 kB	419 kB	418

(b) Computation of the key graph.



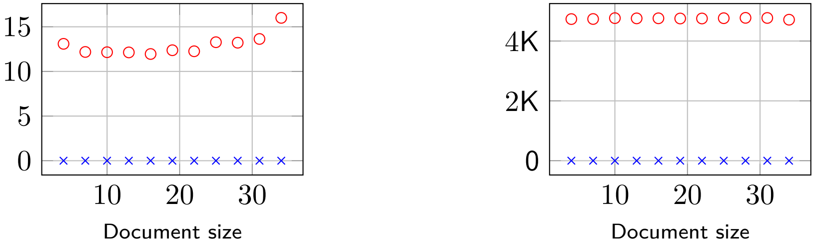
(c) Time usage (ms) for validation.



(d) Mem. usage (kB) for validation.

Figure 23: Results for **VIM plugins**, with $|\Sigma_{\text{key}}| = 16$. Red circles = classical algorithm. Blue crosses = our algorithm.

We use Boolean operations to force the classical algorithm to explore **multiple** branches, while our algorithm is **immediate**.



(a) Time usage (ms).

(b) Mem. usage (kB).

Figure 24: Results for a **worst case**, with $|\Sigma_{\text{key}}| = 1$. Red circles = classical algorithm. Blue crosses = our algorithm.

A run $p_0 \xrightarrow[u_1]{i_1} \cdots \xrightarrow[u_n]{i_n} p_n$ is said ***x-spanning*** (with $x \in X$) if it **begins** with a transition (re)starting x, **ends** with a $to[x]$-transition, and **no intermediate transition** restarts or stops x. That is,

- $u_1 = (x, c)$,
- $i_n = to[x]$,
- $u_j \neq (x, d)$ for all $j \in \{2, \ldots, n-1\}$ and $d \in \mathbb{N}^{>0}$, and
- $x \in \chi(p_j)$ for all $j \in \{2, \ldots, n-1\}$.

Let

$$\rho = (p_0, \kappa_0) \xrightarrow{d_1} (p_0, \kappa_0 - d_1) \xrightarrow[u_1]{i_1} (p_1, \kappa_1) \xrightarrow{d_2} \cdots$$
$$\xrightarrow[u_n]{i_n} (p_n, \kappa_n) \xrightarrow{d_{n+1}} (p_n, \kappa_n - d_{n+1}) \in \mathit{truns}(\mathcal{M})$$

be a timed run. A **block** of ρ is a pair $B = (k_1 k_2 \ldots k_m, \gamma)$ such that $i_{k_1}, i_{k_2}, \ldots, i_{k_m}$ is a **maximal** subsequence of actions of ρ such that

- $i_{k_1} \in I$,
- $p_{k_\ell - 1} \xrightarrow{i_{k_\ell} \cdots i_{k_{\ell+1}}} p_{k_{\ell+1}}$ is x-spanning for some timer x and for all $1 \leq \ell < m$, and
- γ is the **timer fate** of B defined as:

$$\gamma = \begin{cases} \bot & \text{if } i_{k_m} \text{ does not restart any timer} \\ \bullet & \text{if } i_{k_m} \text{ restarts a timer which is discarded (by some } i_\ell\text{, with } k_m < \ell \leq n \text{ or by the end of the run), when its value is zero} \\ \times & \text{otherwise.} \end{cases}$$

Theorem 21. *For every MMT $\mathcal{M}$, there exists a timed Mealy machine $\mathcal{N}$ such that $\mathcal{M}$ and $\mathcal{N}$ output the same timed words.*
The opposite direction does not hold.

Let $\mathcal{M} = (I, O, X, Q, q_0, \chi, \delta)$ be an MMT. Two valuations κ and κ' are said **timer-equivalent**, noted $\kappa \cong \kappa'$, if $\mathrm{dom}(\kappa) = \mathrm{dom}(\kappa')$ and the following hold:

- for all $x \in X$, $\lfloor \kappa(x) \rfloor = \lfloor \kappa'(x) \rfloor$ and
- for all $x \in X$, $\mathrm{frac}(\kappa(x)) = 0$ if and only if $\mathrm{frac}(\kappa'(x)) = 0$, and
- for all $x_1, x_2 \in X$, $\mathrm{frac}(\kappa(x_1)) \leq \mathrm{frac}(\kappa(x_2))$ if and only if $\mathrm{frac}(\kappa'(x_1)) \leq \mathrm{frac}(\kappa'(x_2))$.

A **timer region** for $\mathcal{M}$ is an equivalence class of timer valuations induced by $\cong$.
We lift the relation to configurations: $(q, \kappa) \cong (q', \kappa')$ if and only if $\kappa \cong \kappa'$ and $q = q'$.

The **region automaton** of $\mathcal{M}$ is denoted $\mathcal{R}(\mathcal{M})$ and such that

- its alphabet is $\Sigma = \{\tau\} \cup A(\mathcal{M})$,
- its set of states $Q^{\mathcal{R}(\mathcal{M})}$ is the quotient of the configurations by $\cong$, *i.e.*

$$Q^{\mathcal{R}(\mathcal{M})} = \{(q, \kappa) \mid q \in Q, \kappa \in \mathsf{Val}(\chi(q))\}_{/\cong},$$

- its initial state $q_0^{\mathcal{R}(\mathcal{M})}$ is the class of the initial configuration of $\mathcal{M}$, *i.e.*,

$$q_0^{\mathcal{R}(\mathcal{M})} = [\![(q_0^{\mathcal{M}}, \emptyset)]\!]_{\cong} = (q_0^{\mathcal{M}}, [\![\emptyset]\!]_{\cong})$$

  (by definition of $\cong$),
- its transition relation $\delta \subseteq S \times \Sigma \times S$ includes
  - $[\![(q, \kappa)]\!]_{\cong} \xrightarrow{\tau} [\![(q, \kappa - d)]\!]_{\cong}$ if $(q, \kappa) \xrightarrow{d} (q, \kappa - d)$ in $\mathcal{M}$ whenever $d > 0$, and
  - $[\![(q, \kappa)]\!]_{\cong} \xrightarrow{i} [\![(q', \kappa')]\!]_{\cong}$ if $(q, \kappa) \xrightarrow[u]{i} (q', \kappa')$ in $\mathcal{M}$.

Lemma 22. *Let $\mathcal{M}$ be an MMT and $\mathcal{R}(\mathcal{M})$ be its region automaton. For a timer $x \in X$, c_x denotes the largest constant to which x is updated in $\mathcal{M}$. Let $C = \max_{x \in X} c_x$. Then,*

$$\left| Q^{\mathcal{R}(\mathcal{M})} \right| \leq \left| Q^{\mathcal{M}} \right| \cdot |X|! \cdot 2^{|X|} \cdot (C+1)^{|X|}$$

and

$$\exists (q, \kappa) \xrightarrow{w} (q', \kappa') \in truns(\mathcal{M}) \Leftrightarrow \exists [\![(q, \kappa)]\!]_{\cong} \xrightarrow{w} [\![(q', \kappa')]\!]_{\cong} \in runs(\mathcal{R}(\mathcal{M})).$$

$$
\begin{aligned}
(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset) \\
&\xrightarrow{i/o} (q_1, x_1 = 1) \\
&\xrightarrow{1} (q_1, x_1 = 0) \\
&\xrightarrow{to[x_1]/o} (q_1, x_1 = 1) \\
&\xrightarrow{0} (q_1, x_1 = 1) \\
&\xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2) \\
&\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1) \\
&\xrightarrow{to[x_1]/o'} (q_0, \emptyset) \\
&\xrightarrow{0.5} (q_0, \emptyset).
\end{aligned}
$$

$$
\begin{aligned}
(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset)\\
&\xrightarrow{i/o} (q_1, x_1 = 1)\\
&\xrightarrow{1} (q_1, x_1 = 0)\\
&\xrightarrow{to[x_1]/o} (q_1, x_1 = 1)\\
&\xrightarrow{0} (q_1, x_1 = 1)\\
&\xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2)\\
&\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1)\\
&\xrightarrow{to[x_1]/o'} (q_0, \emptyset)\\
&\xrightarrow{0.5} (q_0, \emptyset).
\end{aligned}
$$

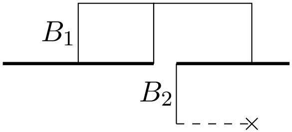


Figure 25: The **blocks** of the timed run.

$$
\begin{aligned}
(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset)\\
&\xrightarrow{i/o} (q_1, x_1 = 1)\\
&\xrightarrow{1} (q_1, x_1 = 0)\\
&\xrightarrow{to[x_1]/o} (q_1, x_1 = 1)\\
&\xrightarrow{0} (q_1, x_1 = 1)\\
&\xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2)\\
&\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1)\\
&\xrightarrow{to[x_1]/o'} (q_0, \emptyset)\\
&\xrightarrow{0.5} (q_0, \emptyset).
\end{aligned}
$$

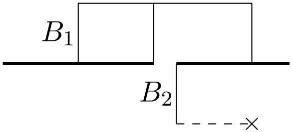


Figure 25: The **blocks** of the timed run.

Definition 23. *We have a* ***race*** *when two actions happen* ***simultaneously****.*

$$
\begin{aligned}
(q_0, \emptyset) &\xrightarrow{1} (q_0, \emptyset) \\
&\xrightarrow{i/o} (q_1, x_1 = 1) \\
&\xrightarrow{1} (q_1, x_1 = 0) \\
&\xrightarrow{to[x_1]/o} (q_1, x_1 = 1) \\
&\xrightarrow{0} (q_1, x_1 = 1) \\
&\xrightarrow{i/o'} (q_2, x_1 = 1, x_2 = 2) \\
&\xrightarrow{1} (q_2, x_1 = 0, x_2 = 1) \\
&\xrightarrow{to[x_1]/o'} (q_0, \emptyset) \\
&\xrightarrow{0.5} (q_0, \emptyset).
\end{aligned}
$$

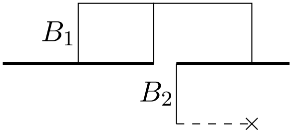


Figure 25: The **blocks** of the timed run.

Definition 23. *We have a **race** when two actions happen **simultaneously**.*

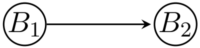
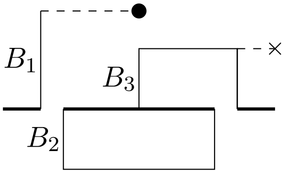


Figure 26: The **block graphs** of the **race**.



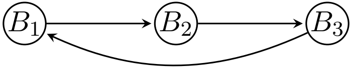
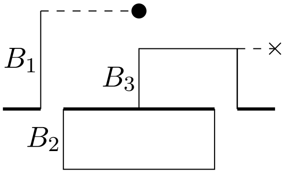


Figure 27: Blocks for a timed run in which **races** are **not** avoidable, and its **block graph**.



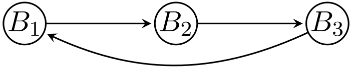
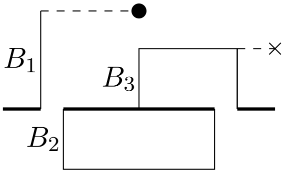


Figure 27: Blocks for a timed run in which **races** are **not** avoidable, and its **block graph**.

> ***Proposition 24.*** *A timed run has **unavoidable races** iff its block graph is **cyclic**.*



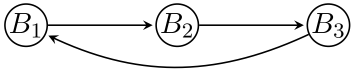


Figure 27: Blocks for a timed run in which **races** are **not** avoidable, and its **block graph**.

Proposition 24. *A timed run has **unavoidable races** iff its block graph is **cyclic**.*

Proposition 25. *There exists an **MSO** formula to decide whether there exists a timed run whose block graph is **cyclic**.*

Let B, B' be two blocks of a padded timed run ρ with timer fates γ and γ'. We say that B and B' **participate in a race** if:

- either there exist actions $i \in B$ and $i' \in B'$ such that the sum of the delays between i and i' in ρ is equal to zero, *i.e.*, no time elapses between them,
- or there exists an action $i \in B$ that is the first action along ρ to discard the timer started by the last action $i' \in B'$ and $\gamma' = \bullet$, *i.e.*, the timer of B' (re)started by i' reaches value zero when i discards it.

We also say that the actions i and i' participate in this race.