Active Learning of Mealy Machines with Timers

Véronique Bruyère, Bharat Garhewal, Guillermo Pérez, Gaëtan Staquet,
Frits Vaandrager

August 27, 2025

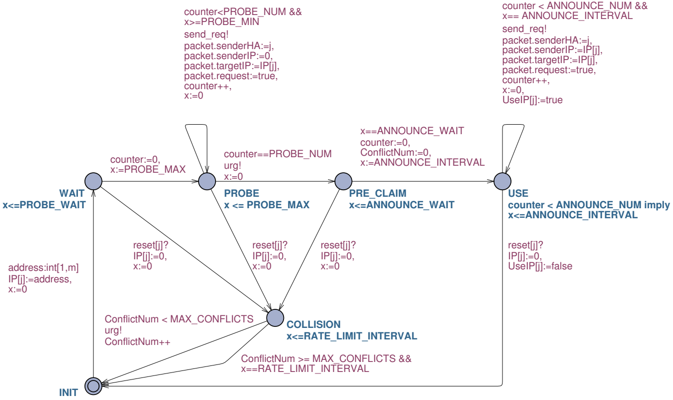
- Black-box active automata learning routinely spots bugs in implementations of major network protocols (TCP, TLS, SSH, MQTT, DTLS, BLE,..)

- Black-box active automata learning routinely spots bugs in implementations of major network protocols (TCP, TLS, SSH, MQTT, DTLS, BLE,..)
- Timing behavior is crucial in these protocols but mostly wiped under the carpet, since leading tools (LearnLib, AALpy, RAlib,..) cannot yet handle it.

- Black-box active automata learning routinely spots bugs in implementations of major network protocols (TCP, TLS, SSH, MQTT, DTLS, BLE,..)
- Timing behavior is crucial in these protocols but mostly wiped under the carpet, since leading tools (LearnLib, AALpy, RAlib,..) cannot yet handle it.
- Prototype tools for active learning of timed systems exist, but suffer from limited expressivity, scalability issues, and/or unrealistic assumptions.

- Black-box active automata learning routinely spots bugs in implementations of major network protocols (TCP, TLS, SSH, MQTT, DTLS, BLE,..)
- Timing behavior is crucial in these protocols but mostly wiped under the carpet, since leading tools (LearnLib, AALpy, RAlib,..) cannot yet handle it.
- Prototype tools for active learning of timed systems exist, but suffer from limited expressivity, scalability issues, and/or unrealistic assumptions.

Important but challenging research area!



Picture from J. Berendsen, B. Gebremichael, F.W. Vaandrager, and M. Zhang. Formal Specification and Analysis of Zeroconf using Uppaal. In ACM TECS 10(3), 2011.

Timed automata (Alur & Dill, 1990): dominant formal model for describing timing behavior, but inferring transition guards during learning requires many queries.

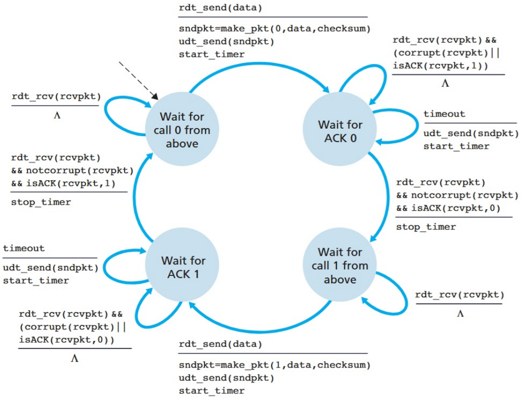


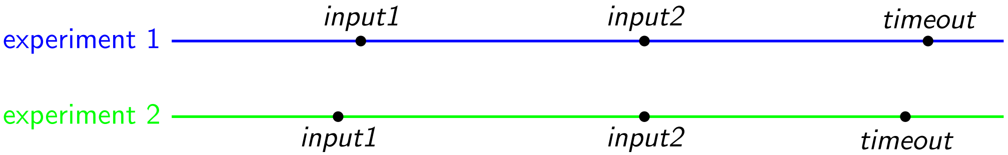
Figure 3.15 from J.F. Kurose and K.W. Ross. Computer Networking: A Top-Down Approach. Pearson. Sixth Edition, 2013

Automata with timers (Dill, 1989): often used by practitioners. Whereas in a timed automaton clock values increase when time advances, timer value in automata with timers decrease; when a timer value reaches 0, a timeout occurs.

During learning, we may efficiently determine which preceding transition cause a timeout by wiggling the timing of inputs[1,2]:

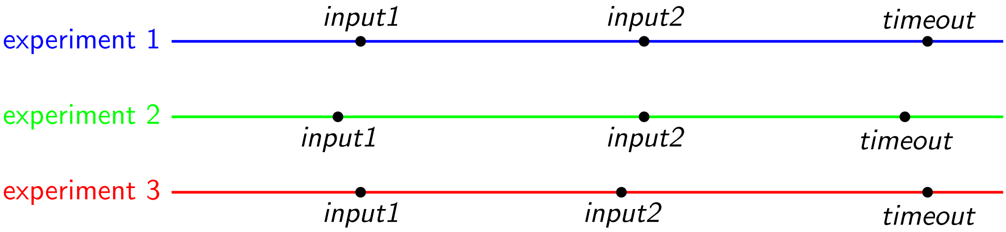
[1] B. Jonsson and F. Vaandrager. Learning Mealy Machines with Timers. Unpublished, 2018.
[2] V. Bruyére, G.A. Pérez, Gaëtan Staquet, and F. Vaandrager. Automata with Timers. FORMATS'23.

During learning, we may efficiently determine which preceding transition cause a timeout by wiggling the timing of inputs[1,2]:



[1] B. Jonsson and F. Vaandrager. Learning Mealy Machines with Timers. Unpublished, 2018.
[2] V. Bruyére, G.A. Pérez, Gaëtan Staquet, and F. Vaandrager. Automata with Timers. FORMATS'23.

During learning, we may efficiently determine which preceding transition cause a timeout by wiggling the timing of inputs[1,2]:



Conclusion: *timeout* caused by timer that was started by *input1*

[1] B. Jonsson and F. Vaandrager. Learning Mealy Machines with Timers. Unpublished, 2018.
[2] V. Bruyére, G.A. Pérez, Gaëtan Staquet, and F. Vaandrager. Automata with Timers. FORMATS'23.

Results presented today

1. An algorithm for active learning of Mealy machines with timers, obtained as an extension of the $L^{\#}$ algorithm[3].
2. Experiments with prototype implementation show that our algorithm is able to efficiently learn realistic benchmarks.

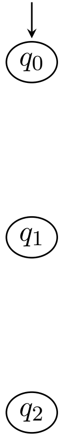
[3] F.W. Vaandrager, B. Garhewal, J. Rot, and T. Wißmann. A New Approach for Active Automata Learning Based on Apartness. TACAS'22.

Mealy machines with timers

Fix sets I and O of inputs resp. outputs.

Definition 1. *A* <u>*Mealy machine with timers (MMT)*</u> *is a tuple* $\mathcal{M} = (Q, q_0, X, \chi, \delta)$ *where*

- Q *is the finite set of* *states*
- $q_0 \in Q$ *is the* *initial state*

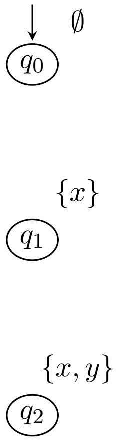


Mealy machines with timers

Fix sets I and O of inputs resp. outputs.

Definition 1. *A* *Mealy machine with timers (MMT)* *is a tuple* $\mathcal{M} = (Q, q_0, X, \chi, \delta)$ *where*

- Q *is the finite set of* *states*
- $q_0 \in Q$ *is the* *initial state*
- X *is the set of* *timers*
- $\chi : Q \to \mathcal{P}(X)$ *assigns* *active timers* *to states*

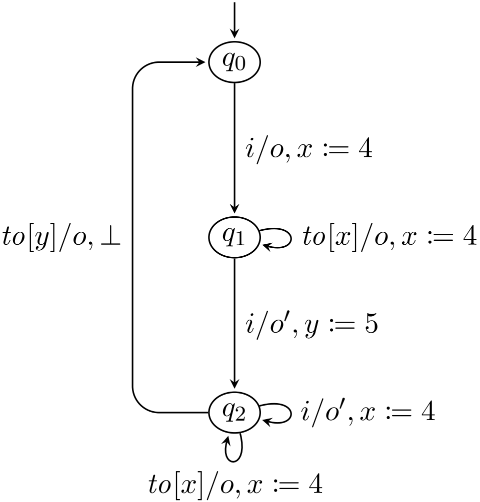


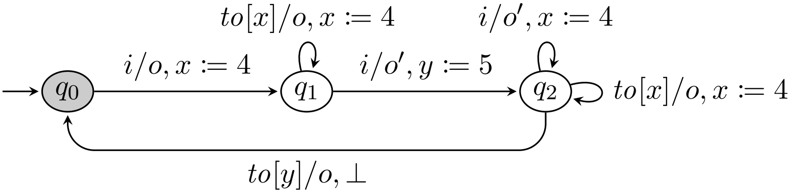
Mealy machines with timers

Fix sets I and O of inputs resp. outputs.

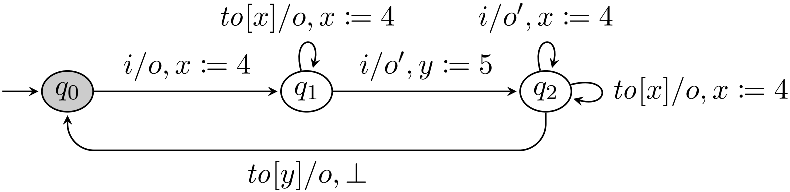
Definition 1. *A Mealy machine with timers (MMT) is a tuple* $\mathcal{M} = (Q, q_0, X, \chi, \delta)$ *where*

- Q *is the finite set of states*
- $q_0 \in Q$ *is the initial state*
- X *is the set of timers*
- $\chi : Q \to \mathcal{P}(X)$ *assigns active timers to states*
- δ *is the transition function*

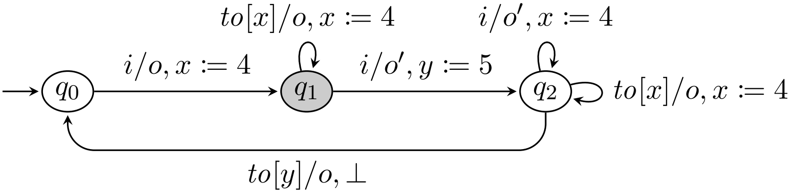




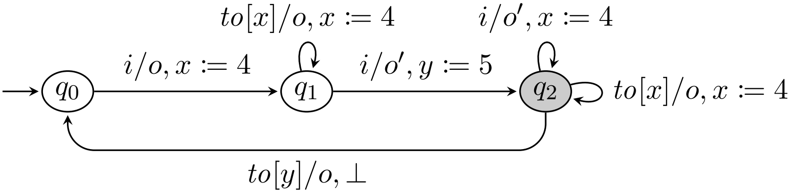
Timed run: $(q_0, \emptyset)$



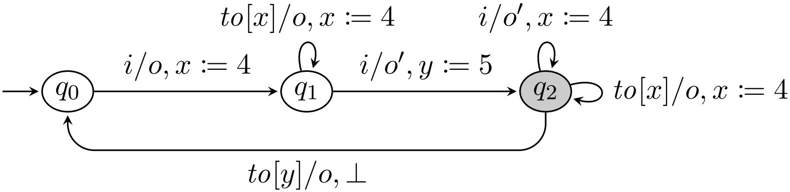
Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset)$



Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x = 4) \xrightarrow{0} (q_1, x = 4)$

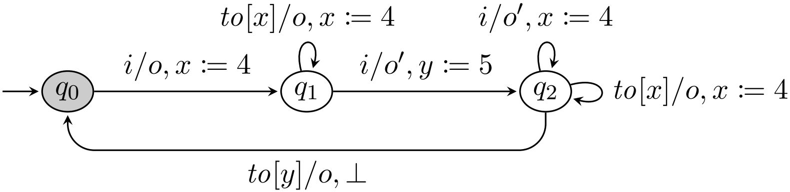


Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x{=}4) \xrightarrow{0} (q_1, x{=}4) \xrightarrow{i/o'} (q_2, x{=}4, y{=}5) \xrightarrow{4}$



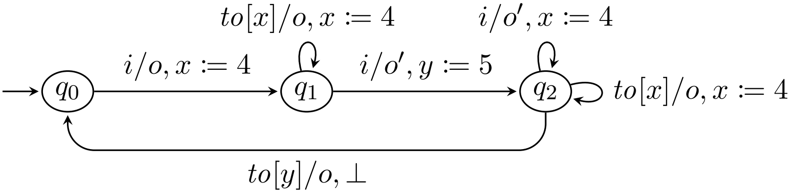
Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x=4) \xrightarrow{0} (q_1, x=4) \xrightarrow{i/o'} (q_2, x=4, y=5) \xrightarrow{4}$

$(q_2, x=0, y=1) \xrightarrow{to[x]/o} (q_2, x=4, y=1) \xrightarrow{1} (q_2, x=3, y=0)$



Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x=4) \xrightarrow{0} (q_1, x=4) \xrightarrow{i/o'} (q_2, x=4, y=5) \xrightarrow{4}$

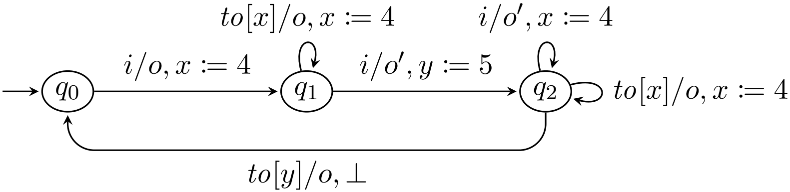
$(q_2, x=0, y=1) \xrightarrow{to[x]/o} (q_2, x=4, y=1) \xrightarrow{1} (q_2, x=3, y=0) \xrightarrow{to[y]/o} (q_0, \emptyset) \xrightarrow{0.4} (q_0, \emptyset)$



Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x=4) \xrightarrow{0} (q_1, x=4) \xrightarrow{i/o'} (q_2, x=4, y=5) \xrightarrow{4}$

$(q_2, x=0, y=1) \xrightarrow{to[x]/o} (q_2, x=4, y=1) \xrightarrow{1} (q_2, x=3, y=0) \xrightarrow{to[y]/o} (q_0, \emptyset) \xrightarrow{0.4} (q_0, \emptyset)$

Timed trace: $0.6 \quad i/o \quad 0 \quad i/o' \quad 4 \quad to/o \quad 1 \quad to/o \quad 0.4$

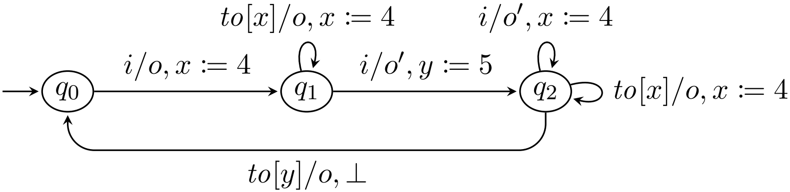


Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x=4) \xrightarrow{0} (q_1, x=4) \xrightarrow{i/o'} (q_2, x=4, y=5) \xrightarrow{4}$

$(q_2, x=0, y=1) \xrightarrow{to[x]/o} (q_2, x=4, y=1) \xrightarrow{1} (q_2, x=3, y=0) \xrightarrow{to[y]/o} (q_0, \emptyset) \xrightarrow{0.4} (q_0, \emptyset)$

Timed trace: $0.6 \quad i/o \quad 0 \quad i/o' \quad 4 \quad to/o \quad 1 \quad to/o \quad 0.4$

Output word: $o \quad o' \quad o \quad o$



Timed run: $(q_0, \emptyset) \xrightarrow{0.6} (q_0, \emptyset) \xrightarrow{i/o} (q_1, x=4) \xrightarrow{0} (q_1, x=4) \xrightarrow{i/o'} (q_2, x=4, y=5) \xrightarrow{4}$

$(q_2, x=0, y=1) \xrightarrow{to[x]/o} (q_2, x=4, y=1) \xrightarrow{1} (q_2, x=3, y=0) \xrightarrow{to[y]/o} (q_0, \emptyset) \xrightarrow{0.4} (q_0, \emptyset)$

Timed trace:	0.6	i/o	0	i/o'	4	to/o	1	to/o	0.4
Output word:		o		o'		o		o	
Symbolic input word:		i		i		$to[4,1]$		$to[5,2]$	

$\uparrow$

triggered by timer set to 4 by 1st event

Symbolic equivalence

Definition 2 (Symbolic equivalence).

- *We write* $L_{sym}(\mathcal{M})$ *for the set of symbolic input words accepted by MMT* $\mathcal{M}$.
- *For* $\mathtt{w} \in L_{sym}(\mathcal{M})$, $out^{\mathcal{M}}(\mathtt{w})$ *is the unique output word of timed runs that accept* $\mathtt{w}$.
- *Complete MMTs* $\mathcal{M}$ *and* $\mathcal{N}$ *are symbolically equivalent, noted* $\mathcal{M} \overset{sym}{\approx} \mathcal{N}$, *if* $L_{sym}(\mathcal{M}) = L_{sym}(\mathcal{N})$ *and, for each* $\mathtt{w} \in L_{sym}(\mathcal{M})$, $out^{\mathcal{M}}(\mathtt{w}) = out^{\mathcal{N}}(\mathtt{w})$.

Timed equivalence

Definition 3 (Timed equivalence). *Two MMTs $\mathcal{M}$ and $\mathcal{N}$ are timed trace equivalent, noted $\mathcal{M} \stackrel{tt}{\approx} \mathcal{N}$, if they have the same timed traces.*

Timed equivalence

Definition 3 (Timed equivalence). *Two MMTs $\mathcal{M}$ and $\mathcal{N}$ are timed trace equivalent, noted $\mathcal{M} \stackrel{tt}{\approx} \mathcal{N}$, if they have the same timed traces.*

Lemma 4. *If $\mathcal{M}$ and $\mathcal{N}$ are complete then $\mathcal{M} \stackrel{sym}{\approx} \mathcal{N}$ implies $\mathcal{M} \stackrel{tt}{\approx} \mathcal{N}$.*

Timed equivalence

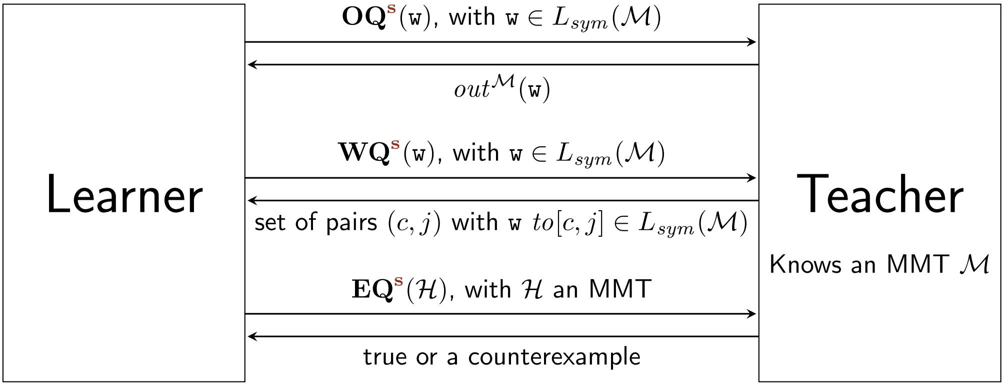
Definition 3 (Timed equivalence). *Two MMTs $\mathcal{M}$ and $\mathcal{N}$ are timed trace equivalent, noted $\mathcal{M} \stackrel{tt}{\approx} \mathcal{N}$, if they have the same timed traces.*

Lemma 4. *If $\mathcal{M}$ and $\mathcal{N}$ are complete then $\mathcal{M} \stackrel{sym}{\approx} \mathcal{N}$ implies $\mathcal{M} \stackrel{tt}{\approx} \mathcal{N}$.*

Lemma 5. *If $\mathcal{M}$ and $\mathcal{N}$ are complete and race-avoiding then $\mathcal{M} \stackrel{tt}{\approx} \mathcal{N}$ implies $\mathcal{M} \stackrel{sym}{\approx} \mathcal{N}$.*

Symbolic learning framework

A learner may pose three different types of queries to a teacher: output queries, wait queries, and equivalence queries.



Teaching assistants

Lemma 6. *For race-avoiding MMTs, the three types of symbolic queries can be realized via a polynomial number of concrete output and equivalence queries.*

Learning algorithm

$L^{\#}_{\mathsf{MMT}}$ is an algorithm for active learning of MMTs, generalizing $L^{\#}$.

Learning algorithm

$L^{\#}_{\mathsf{MMT}}$ is an algorithm for active learning of MMTs, generalizing $L^{\#}$.

Like $L^{\#}$, $L^{\#}_{\mathsf{MMT}}$ uses an observation tree as primary data structure: a tree shaped MMT $\mathcal{T}$ that contains the responses to all queries asked by the learner.

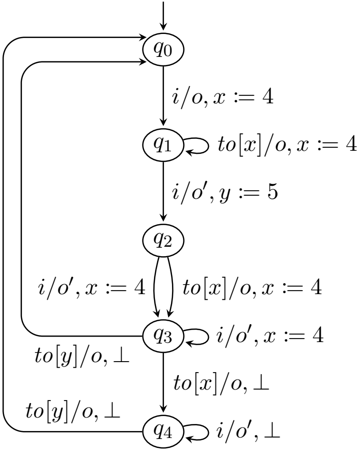
Learning algorithm

$L^{\#}_{\mathsf{MMT}}$ is an algorithm for active learning of MMTs, generalizing $L^{\#}$.

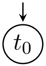
Like $L^{\#}$, $L^{\#}_{\mathsf{MMT}}$ uses an observation tree as primary data structure: a tree shaped MMT $\mathcal{T}$ that contains the responses to all queries asked by the learner.

Every (noninitial) state t_j of $\mathcal{T}$ has a dedicated timer x_j, which can only be started by the incoming transition of t_j.

Hidden MMT $\mathcal{M}$

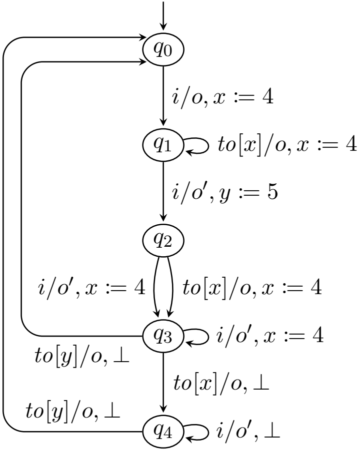


Observation tree $\mathcal{T}$

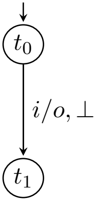


Symbolic queries

Hidden MMT $\mathcal{M}$



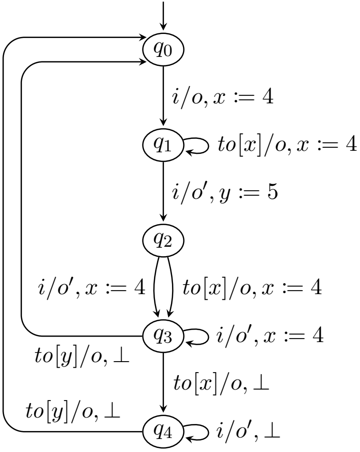
Observation tree $\mathcal{T}$



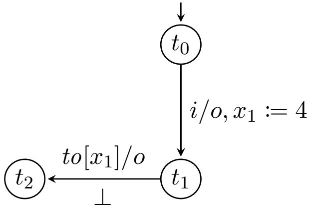
Symbolic queries

1. $\mathbf{OQ^s}(i)$

Hidden MMT $\mathcal{M}$



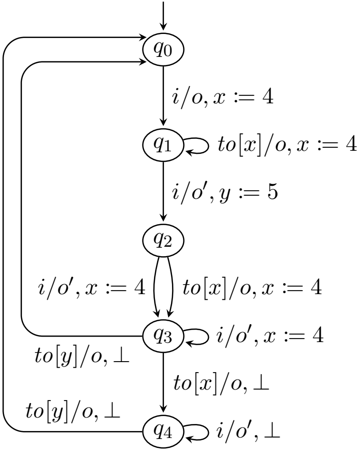
Observation tree $\mathcal{T}$



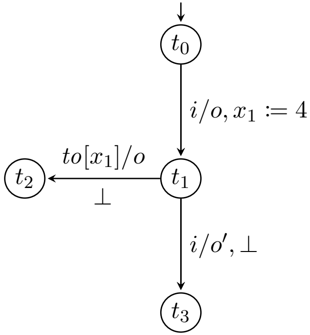
Symbolic queries

1. $\mathbf{OQ^s}(i)$
2. $\mathbf{WQ^s}(i)$

Hidden MMT $\mathcal{M}$



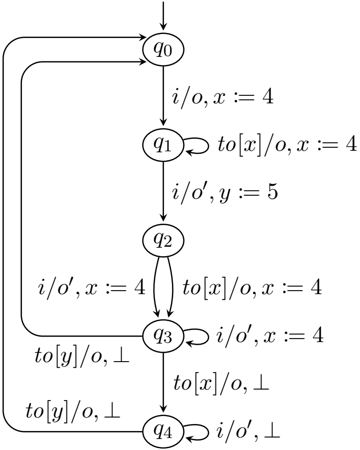
Observation tree $\mathcal{T}$



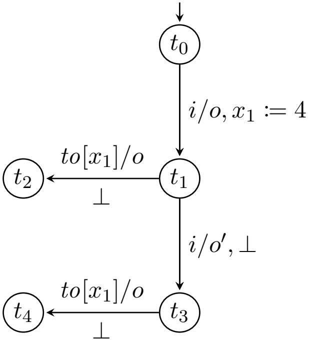
Symbolic queries

1. $\mathbf{OQ^s}(i)$
2. $\mathbf{WQ^s}(i)$
3. $\mathbf{OQ^s}(i \cdot i)$

Hidden MMT $\mathcal{M}$



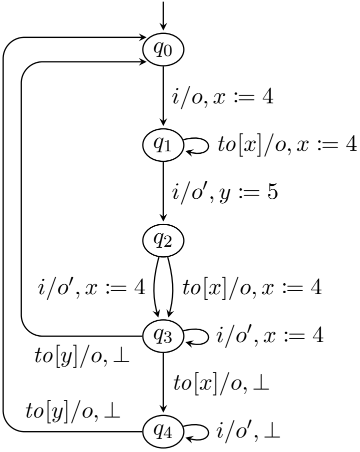
Observation tree $\mathcal{T}$



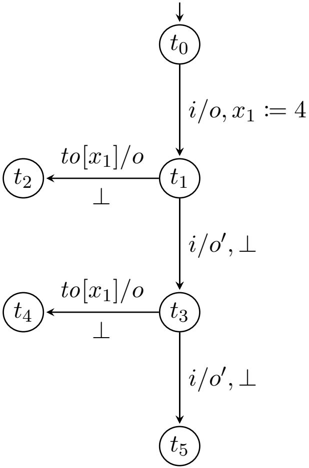
Symbolic queries

1. $\mathbf{OQ^s}(i)$
2. $\mathbf{WQ^s}(i)$
3. $\mathbf{OQ^s}(i \cdot i)$
4. $\mathbf{WQ^s}(i \cdot i)$

Hidden MMT $\mathcal{M}$



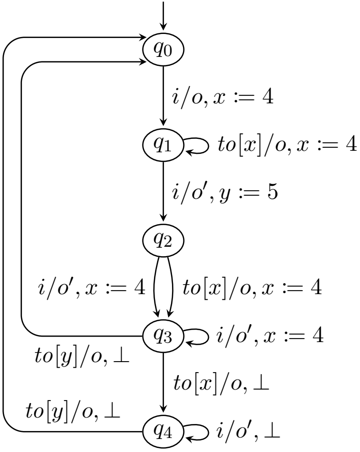
Observation tree $\mathcal{T}$



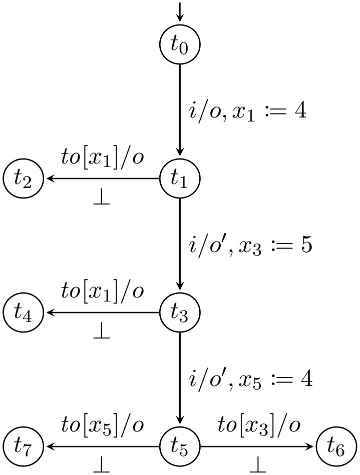
Symbolic queries

1. $\mathbf{OQ^s}(i)$
2. $\mathbf{WQ^s}(i)$
3. $\mathbf{OQ^s}(i \cdot i)$
4. $\mathbf{WQ^s}(i \cdot i)$
5. $\mathbf{OQ^s}(i \cdot i \cdot i)$

Hidden MMT $\mathcal{M}$



Observation tree $\mathcal{T}$



Symbolic queries

1. $\mathbf{OQ^s}(i)$
2. $\mathbf{WQ^s}(i)$
3. $\mathbf{OQ^s}(i \cdot i)$
4. $\mathbf{WQ^s}(i \cdot i)$
5. $\mathbf{OQ^s}(i \cdot i \cdot i)$
6. $\mathbf{WQ^s}(i \cdot i \cdot i)$

Algorithm

The algorithm maintains a partition of the observation tree states:

- The basis: a prefix-closed set of states that are pairwise apart, meaning they represent different states of the hidden MMT.
- The frontier: the immediate successors of basis states that are not in the basis
- The remaining states

Algorithm

The algorithm maintains a partition of the observation tree states:

- The basis: a prefix-closed set of states that are pairwise apart, meaning they represent different states of the hidden MMT.
- The frontier: the immediate successors of basis states that are not in the basis
- The remaining states

The following steps are performed repeatedly (in any order):

1. If a frontier state is apart from all basis states, add it to the basis

Algorithm

The algorithm maintains a partition of the observation tree states:

- The basis: a prefix-closed set of states that are pairwise apart, meaning they represent different states of the hidden MMT.
- The frontier: the immediate successors of basis states that are not in the basis
- The remaining states

The following steps are performed repeatedly (in any order):

1. If a frontier state is apart from all basis states, add it to the basis
2. If no wait query has been performed for a basis or frontier state, do it

Algorithm

The algorithm maintains a partition of the observation tree states:

- The basis: a prefix-closed set of states that are pairwise apart, meaning they represent different states of the hidden MMT.
- The frontier: the immediate successors of basis states that are not in the basis
- The remaining states

The following steps are performed repeatedly (in any order):

1. If a frontier state is apart from all basis states, add it to the basis
2. If no wait query has been performed for a basis or frontier state, do it
3. If basis state has no outgoing transition for some $i \in I$, add it

Algorithm

The algorithm maintains a partition of the observation tree states:

- The basis: a prefix-closed set of states that are pairwise apart, meaning they represent different states of the hidden MMT.
- The frontier: the immediate successors of basis states that are not in the basis
- The remaining states

The following steps are performed repeatedly (in any order):

1. If a frontier state is apart from all basis states, add it to the basis
2. If no wait query has been performed for a basis or frontier state, do it
3. If basis state has no outgoing transition for some $i \in I$, add it
4. If a frontier state r is not apart from two basis states p and q, use a witness for apartness of p and q, to establish apartness of r from either p or q

Algorithm

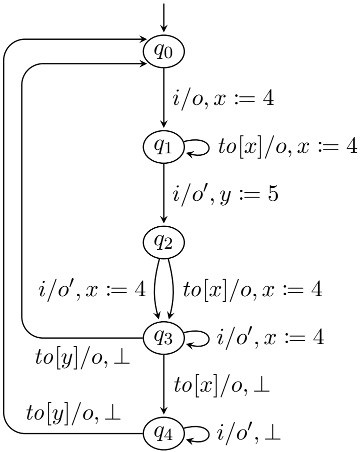
The algorithm maintains a partition of the observation tree states:

- The basis: a prefix-closed set of states that are pairwise apart, meaning they represent different states of the hidden MMT.
- The frontier: the immediate successors of basis states that are not in the basis
- The remaining states

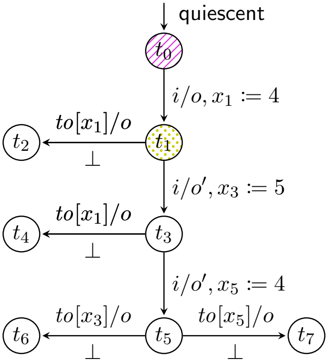
The following steps are performed repeatedly (in any order):

1. If a frontier state is apart from all basis states, add it to the basis
2. If no wait query has been performed for a basis or frontier state, do it
3. If basis state has no outgoing transition for some $i \in I$, add it
4. If a frontier state r is not apart from two basis states p and q, use a witness for apartness of p and q, to establish apartness of r from either p or q
5. Do an equivalence query with hypothesis obtained by folding transitions to frontier states back into the basis

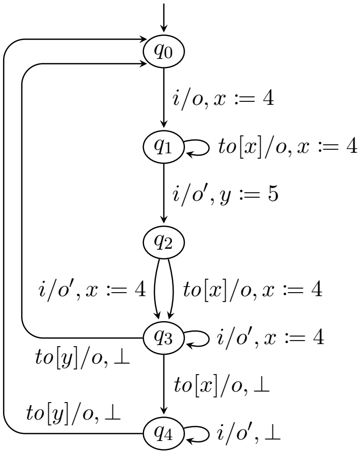
Hidden MMT $\mathcal{M}$



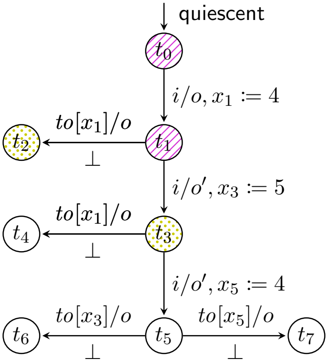
Observation tree $\mathcal{T}$



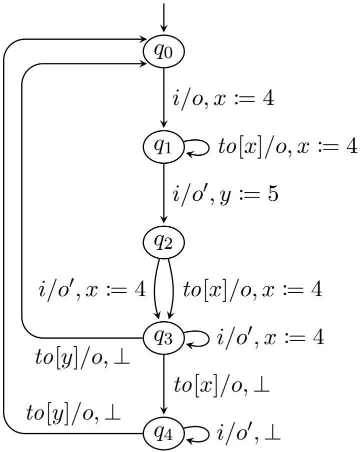
Hidden MMT $\mathcal{M}$



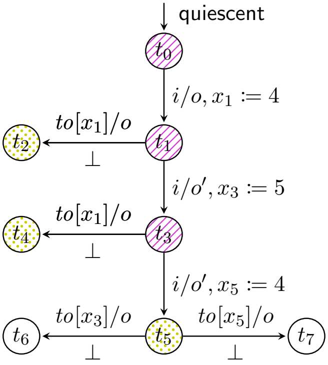
Observation tree $\mathcal{T}$



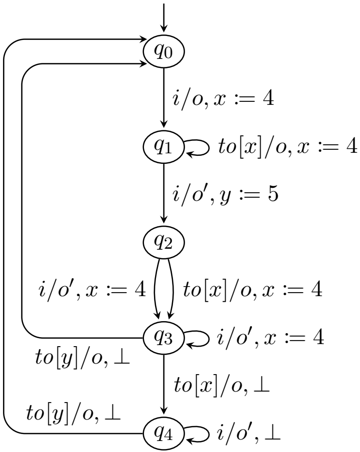
Hidden MMT $\mathcal{M}$



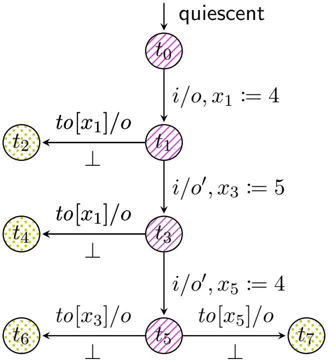
Observation tree $\mathcal{T}$



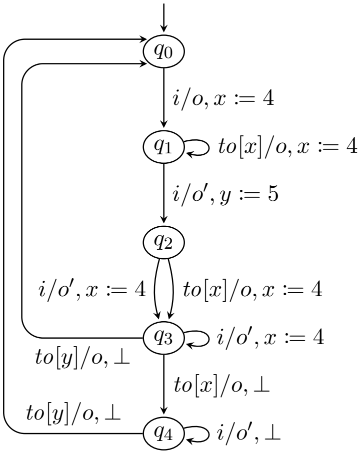
Hidden MMT $\mathcal{M}$



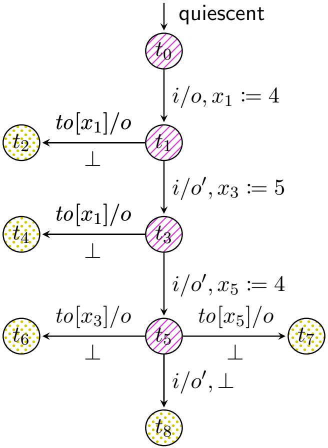
Observation tree $\mathcal{T}$



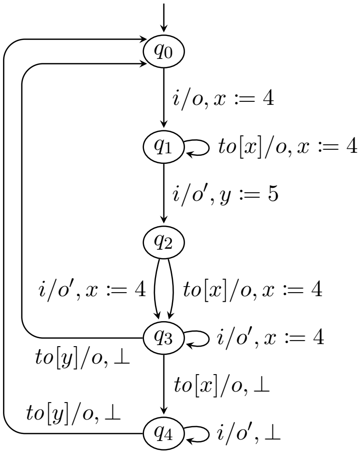
Hidden MMT $\mathcal{M}$



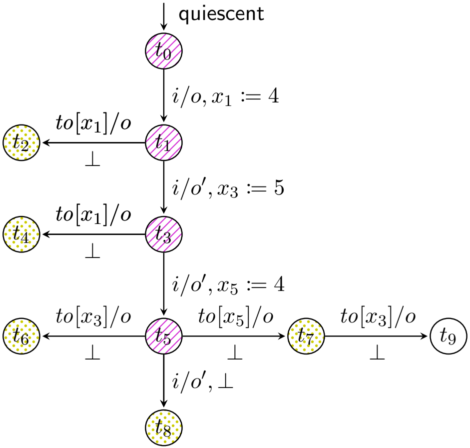
Observation tree $\mathcal{T}$



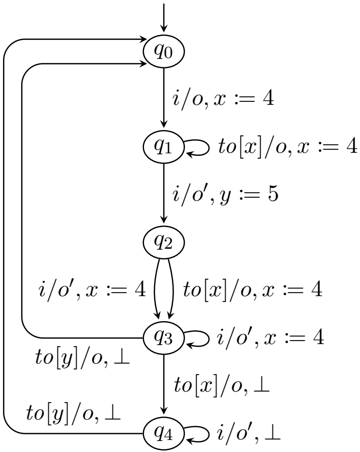
Hidden MMT $\mathcal{M}$



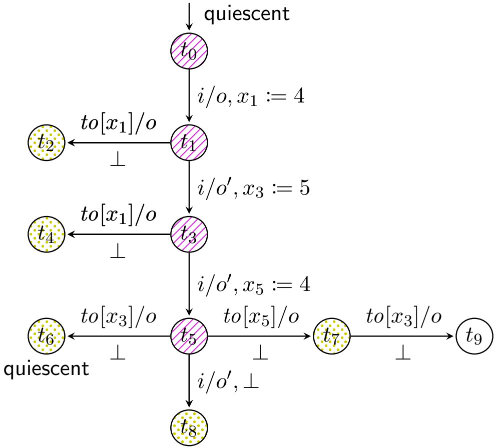
Observation tree $\mathcal{T}$



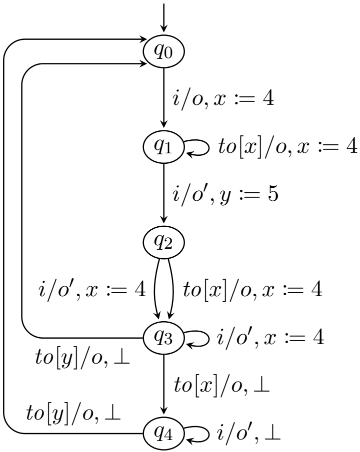
Hidden MMT $\mathcal{M}$



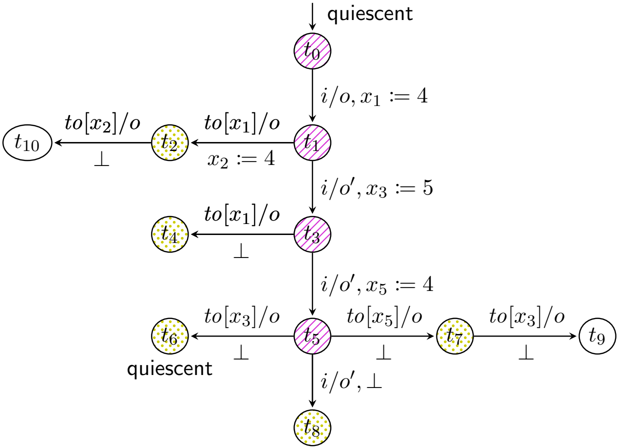
Observation tree $\mathcal{T}$



Hidden MMT $\mathcal{M}$

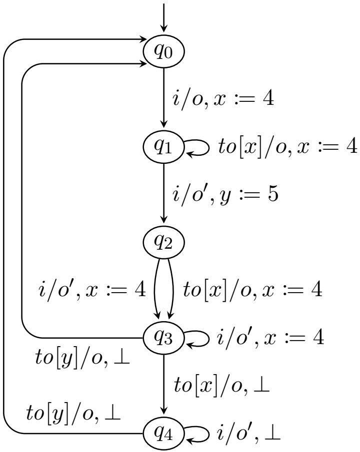


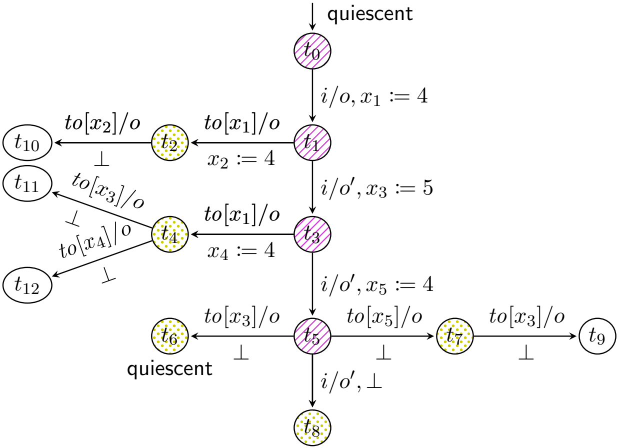
Observation tree $\mathcal{T}$



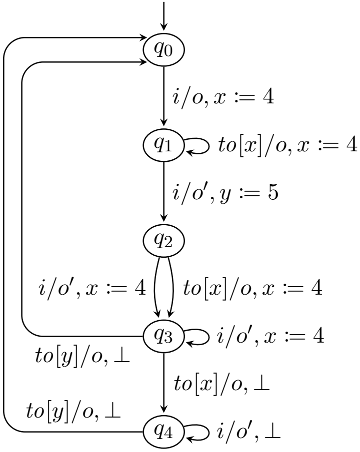
Hidden MMT $\mathcal{M}$

Observation tree $\mathcal{T}$

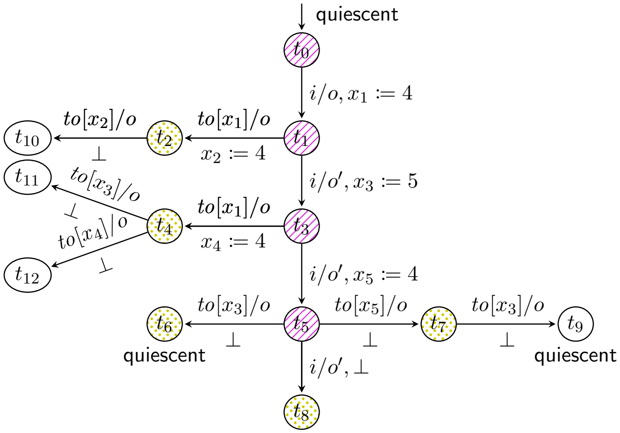




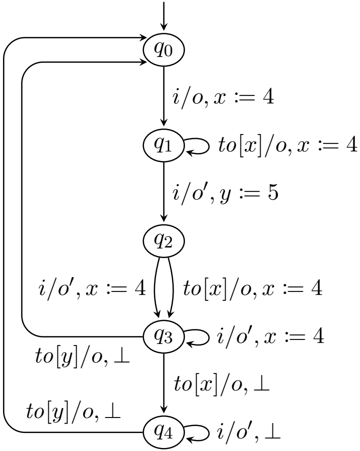
Hidden MMT $\mathcal{M}$



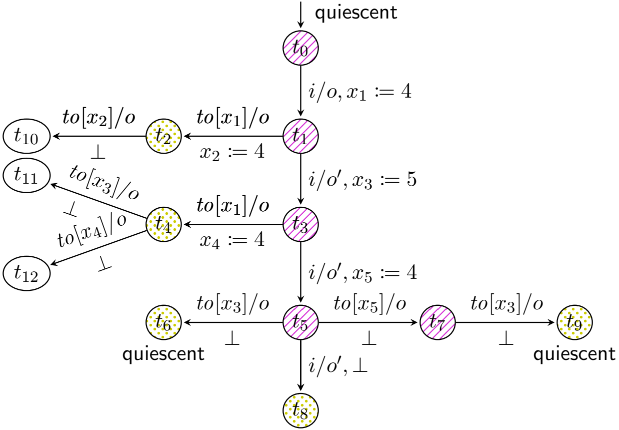
Observation tree $\mathcal{T}$



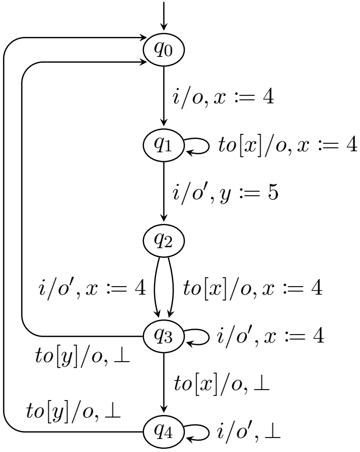
Hidden MMT $\mathcal{M}$



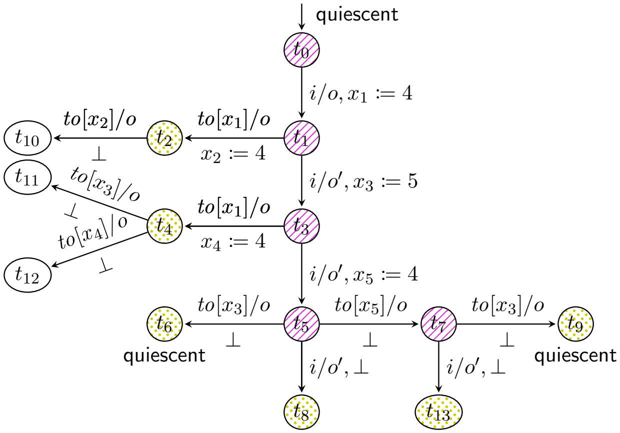
Observation tree $\mathcal{T}$



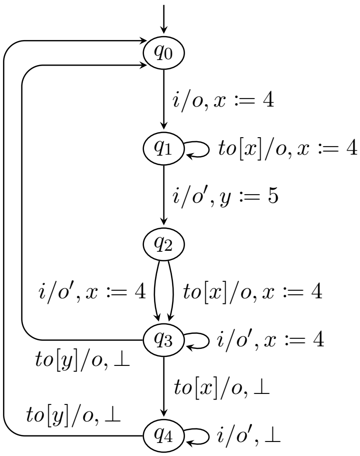
Hidden MMT $\mathcal{M}$



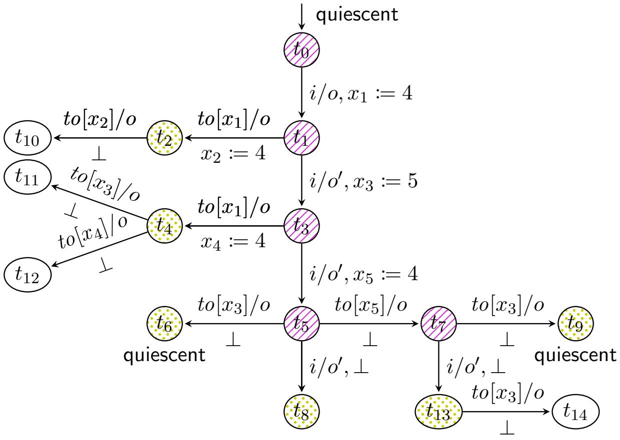
Observation tree $\mathcal{T}$



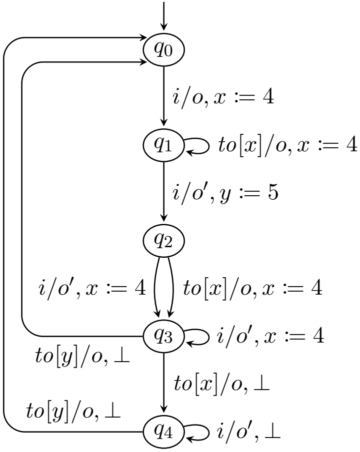
Hidden MMT $\mathcal{M}$



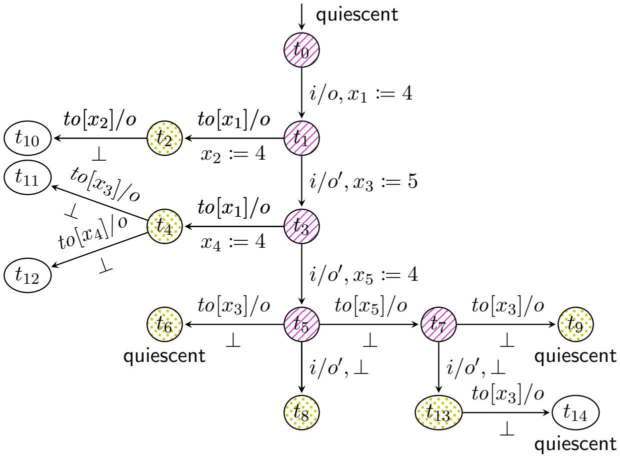
Observation tree $\mathcal{T}$



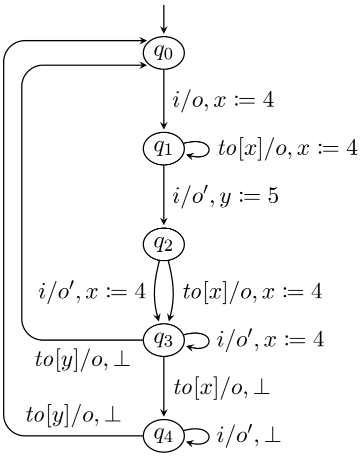
Hidden MMT $\mathcal{M}$



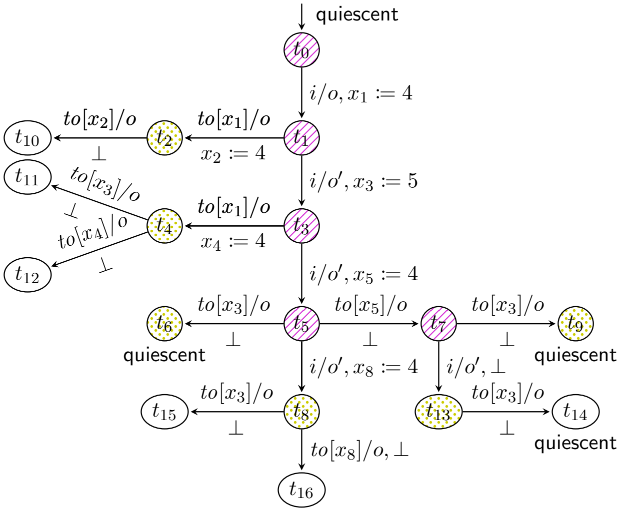
Observation tree $\mathcal{T}$



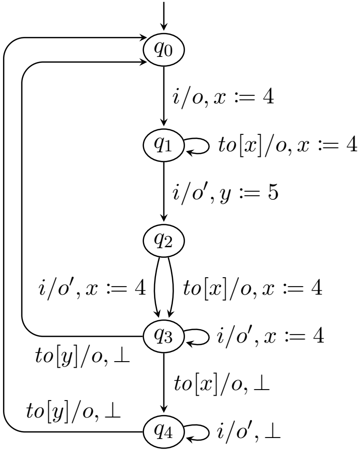
Hidden MMT $\mathcal{M}$



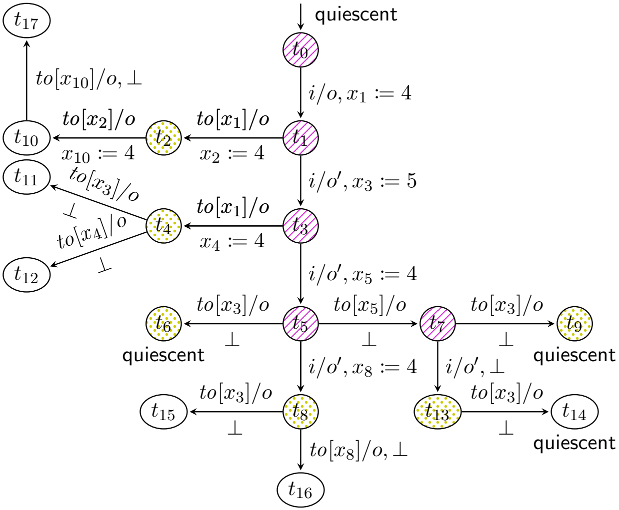
Observation tree $\mathcal{T}$



Hidden MMT $\mathcal{M}$



Observation tree $\mathcal{T}$



We implemented symbolic $L^{\#}_{\mathsf{MMT}}$ in Rust and ran some experiments[4].

Model	$\lvert Q \rvert$	$\lvert I \rvert$	$\lvert X \rvert$	$\lvert \mathbf{WQ}^{\mathsf{s}} \rvert$	$\lvert \mathbf{OQ}^{\mathsf{s}} \rvert$	$\lvert \mathbf{EQ}^{\mathsf{s}} \rvert$	Time[ms]	$\lvert \mathbf{MQ} \rvert$[5]	$\lvert \mathbf{EQ} \rvert$[7]
AKM	4	5	1	22	35	2	684	12263	11
CAS	8	4	1	60	89	3	1344	66067	17
Light	4	2	1	10	13	2	302	3057	7
PC	8	9	1	75	183	4	2696	245134	23
TCP	11	8	1	123	366	8	3182	11300	15
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	118193	8
Oven	12	5	1	907	317	3	9452	-	-
WSN	9	4	1	175	108	4	3291	-	-

[4] `https://doi.org/10.5281/zenodo.10647628`.

[5] Masaki Waga. *Active Learning of Deterministic Timed Automata with Myhill-Nerode Style Characterization*. CAV'23.

Future work

- Generalize MMT framework, allowing transitions to start/rename multiple timers

Future work

- Generalize MMT framework, allowing transitions to start/rename multiple timers
- Implement assistants that realize symbolic queries in terms of concrete queries

Future work

- Generalize MMT framework, allowing transitions to start/rename multiple timers
- Implement assistants that realize symbolic queries in terms of concrete queries
- Design timed testing algorithms to approximate concrete equivalence queries

Future work

- Generalize MMT framework, allowing transitions to start/rename multiple timers
- Implement assistants that realize symbolic queries in terms of concrete queries
- Design timed testing algorithms to approximate concrete equivalence queries
- Perform case studies with real systems and larger benchmarks!

Thank you!

For details, see `https://arxiv.org/abs/2403.02019v3`