Active Learning of Mealy Machines with Timers

Véronique Bruyère, Bharat Garhewal, Guillermo A. Pérez,
Gaëtan Staquet, Frits W. Vaandrager

Theoretical computer science
University of Mons

Formal Techniques in Software Engineering
University of Antwerp

May 30, 2024

UMONS Université de Mons

fnrs LA LIBERTÉ DE CHERCHER

University of Antwerp

Many computer systems have **timing** constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, **real-time** systems.

Many computer systems have **timing** constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, **real-time** systems.

Well-known model for these systems: **timed Mealy machines**.

Many computer systems have **timing** constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, **real-time** systems.

Well-known model for these systems: **timed Mealy machines**.

In short: finite Mealy machines augmented with **clocks** that can be reset or used in guards along transitions and states.

Many computer systems have **timing** constraints:

- Network protocols;
- Schedulers;
- Embedded systems;
- In general, **real-time** systems.

Well-known model for these systems: **timed Mealy machines**.

In short: finite Mealy machines augmented with **clocks** that can be reset or used in guards along transitions and states.

BUT timed Mealy machines are hard to construct and understand.

We focus on systems that can be represented with **timers**: **Mealy machines with timers**.

Timed Mealy machines

-
-
-
-
-

Mealy machines with timers

-
-
-
-
-

1

We focus on systems that can be represented with **timers**: **Mealy machines with timers**.

Timed Mealy machines

- Clocks go **from 0 to infinity**;
-
-
-
-

Mealy machines with timers

- Timers go **from a given value to 0**;
-
-
-
-

1

We focus on systems that can be represented with **timers**: **Mealy machines with timers**.

Timed Mealy machines

- Clocks go **from 0 to infinity**;
- We can test the current value of the clocks;
-
-
-

Mealy machines with timers

- Timers go **from a given value to 0**;
- We can only test if a timer **is zero**;
-
-
-

1

We focus on systems that can be represented with **timers**: **Mealy machines with timers**.

Timed Mealy machines

- Clocks go **from 0 to infinity**;
- We can test the current value of the clocks;
- Timed Mealy machines are more expressive;
-
-

Mealy machines with timers

- Timers go **from a given value to 0**;
- We can only test if a timer **is zero**;
- Mealy machines with timers are more restrictive;
-
-

1

We focus on systems that can be represented with **timers**: **Mealy machines with timers**.

Timed Mealy machines

- Clocks go **from 0 to infinity**;
- We can test the current value of the clocks;
- Timed Mealy machines are more expressive;
- Well-known model;
-

Mealy machines with timers

- Timers go **from a given value to 0**;
- We can only test if a timer **is zero**;
- Mealy machines with timers are more restrictive;
- We previously studied some properties of Mealy machines with timers;[1]
-

[1] Bruyère, Pérez, et al., "Automata with Timers", 2023

We focus on systems that can be represented with **timers**: **Mealy machines with timers**.

Timed Mealy machines

- Clocks go **from 0 to infinity**;
- We can test the current value of the clocks;
- Timed Mealy machines are more expressive;
- Well-known model;
- Learning timed Mealy machines is challenging.

Mealy machines with timers

- Timers go **from a given value to 0**;
- We can only test if a timer **is zero**;
- Mealy machines with timers are more restrictive;
- We previously studied some properties of Mealy machines with timers;[1]
- This work: learning algorithm.

[1] Bruyère, Pérez, et al., "Automata with Timers", 2023

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

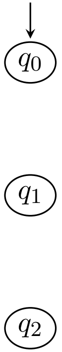


Figure 1: An MMT.

A **Mealy machine with timers** (MMT) is a tuple $\mathcal{M} = (X, I, O, Q, q_0, \delta)$ 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;
- δ is the transition function.

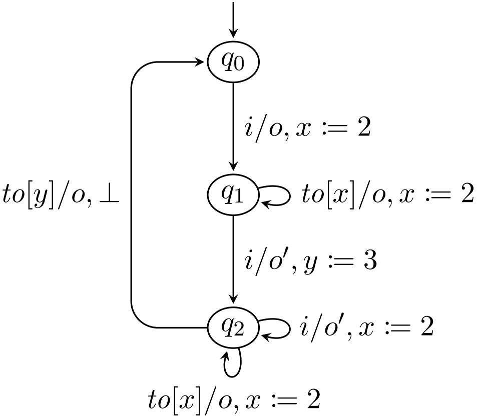


Figure 1: An MMT.

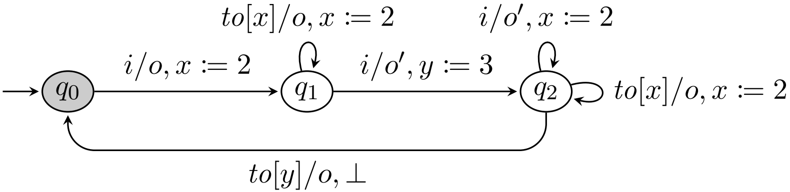


Figure 2: The same AT.

$(q_0, \emptyset)$

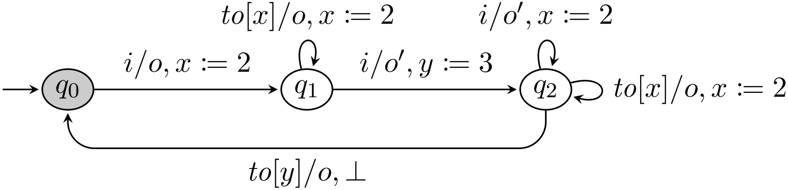


Figure 2: The same AT.

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

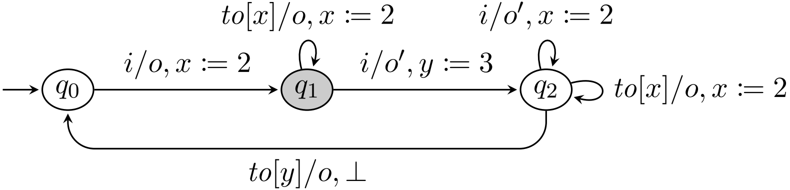


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x = 2)$$

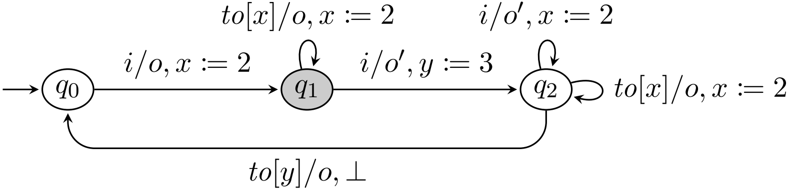


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x = 2) \xrightarrow{2} (q_1, x = 0)$$

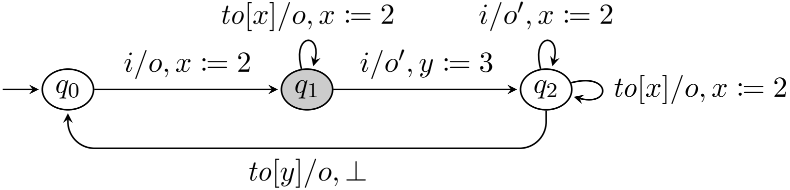


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x = 2) \xrightarrow{2} (q_1, x = 0) \xrightarrow{to[x]/o} (q_1, x = 2)$$

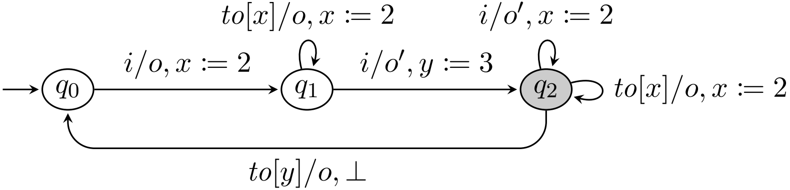


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x = 2) \xrightarrow{2} (q_1, x = 0) \xrightarrow{to[x]/o} (q_1, x = 2)$$
$$\xrightarrow{0} (q_1, x = 2) \xrightarrow{i/o'} (q_2, x = 2, y = 3)$$

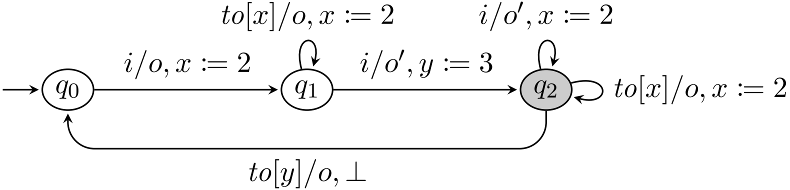


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x = 2) \xrightarrow{2} (q_1, x = 0) \xrightarrow{to[x]/o} (q_1, x = 2)$$
$$\xrightarrow{0} (q_1, x = 2) \xrightarrow{i/o'} (q_2, x = 2, y = 3) \xrightarrow{2} (q_2, x = 0, y = 1)$$

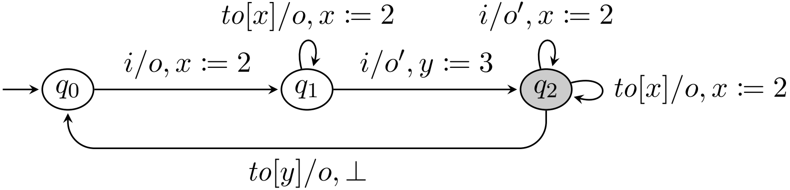


Figure 2: The same AT.

$$(q_0, \emptyset) \xrightarrow{1} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x = 2) \xrightarrow{2} (q_1, x = 0) \xrightarrow{to[x]/o} (q_1, x = 2)$$
$$\xrightarrow{0} (q_1, x = 2) \xrightarrow{i/o'} (q_2, x = 2, y = 3) \xrightarrow{2} (q_2, x = 0, y = 1)$$
$$\xrightarrow{i/o'} (q_2, x = 2, y = 1) \xrightarrow{0.5} (q_2, x = 1.5, y = 0.5).$$

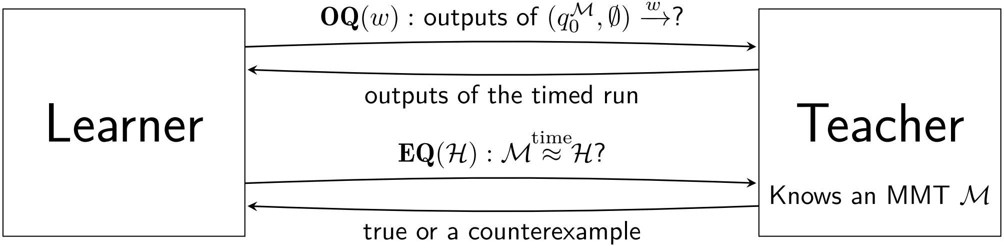


Figure 3: Adaptation of Angluin's framework[2] to MMTs.

[2] Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987; Shahbaz and Groz, "Inferring mealy machines", 2009.

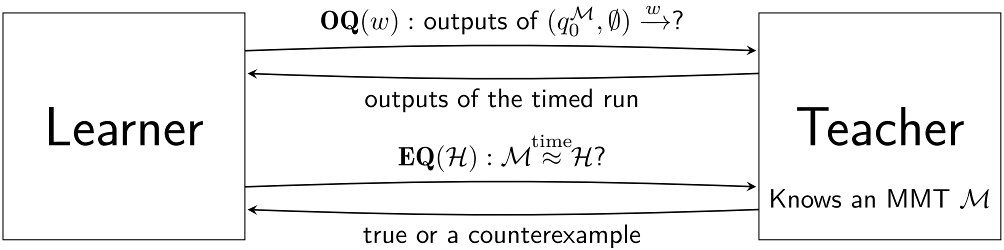


Figure 3: Adaptation of Angluin's framework[2] to MMTs.

Both queries are in the **timed** world... Cumbersome to use!

[2]Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987; Shahbaz and Groz, "Inferring mealy machines", 2009.

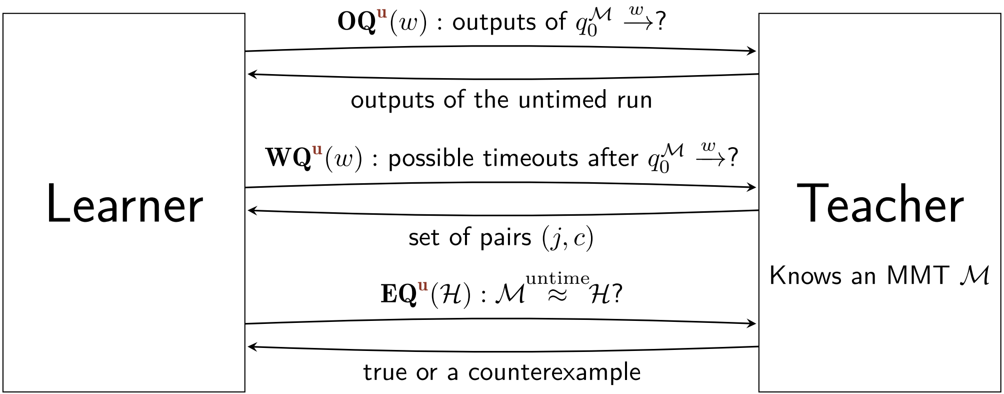


Figure 4: Untimed adaptation of Angluin's framework[3] to MMTs.

We stay in the **untimed** world!

[3] Angluin, "Learning Regular Sets from Queries and Counterexamples", 1987; Shahbaz and Groz, "Inferring mealy machines", 2009.

Proposition 1

The untimed queries can be implemented via a polynomial number of timed queries.

Proposition 1

The untimed queries can be implemented via a polynomial number of timed queries.

Does **not** hold for all MMTs!

Proposition 1

The untimed queries can be implemented via a polynomial number of timed queries.

Does **not** hold for all MMTs!
It holds when an MMT is “good”:

- timeouts are **observed** via their **outputs**,

Proposition 1

The untimed queries can be implemented via a polynomial number of timed queries.

Does **not** hold for all MMTs!
It holds when an MMT is "good":

- timeouts are **observed** via their **outputs**,
- for **every** untimed sequence of transitions, there exists a timed run using **exactly** this sequence of transitions…

Proposition 1

The untimed queries can be implemented via a polynomial number of timed queries.

Does **not** hold for all MMTs!
It holds when an MMT is “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 (see Bruyère, Pérez, et al., “Automata with Timers”, 2023).

Proposition 1

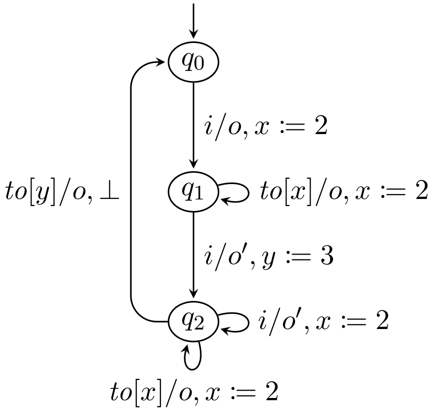
The untimed queries can be implemented via a polynomial number of timed queries.

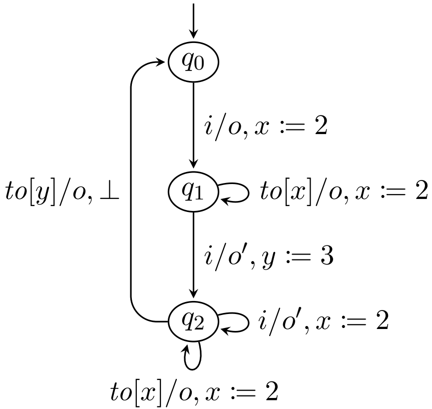
Does **not** hold for all MMTs!
It holds when an MMT is "good":

- 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 (see Bruyère, Pérez, et al., "Automata with Timers", 2023).

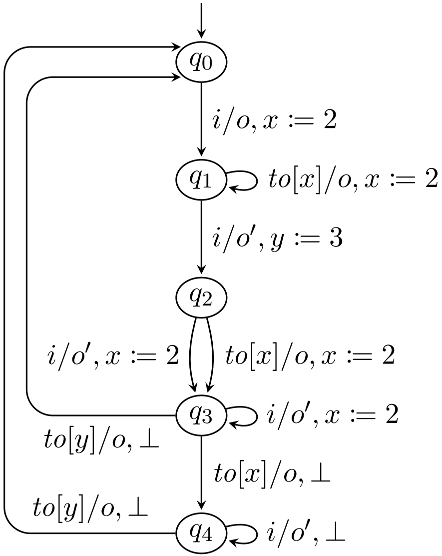
Proposition 2

It is possible to construct an MMT in which the second condition is satisfied.


q_0
$i/o, x := 2$
$to[y]/o, \bot$
q_1
$to[x]/o, x := 2$
$i/o', y := 3$
q_2
$i/o', x := 2$
$to[x]/o, x := 2$



$\rightsquigarrow$



We adapt $L^{\#}$ (active learning algorithm for Mealy machines[4]) to MMTs: $L^{\#}_{\mathsf{MMT}}$.

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

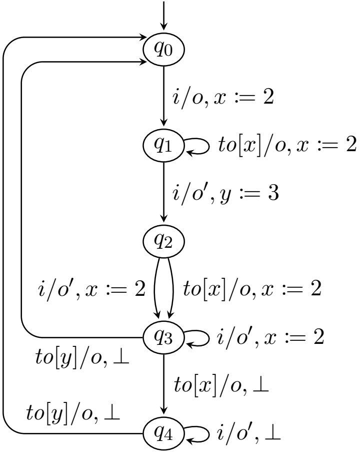
We adapt $L^{\#}$ (active learning algorithm for Mealy machines[4]) to MMTs: $L^{\#}_{\mathsf{MMT}}$.

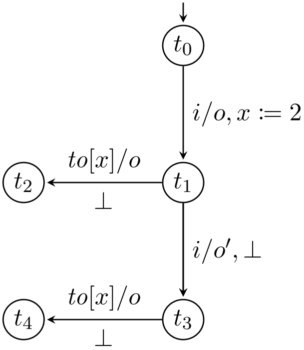
Theorem 3

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

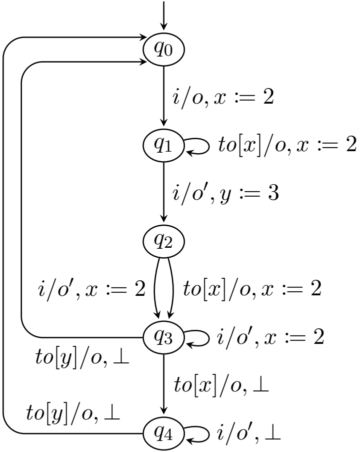
- *the $L^{\#}_{MMT}$ algorithm eventually terminates and returns an MMT $\mathcal{N}$ such that $\mathcal{M} \overset{\text{time}}{\approx} \mathcal{N}$ and whose size is polynomial in $\left|Q^{\mathcal{M}}\right|$ and factorial in $\left|X^{\mathcal{M}}\right|$, and*
- *in time and number of untimed queries polynomial in $\left|Q^{\mathcal{M}}\right|, |I|$, and ℓ, and factorial in $\left|X^{\mathcal{M}}\right|$.*

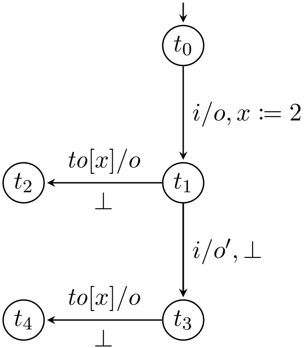
[4] Vaandrager et al., "A New Approach for Active Automata Learning Based on Apartness", 2022.





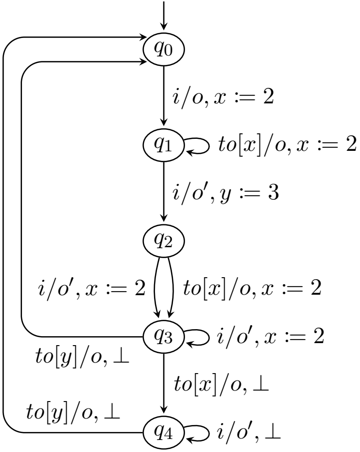
We want to add $i \cdot i \cdot i$ and the potential timeouts.

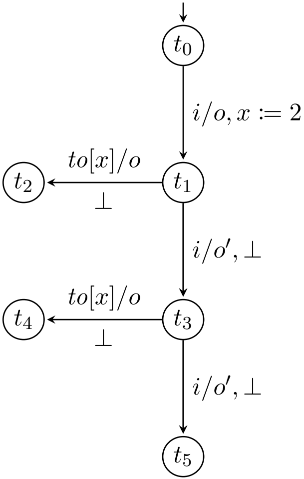




We want to add $i \cdot i \cdot i$ and the potential timeouts.

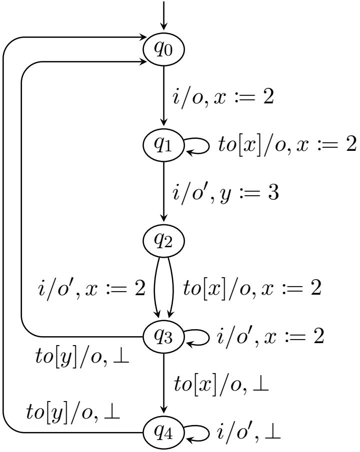
- $\mathbf{OQ^u}(i \cdot i \cdot i) \rightsquigarrow o \cdot o' \cdot o'$.

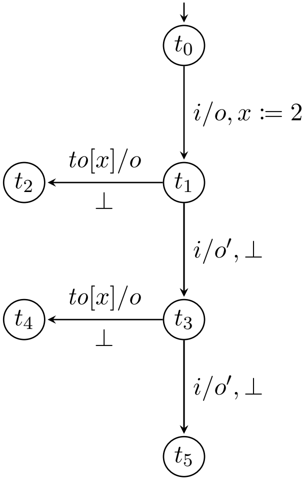




We want to add $i \cdot i \cdot i$ and the potential timeouts.

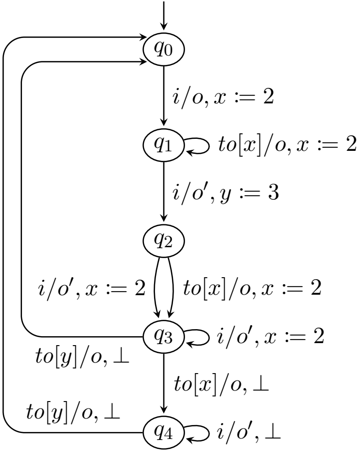
- $\mathbf{OQ^u}(i{\cdot}i{\cdot}i) \rightsquigarrow o{\cdot}o'{\cdot}o'$.
- So, $t_3 \xrightarrow[\bot]{i/o'} t_5$.

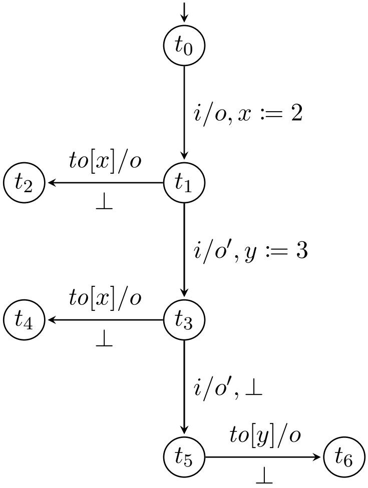




We want to add $i \cdot i \cdot i$ and the potential timeouts.

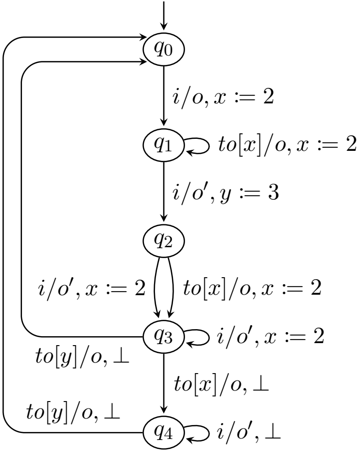
- $\mathbf{OQ^u}(i \cdot i \cdot i) \rightsquigarrow o \cdot o' \cdot o'$.
- So, $t_3 \xrightarrow[\bot]{i/o'} t_5$.
- $\mathbf{WQ^u}(i \cdot i \cdot i)$ $\rightsquigarrow \{(2,3),(3,2)\}$.

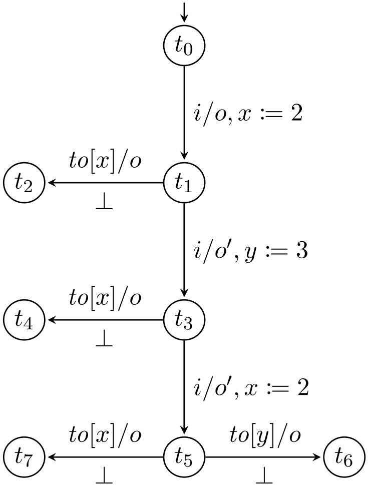




We want to add $i \cdot i \cdot i$ and the potential timeouts.

- $\mathbf{OQ^u}(i \cdot i \cdot i) \rightsquigarrow o \cdot o' \cdot o'$.
- So, $t_3 \xrightarrow[\bot]{i/o'} t_5$.
- $\mathbf{WQ^u}(i \cdot i \cdot i)$ $\rightsquigarrow \{(2,3), (3,2)\}$.
- So, $t_1 \xrightarrow{i} t_3$ starts a timer at constant 3.





We want to add $i \cdot i \cdot i$ and the potential timeouts.

- $\mathbf{OQ^u}(i \cdot i \cdot i) \rightsquigarrow o \cdot o' \cdot o'$.
- So, $t_3 \xrightarrow[\bot]{i/o'} t_5$.
- $\mathbf{WQ^u}(i \cdot i \cdot i)$ $\rightsquigarrow \{(2, 3), (3, 2)\}$.
- So, $t_1 \xrightarrow{i} t_3$ starts a timer at constant 3.
- And $t_3 \xrightarrow{i} t_5$ starts a timer at constant 2.

We implemented $L^{\#}_{\mathsf{MMT}}$ in Rust[5] and ran some experiments.

Model	$\lvert Q \rvert$	$\lvert I \rvert$	$\lvert X \rvert$	$\lvert \mathbf{WQ^u} \rvert$	$\lvert \mathbf{OQ^u} \rvert$	$\lvert \mathbf{EQ^u} \rvert$	Time[msecs]
AKM	4	5	1	22	35	2	684
CAS	8	4	1	60	89	3	1344
Light	4	2	1	10	13	2	302
PC	8	9	1	75	183	4	2696
TCP	11	8	1	123	366	8	3182
Train	6	3	1	32	28	3	1559
Running example	3	1	2	11	5	2	1039
FDDI 1-station	9	2	2	32	20	1	1105
Oven	12	5	1	907	317	3	9452
WSN	9	4	1	175	108	4	3291

[5] `https://gitlab.science.ru.nl/bharat/mmt_lsharp`.

Still work to be done:

- Further experiments with more timers,
- Simplify the learning algorithm as much as possible.

Still work to be done:

- Further experiments with more timers,
- Simplify the learning algorithm as much as possible.

Thank you!

For all details, see
Bruyère, Garhewal, et al., “Active Learning of Mealy Machines with Timers”, 2024.

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.
- 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.
- Bruyère, Véronique, Guillermo A. Pérez, et al. "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. Lecture Notes in Computer Science. Springer, 2023, pp. 33–49. DOI: 10.1007/978-3-031-42626-1_3.
- Shahbaz, Muzammil and Roland Groz. "Inferring mealy machines". In: *International Symposium on Formal Methods*. Springer. 2009, pp. 207–222.

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. Lecture Notes in Computer Science. Springer, 2022, pp. 223–243. DOI: 10.1007/978-3-030-99524-9_12.