ATVA 2013

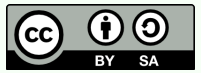
18th October 2013
Hà Nội, Việt Nam

Merge and Conquer

State Merging in Parametric Timed Automata

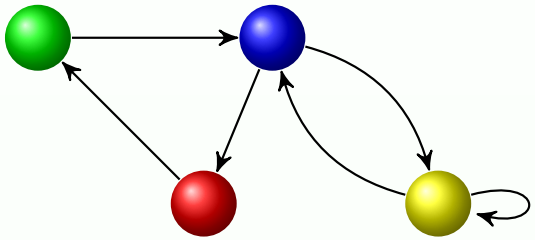
Étienne André, Laurent Fribourg, Romain Soulat

Laboratoire d'Informatique de Paris Nord
Université Paris 13, Sorbonne Paris Cité, France



Context: Verifying Complex Timed Systems

- Use formal methods



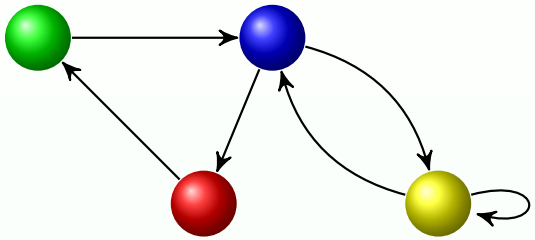
A model of the system

● is unreachable

A property to be satisfied

Context: Verifying Complex Timed Systems

- Use formal methods



?

$\models$

● is unreachable

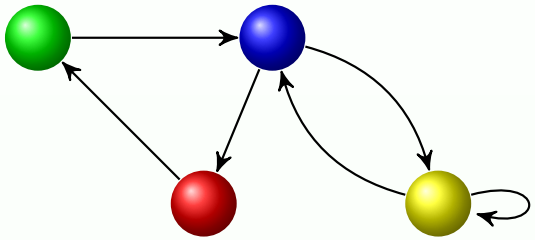
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Context: Verifying Complex Timed Systems

- Use formal methods



?

$\models$

● is unreachable

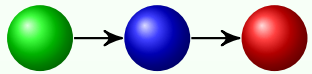
A model of the system

A property to be satisfied

- Question: does the model of the system satisfy the property?

Yes

No



Counterexample

Context: Parameter Synthesis

- Timed systems are characterized by a set of timing constants
  - "The packet transmission lasts for $50\,ms$"
  - "The sensor reads the value every $10\,s$"
- Verification for one set of constants does not usually guarantee the correctness for other values
- Challenges
  - Numerous verifications: is the system correct for any value within $[40; 60]$?
  - Optimization: until what value can we increase 10?
  - Robustness: What happens if 50 is implemented with 49.99?

Context: Parameter Synthesis

- Timed systems are characterized by a set of timing constants
  - "The packet transmission lasts for $50\,ms$"
  - "The sensor reads the value every $10\,s$"
- Verification for one set of constants does not usually guarantee the correctness for other values
- Challenges
  - Numerous verifications: is the system correct for any value within $[40; 60]$?
  - Optimization: until what value can we increase 10?
  - Robustness: What happens if 50 is implemented with 49.99?
- Parameter synthesis
  - Consider that timing constants are unknown constants (parameters)
  - Find good values for the parameters

Outline

Outline

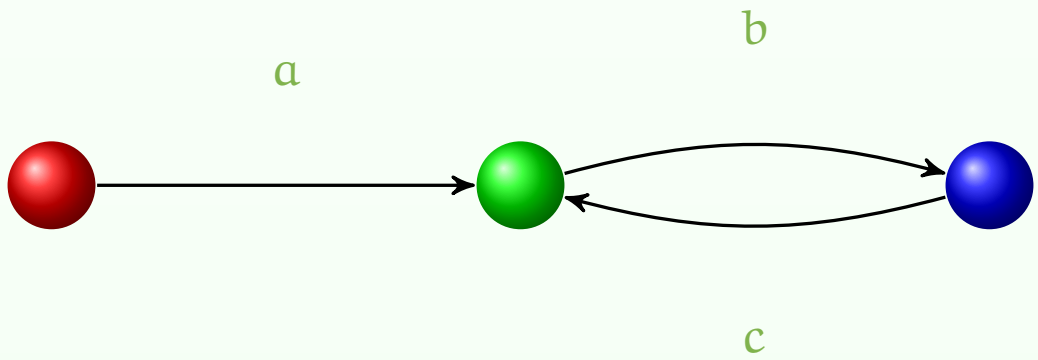
Timed Automaton

- Finite state automaton (sets of locations)



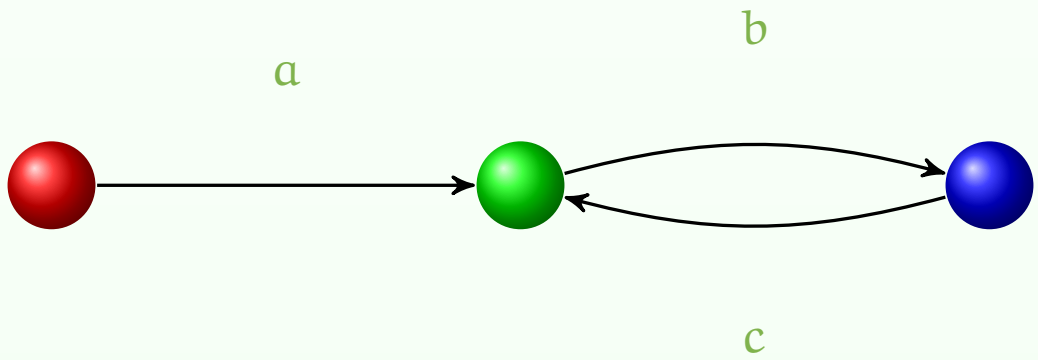
Timed Automaton

- Finite state automaton (sets of locations and actions)



Timed Automaton

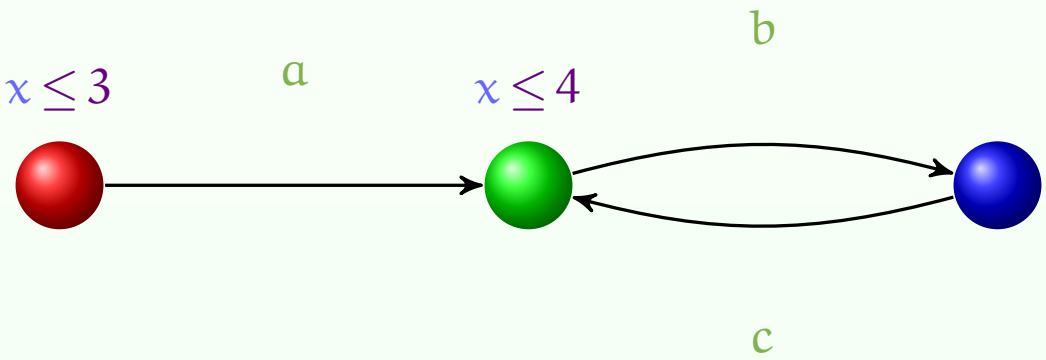
- Finite state automaton (sets of locations and actions) augmented with
  - A set X of clocks (i.e., real-valued variables evolving linearly at the same rate [Alur and Dill, 1994])



Timed Automaton

- Finite state automaton (sets of locations and actions) augmented with
  - A set X of clocks (i.e., real-valued variables evolving linearly at the same rate [Alur and Dill, 1994])

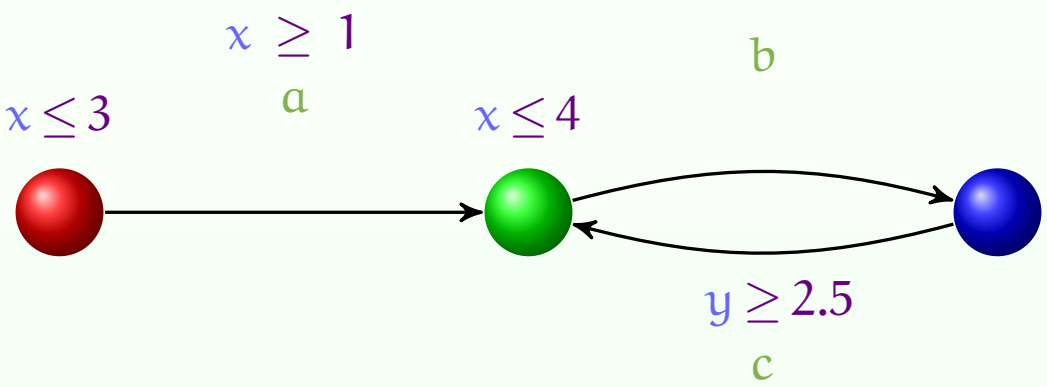
- Features
  - Location invariant: property to be verified to stay at a location



Timed Automaton

- Finite state automaton (sets of locations and actions) augmented with
  - A set X of clocks (i.e., real-valued variables evolving linearly at the same rate [Alur and Dill, 1994])

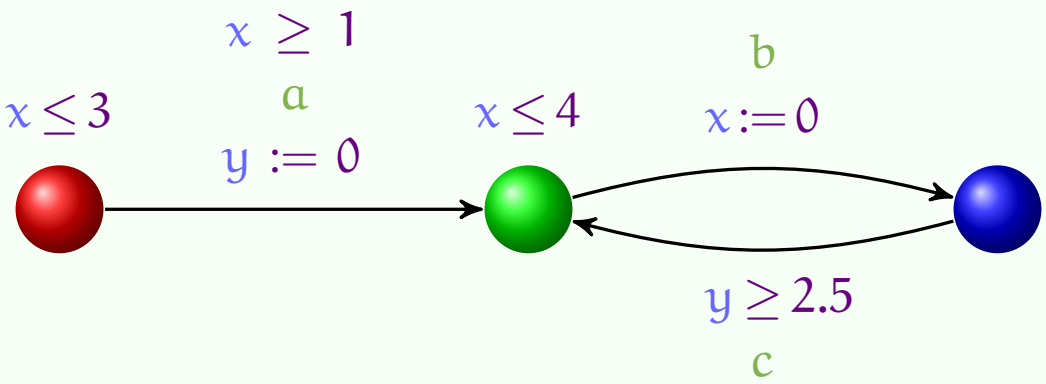
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition



Timed Automaton

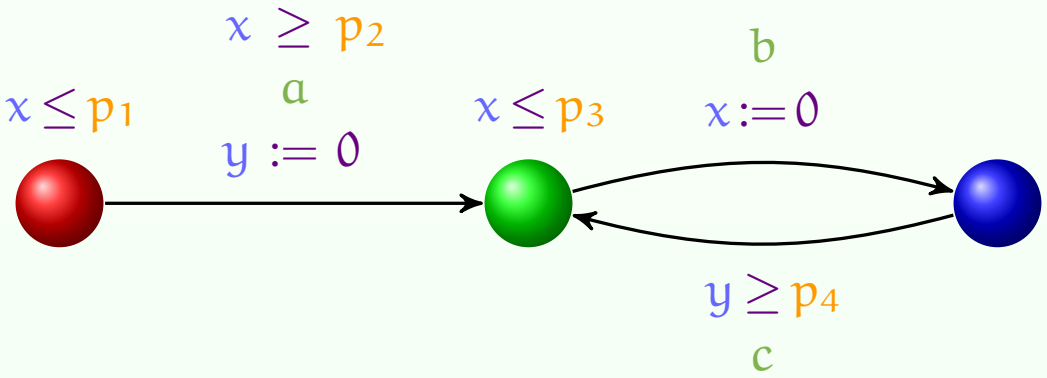
- Finite state automaton (sets of locations and actions) augmented with
  - A set X of clocks (i.e., real-valued variables evolving linearly at the same rate [Alur and Dill, 1994])

- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition



Parametric Timed Automaton (PTA)

- Finite state automaton (sets of locations and actions) augmented with
  - A set X of clocks (i.e., real-valued variables evolving linearly at the same rate [Alur and Dill, 1994])
  - A set P of parameters (i.e., unknown constants), used in guards and invariants [Alur et al., 1993b]
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition



Semantics of Parametric Timed Automata

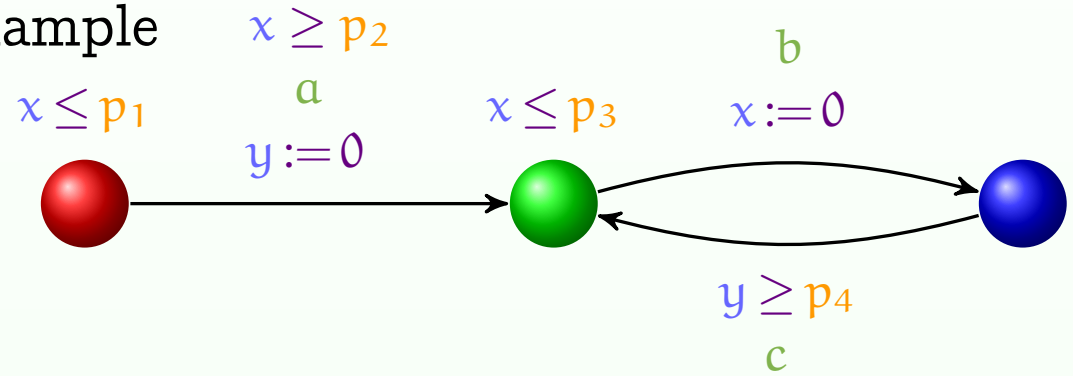
- State of a PTA: pair (l, C), where
  - l is a location (e.g., ),
  - C is a constraint (conjunction of inequalities) over X and P

Semantics of Parametric Timed Automata

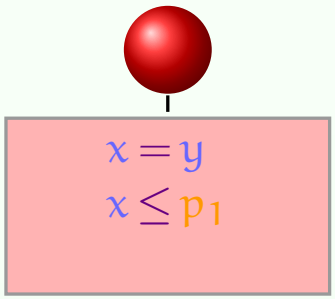
- State of a PTA: pair (l, C), where
  - l is a location (e.g., ),
  - C is a constraint (conjunction of inequalities) over X and P
- Path: alternating sequence of states and actions

Semantics of Parametric Timed Automata

- State of a PTA: pair (l, C), where
  - l is a location (e.g., ),
  - C is a constraint (conjunction of inequalities) over X and P
- Path: alternating sequence of states and actions
- Example

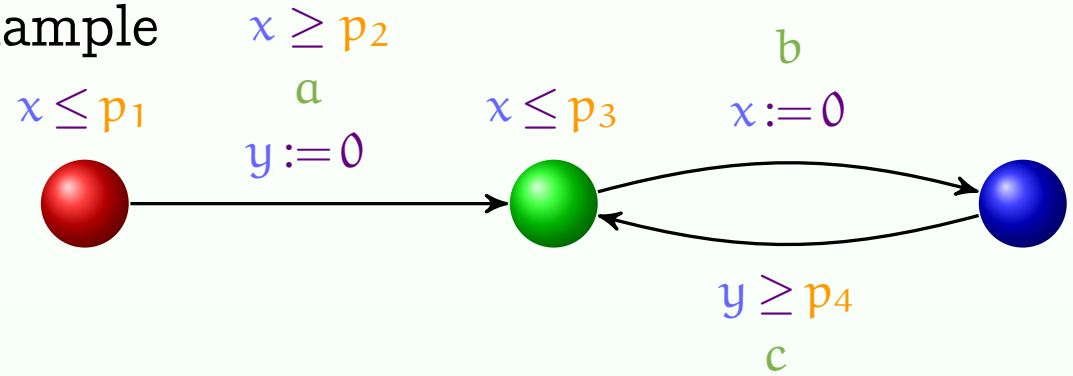


  - Possible path for this PTA

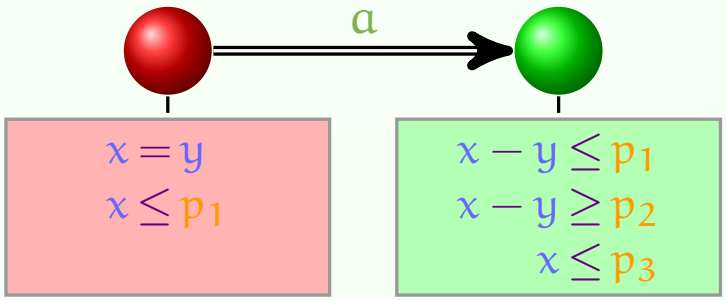


Semantics of Parametric Timed Automata

- State of a PTA: pair (l, C), where
  - l is a location (e.g., ),
  - C is a constraint (conjunction of inequalities) over X and P
- Path: alternating sequence of states and actions
- Example

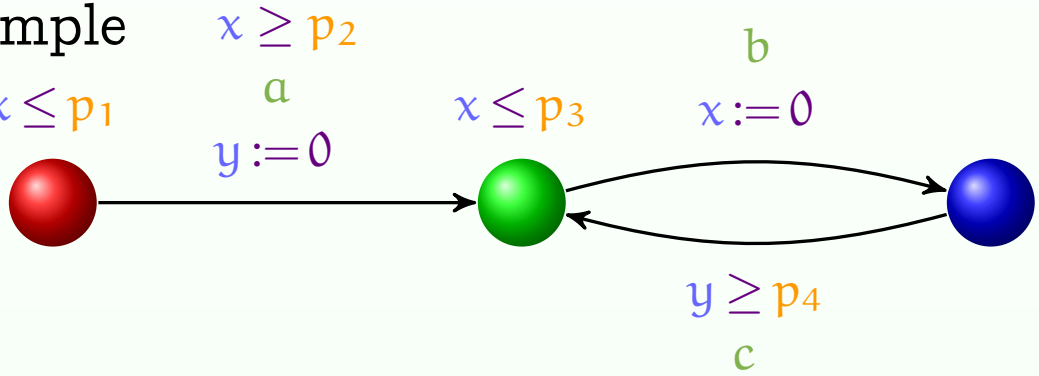


  - Possible path for this PTA

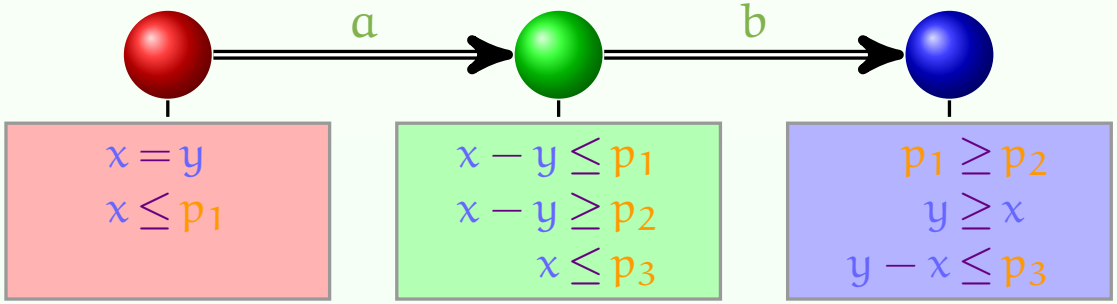


Semantics of Parametric Timed Automata

- State of a PTA: pair (l, C), where
  - l is a location (e.g., ),
  - C is a constraint (conjunction of inequalities) over X and P
- Path: alternating sequence of states and actions
- Example

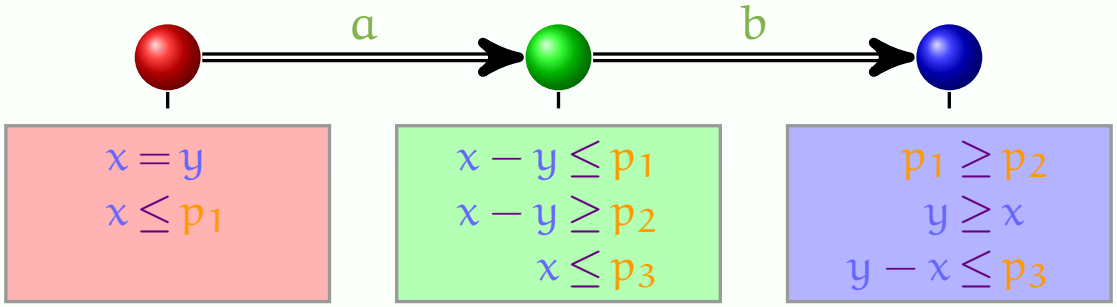


  - Possible path for this PTA



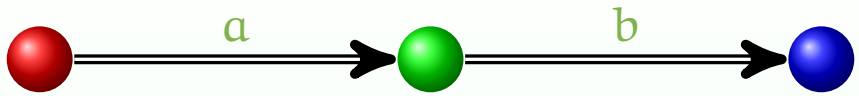
Traces

- Trace over a PTA: time-abstract path
  - Finite alternating sequence of locations and actions



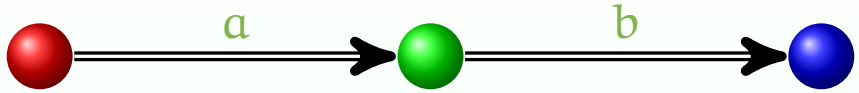
Traces

- Trace over a PTA: time-abstract path
  - Finite alternating sequence of locations and actions

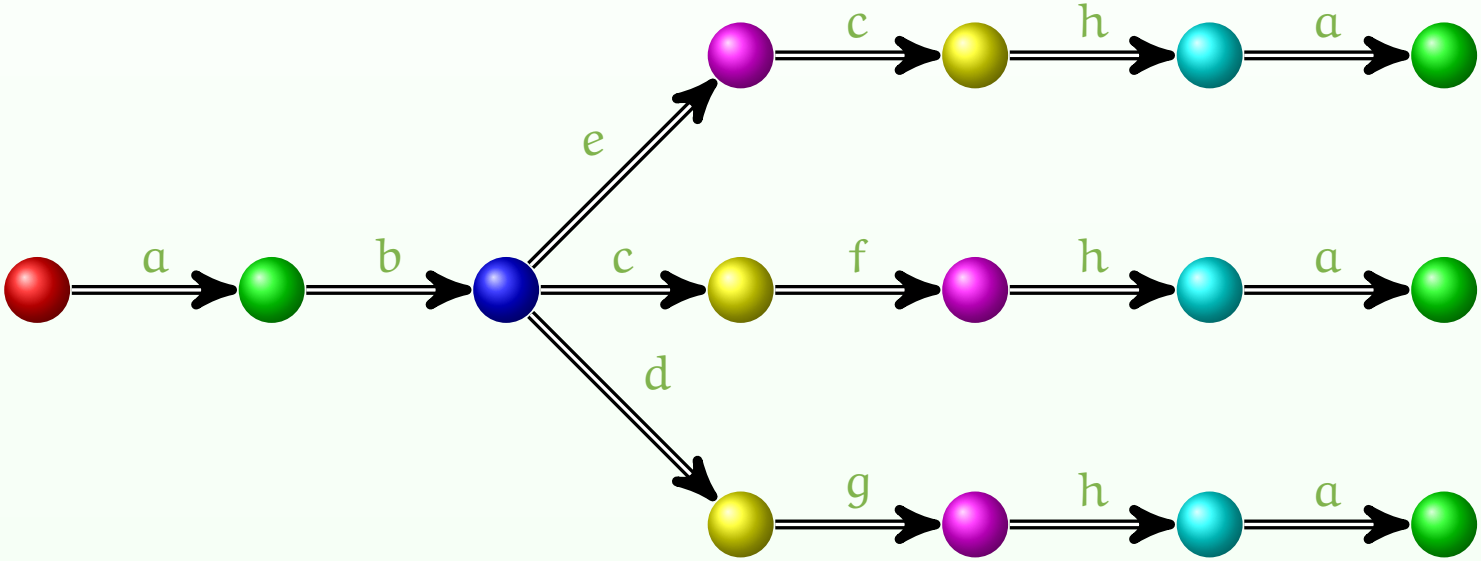


Traces

- Trace over a PTA: time-abstract path
  - Finite alternating sequence of locations and actions



- Trace set of a PTA: set of all its traces

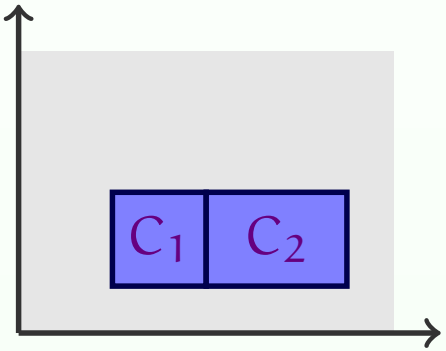


Outline

State Merging

- In timed automata, two states $(\bullet, C_1)$ and $(\bullet, C_2)$ are mergeable if
  1. Their discrete part is equal ($\bullet = \bullet$), and
  2. Their union $C_1 \cup C_2$ is convex

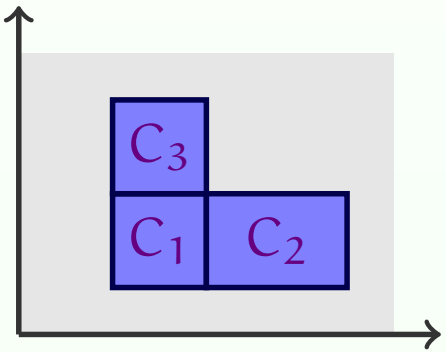
  Their merging is $(\bullet, C_1 \cup C_2)$ [David, 2005, David, 2006]



- Preserves safety properties

State Merging: Implementation

- States in timed automata are often encoded using difference bound matrices (DBMs)
- The mergeability test is cheap using DBMs
  - Partially based on a purely syntactic criterion
  - However different ways of merging sets of mergeable DBMs



  - But efficient heuristics proposed [David, 2005]

Extending Merging to Parametric State Automata

- Problems when extending the principle of merging to parametric timed automata
  1. Constraints are now constraints over the clocks and the parameters
  2. What properties are preserved when merging states?
  3. Implementation issues
     - No structure equivalent to DBMs exists for parametric timed automata (parametric DBMs [Hune et al., 2002] are not as efficient as DBMs)
       $\rightsquigarrow$ No syntactic criterion

Outline

Principle

We use the same definition as for timed automata

Definition

Two states $(\bullet, C_1)$ and $(\bullet, C_2)$ are mergeable if

1. Their discrete part is equal ($\bullet = \bullet$), and
2. Their union $C_1 \cup C_2$ is convex.

Their merging is $(\bullet, C_1 \cup C_2)$.

C_1 and C_2 are now constraints over the clocks and the parameters

When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$

When to Merge States?

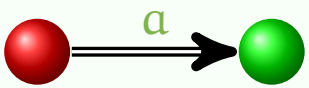
- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two

When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example

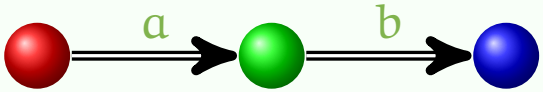
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



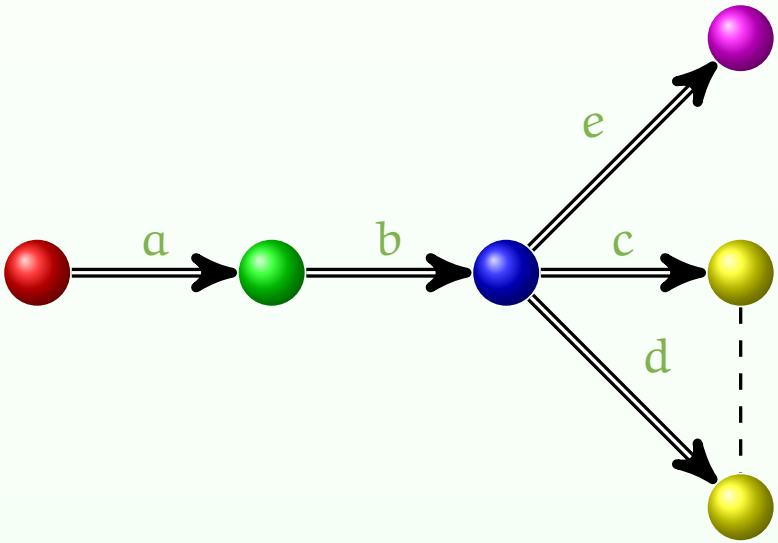
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



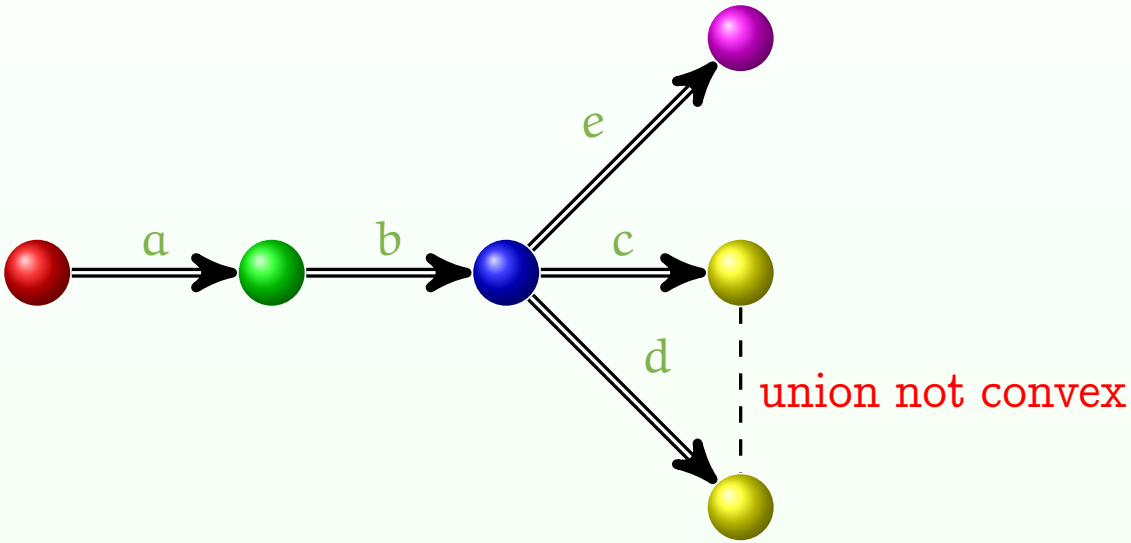
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



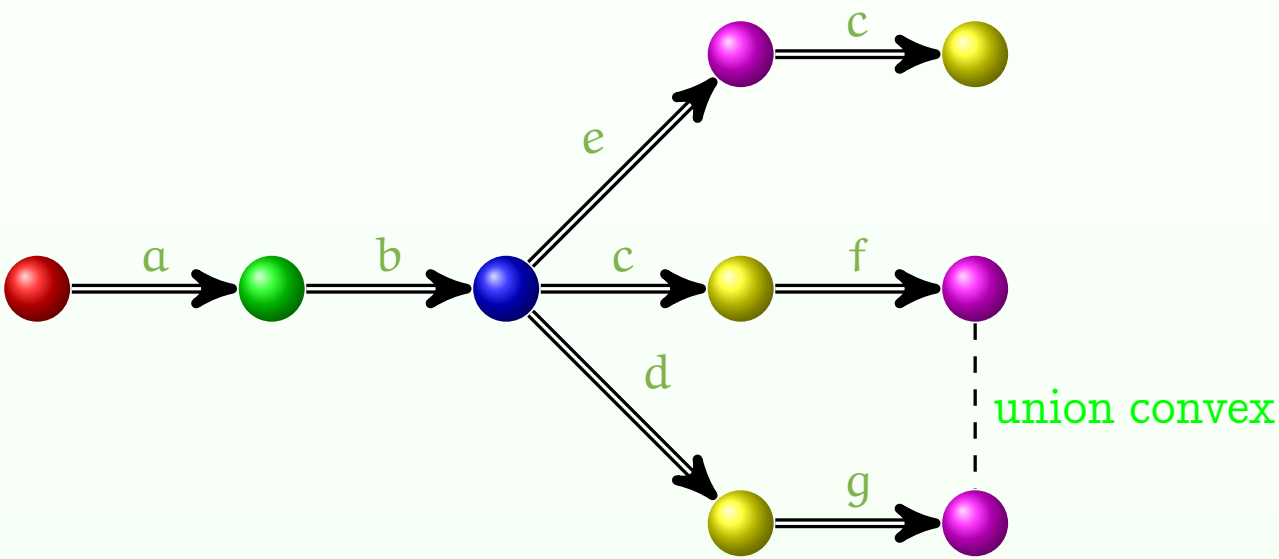
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



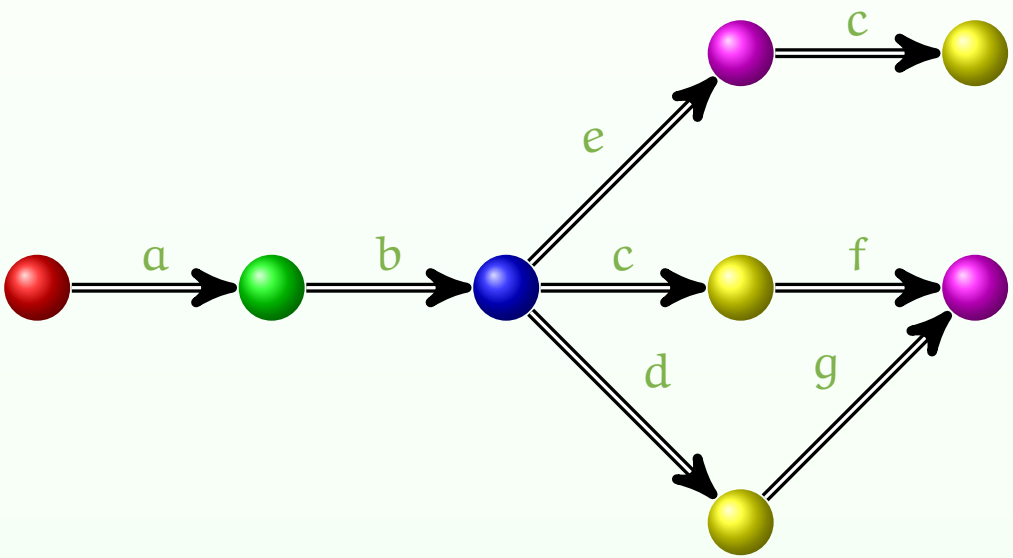
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



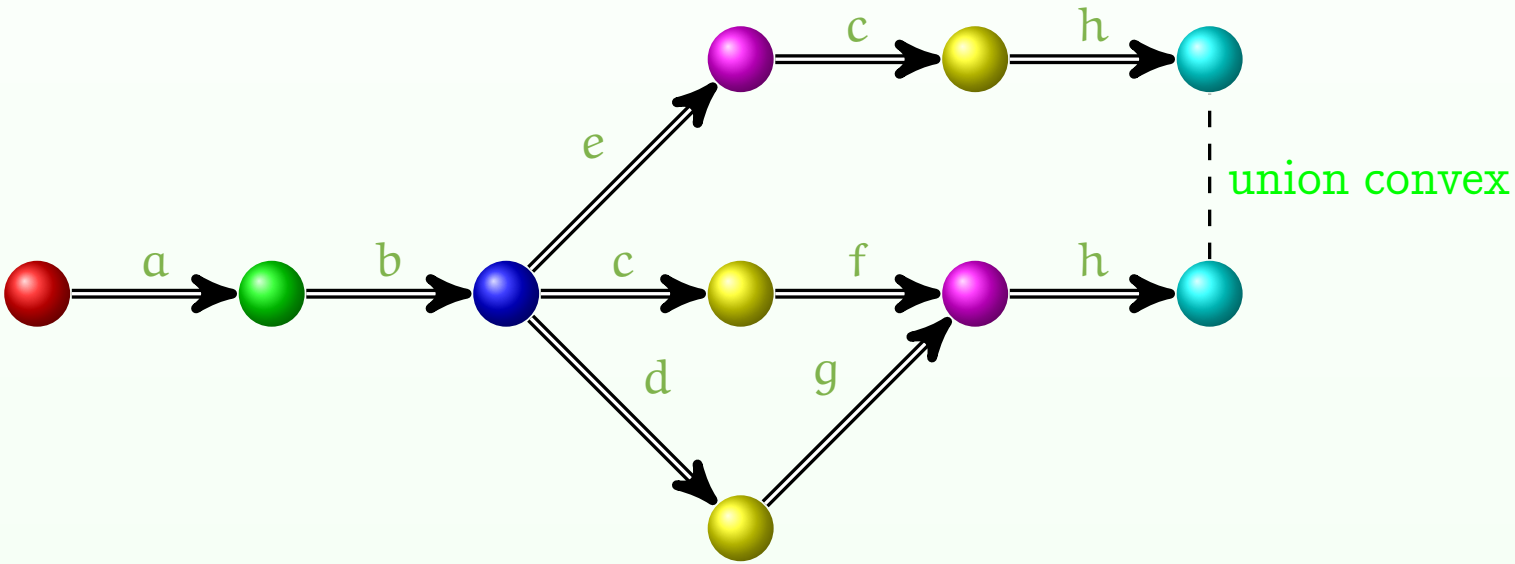
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



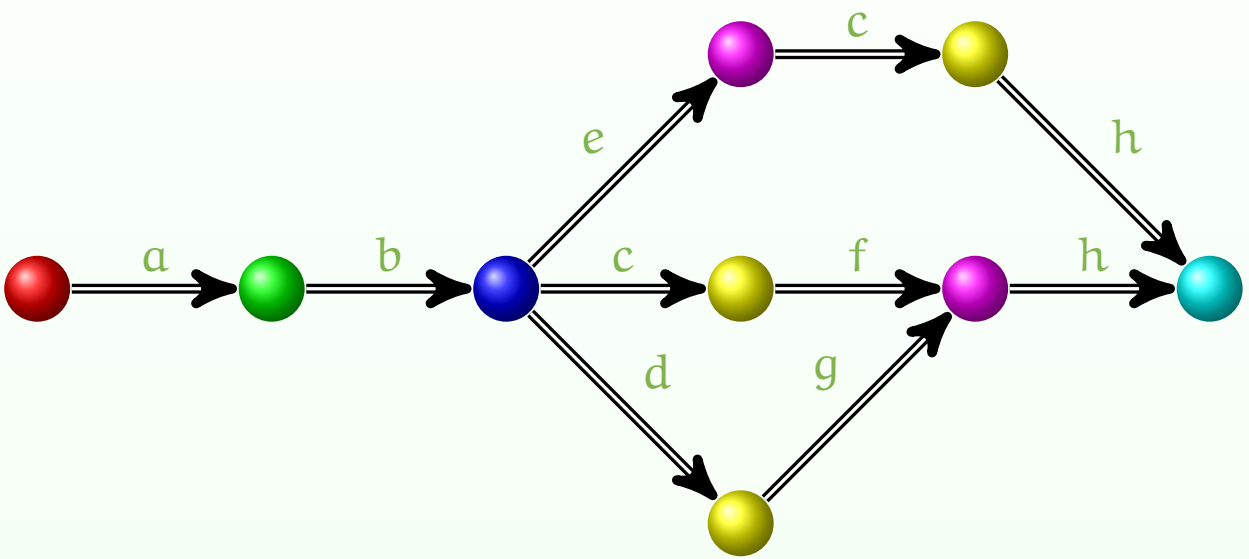
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



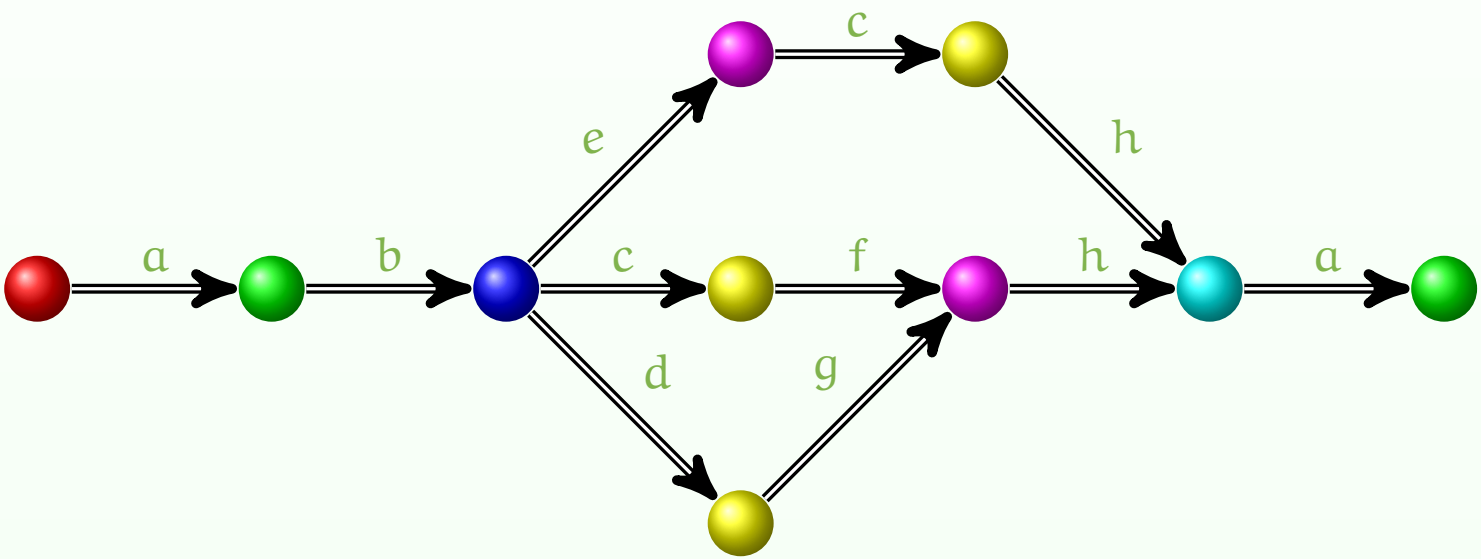
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example



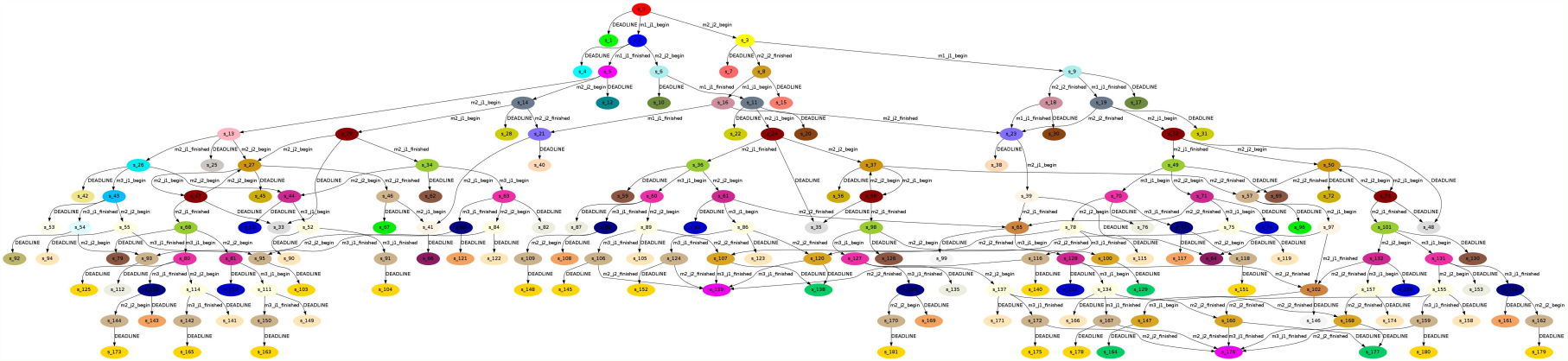
When to Merge States?

- Breadth-first analysis
  - At each level n, compute the successor states of level $n+1$
  - and merge the mergeable states of level $n+1$ two by two
- Example

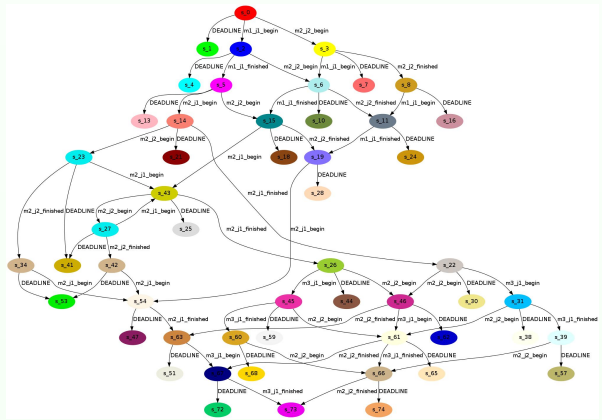


A Graphical Example

- The trace set of an example PTA



- The trace set obtained from the same PTA when merging states at each level



Case study "LA02" (trace sets generated by IMITATOR)

Properties

Notations:

- $Paths(\mathcal{A})$: paths of $\mathcal{A}$
- $Paths_{Mrg}(\mathcal{A})$: paths of $\mathcal{A}$ obtained by merging states at each level

Theorem (Characterization of merging)

- *For all* $(\bullet, C_0) \stackrel{a_0}{\Rightarrow} \dots \stackrel{a_{n-1}}{\Rightarrow} (\bullet, C_n) \in Paths(\mathcal{A})$, *there exist* $C'_1, \dots, C'_n$ *such that:*
  1. $(\bullet, C'_0) \stackrel{a_0}{\Rightarrow}_{Mrg} \dots \stackrel{a_{n-1}}{\Rightarrow}_{Mrg} (\bullet, C'_n) \in Paths_{Mrg}(\mathcal{A})$, *and*
  2. $C_i \subseteq C'_i$, *for all* $0 \leq i \leq n$.
- *For all* $(\bullet, C) \in Reach^*_{Mrg}(\mathcal{A})$ *there exist* $m \in \mathbb{N}$ *and* $(\bullet, C_1), \dots, (\bullet, C_m) \in Reach^*(\mathcal{A})$ *such that*

$$C = \bigcup_{1 \leq i \leq m} C_i.$$

Properties: Interpretation

1. The traces of $\mathcal{A}$ are included in the traces of $\mathcal{A}$ with merging
   - Hence the set of traces is over-approximated when merging states
   - The converse is not true in general
2. Each time-abstract transition  in the traces of $\mathcal{A}$ exists in the traces of $\mathcal{A}$ with merging, and conversely
   - Cannot be generalized to full traces!
3. The set of parameter valuations allowing to reach a location  in $Paths(\mathcal{A})$ is the same as the set of parameter valuations allowing to reach  in $Paths_{Mrg}(\mathcal{A})$

Properties: Interpretation

1. The traces of $\mathcal{A}$ are included in the traces of $\mathcal{A}$ with merging
   - Hence the set of traces is over-approximated when merging states
   - The converse is not true in general
2. Each time-abstract transition $\stackrel{a}{\Rightarrow}$  in the traces of $\mathcal{A}$ exists in the traces of $\mathcal{A}$ with merging, and conversely
   - Cannot be generalized to full traces!
3. The set of parameter valuations allowing to reach a location  in $Paths(\mathcal{A})$ is the same as the set of parameter valuations allowing to reach  in $Paths_{Mrg}(\mathcal{A})$

Consequence

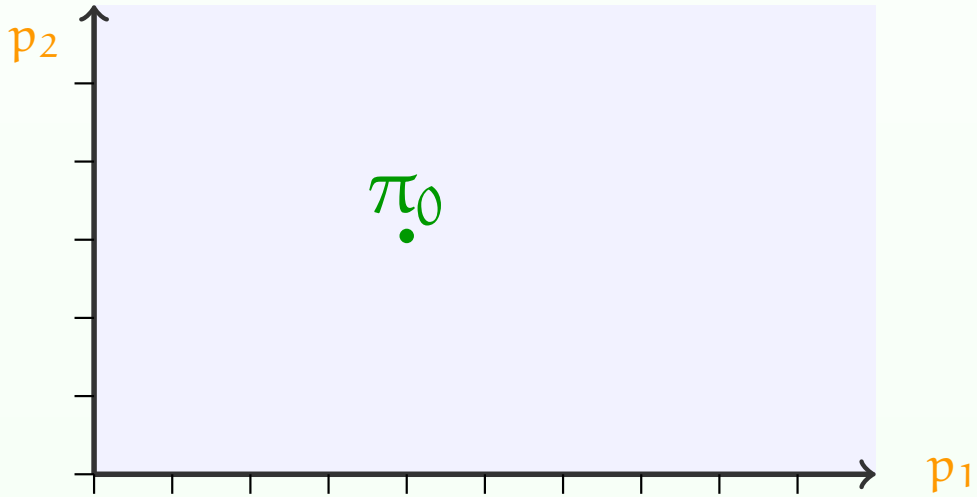
Merging states is safe for safety but not for linear-time properties.

The Inverse Method *IM*

- Algorithm for the synthesis of parameters in PTA

  [A., Chatain, Encrenaz, Fribourg, 2009]

- Takes as input a reference parameter valuation π_0, and outputs a constraint K generalizing this reference valuation



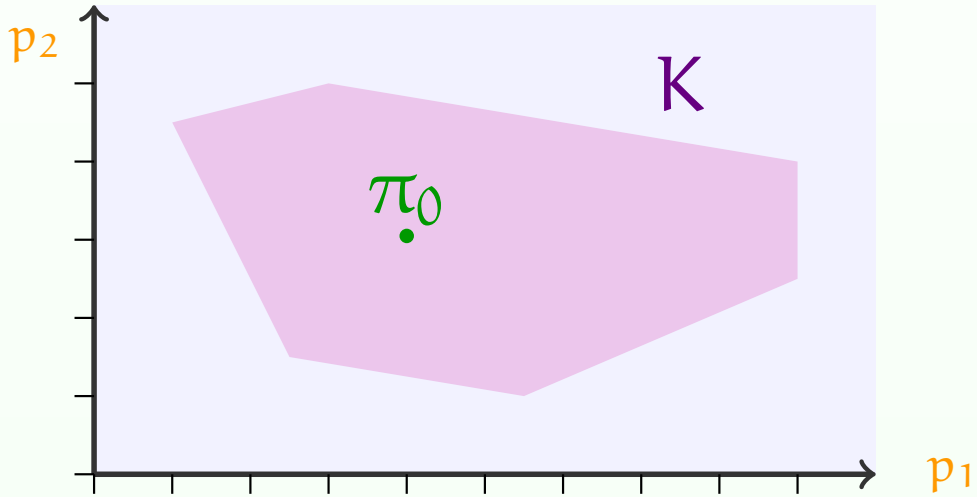
  - For all $\pi \models K$, the trace set is the same as for π_0

The Inverse Method *IM*

- Algorithm for the synthesis of parameters in PTA

  [A., Chatain, Encrenaz, Fribourg, 2009]

- Takes as input a reference parameter valuation π_0, and outputs a constraint K generalizing this reference valuation



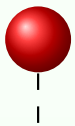
  - For all $\pi \models K$, the trace set is the same as for π_0

The Inverse Method: Example

π_0 :

$$\delta_1^- = 7 \quad \delta_1^+ = 7 \quad T_{HI} = 24$$
$$\delta_2^- = 5 \quad \delta_2^+ = 6 \quad T_{LO} = 15$$
$$\delta_3^- = 8 \quad \delta_3^+ = 10 \quad T_{Setup} = 10$$
$$\delta_4^- = 3 \quad \delta_4^+ = 7 \quad T_{Hold} = 17$$

$K = \texttt{true}$



$T_{Setup} \leq T_{LO}$

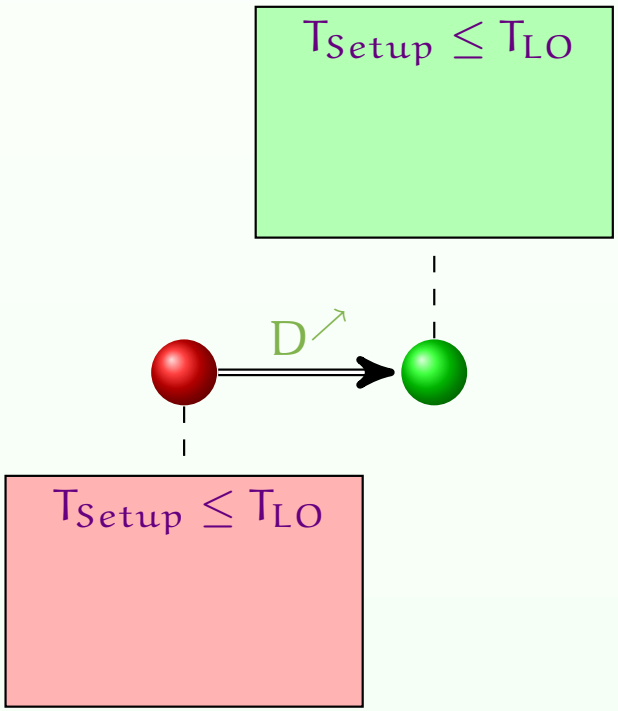
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K = \texttt{true}$



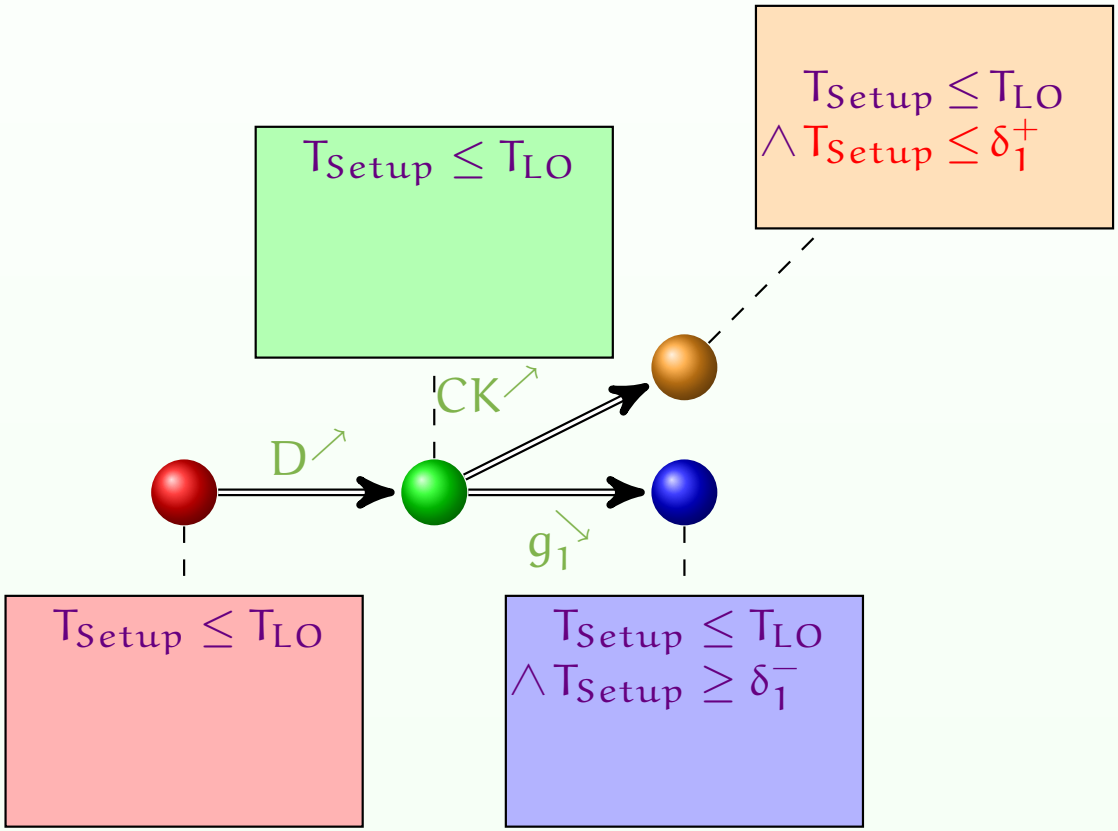
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K = \texttt{true}$



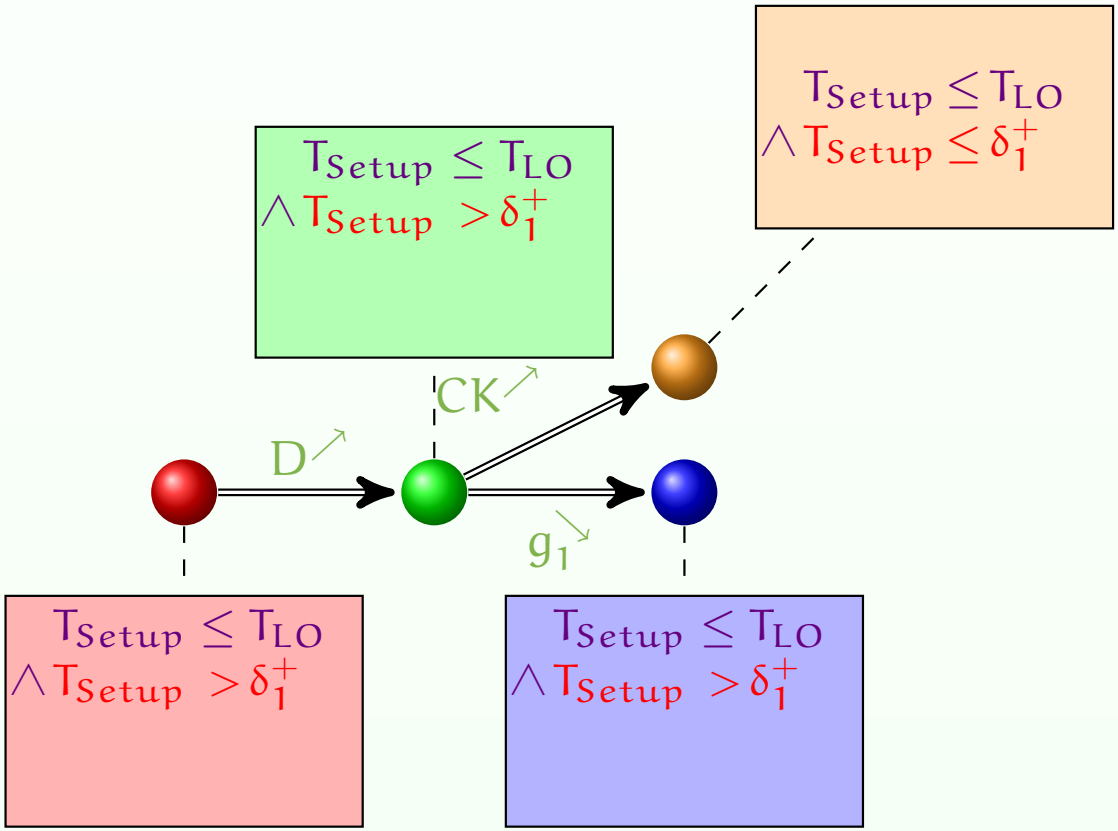
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K = T_{Setup} > \delta_1^+$



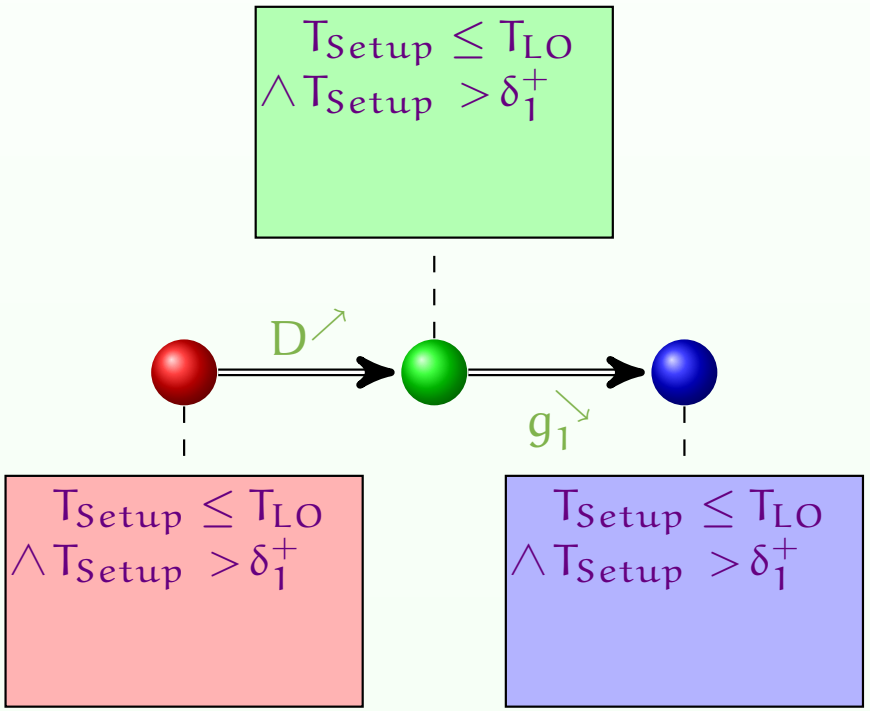
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K = T_{Setup} > \delta_1^+$



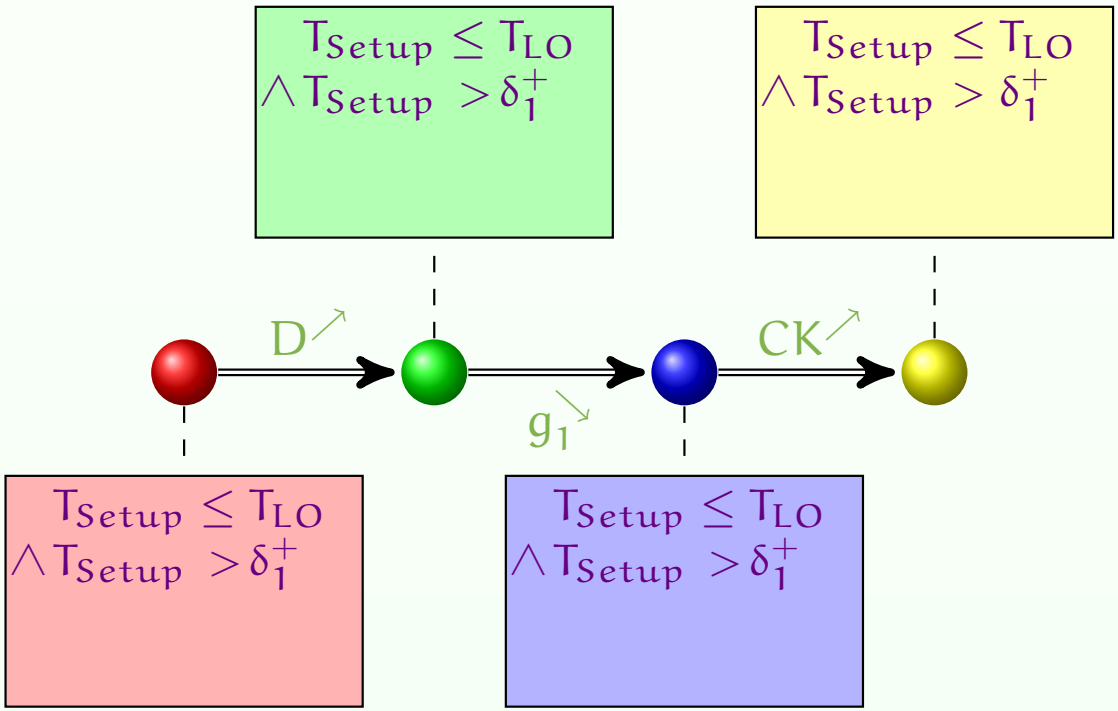
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K = T_{Setup} > \delta_1^+$



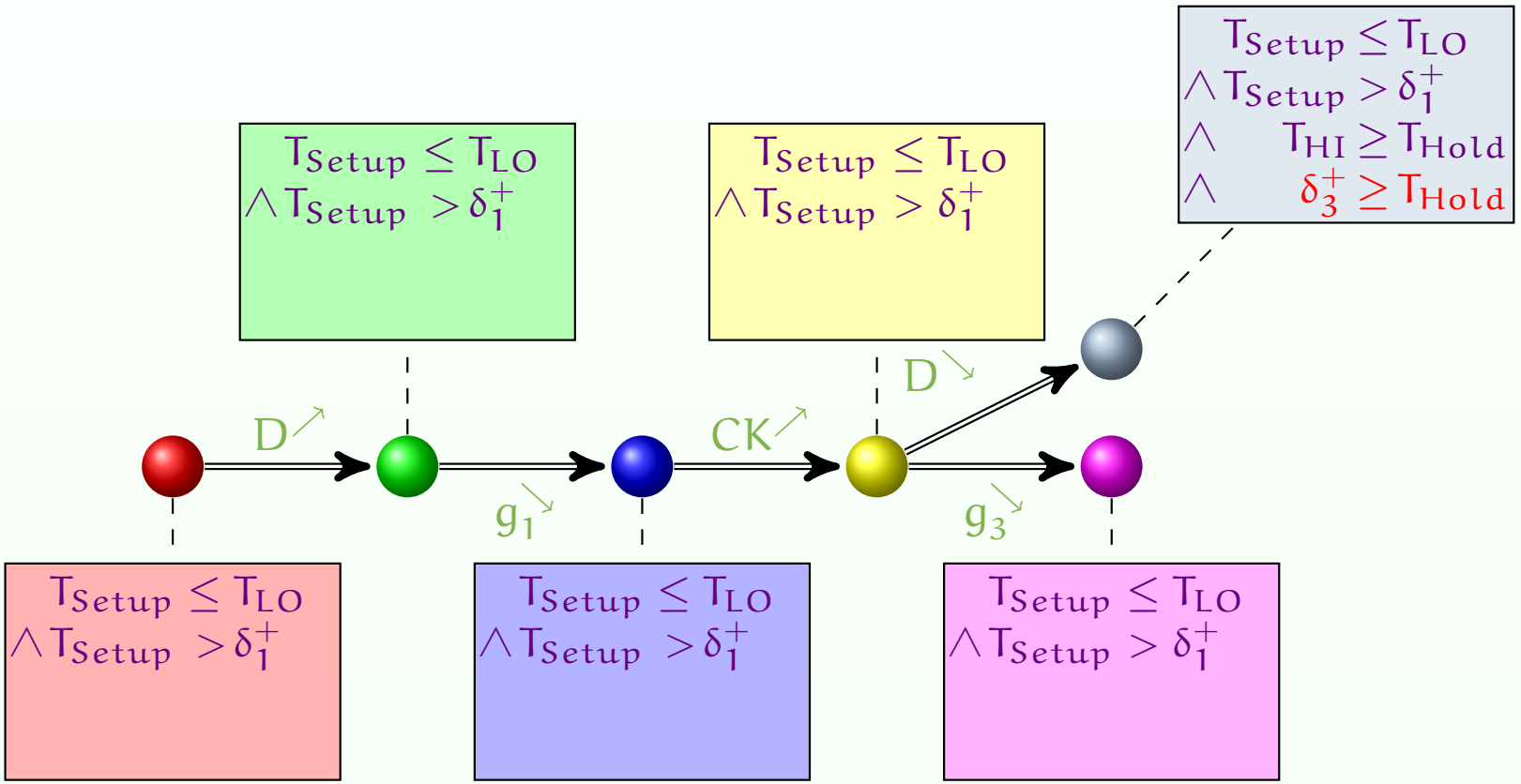
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K = T_{Setup} > \delta_1^+$



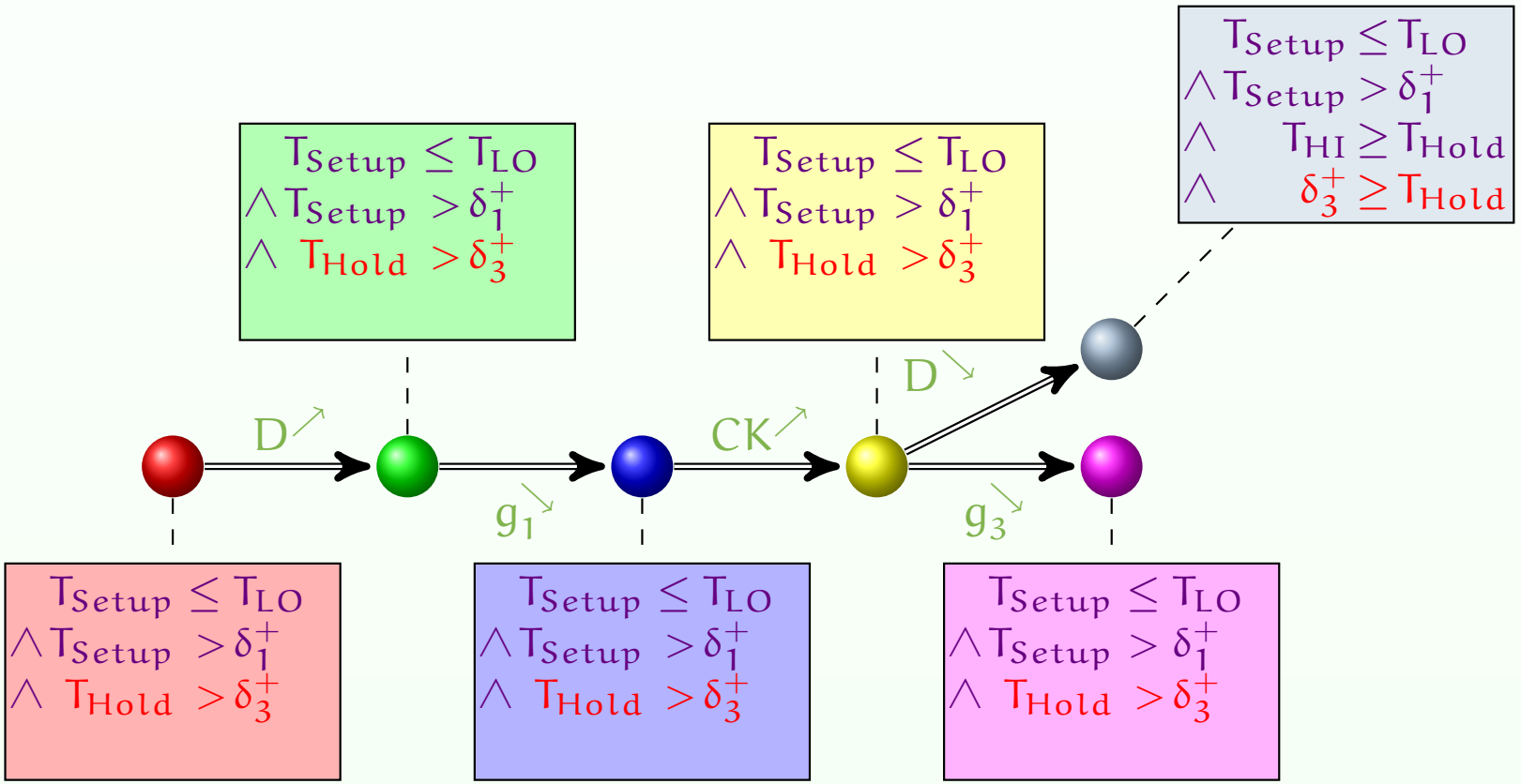
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K = T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$



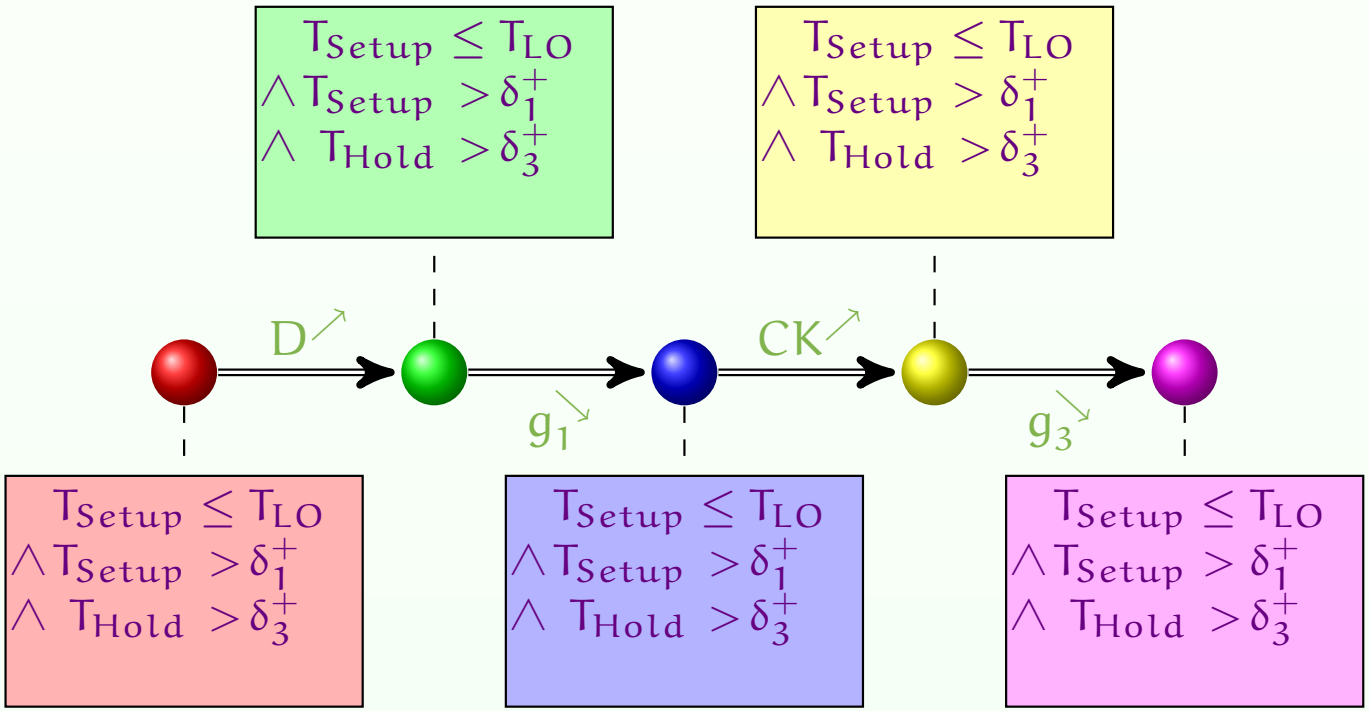
Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K =$
$T_{Setup} > \delta_1^+$
$\wedge\ T_{Hold} > \delta_3^+$



Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method: Example

π_0 :

$\delta_1^- = 7$ $\quad \delta_1^+ = 7$ $\quad T_{HI} = 24$

$\delta_2^- = 5$ $\quad \delta_2^+ = 6$ $\quad T_{LO} = 15$

$\delta_3^- = 8$ $\quad \delta_3^+ = 10$ $\quad T_{Setup} = 10$

$\delta_4^- = 3$ $\quad \delta_4^+ = 7$ $\quad T_{Hold} = 17$

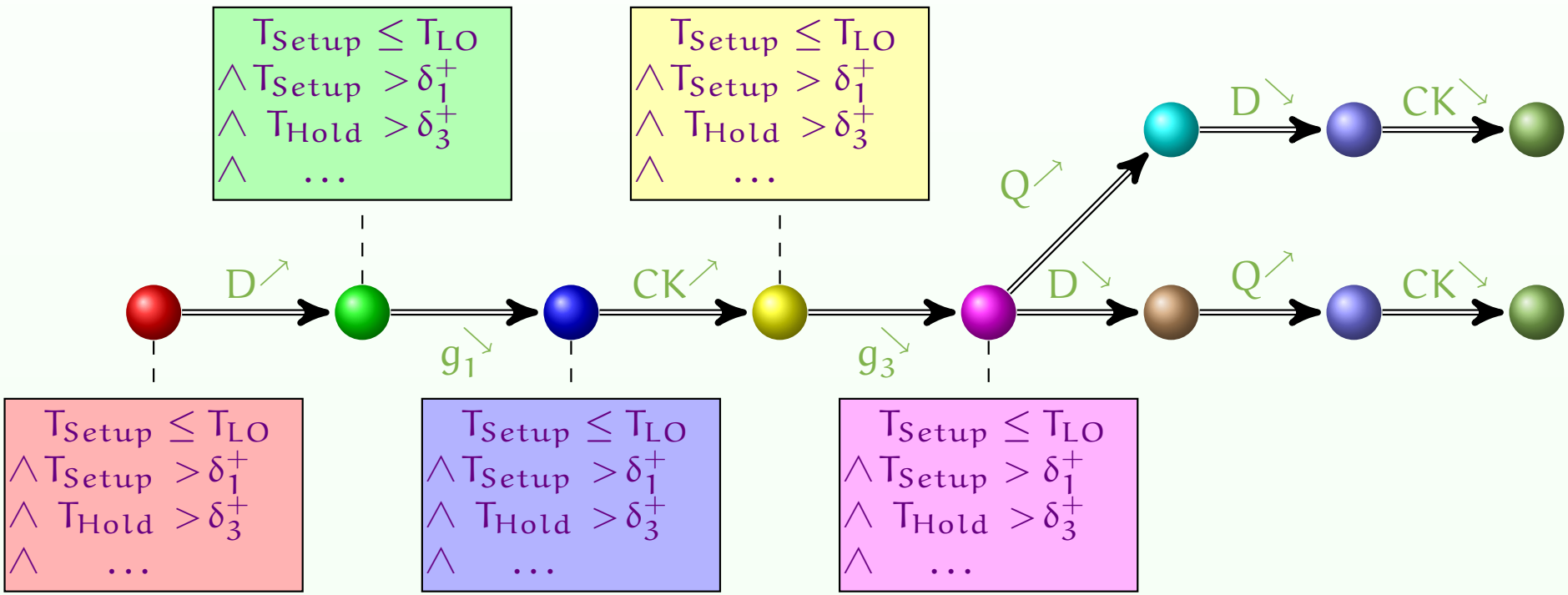
$K =$

$T_{Setup} > \delta_1^+ \wedge \delta_3^+ + \delta_4^+ \geq T_{Hold}$

$\wedge\ T_{Hold} > \delta_3^+ \wedge \delta_3^+ + \delta_4^+ < T_{HI}$

$\wedge\ T_{Setup} \leq T_{LO} \wedge \delta_3^- + \delta_4^- \leq T_{Hold}$

$\wedge\ \delta_1^- > 0$



Example of a flip-flop circuit [Clarisó and Cortadella, 2007]

The Inverse Method

Starting from the initial state with $K = \texttt{true}$

While there are successor states **do**

1. Compute successor states (breadth-first)

3. Remove π_0-incompatible states $(\bullet, C)$ (i.e., such that $\pi_0 \not\models C$) by refining K
4. Go to 1

Return the intersection of all constraints

The Inverse Method With Merging IM_{Mrg}

Starting from the initial state with $K = \texttt{true}$

While there are successor states **do**

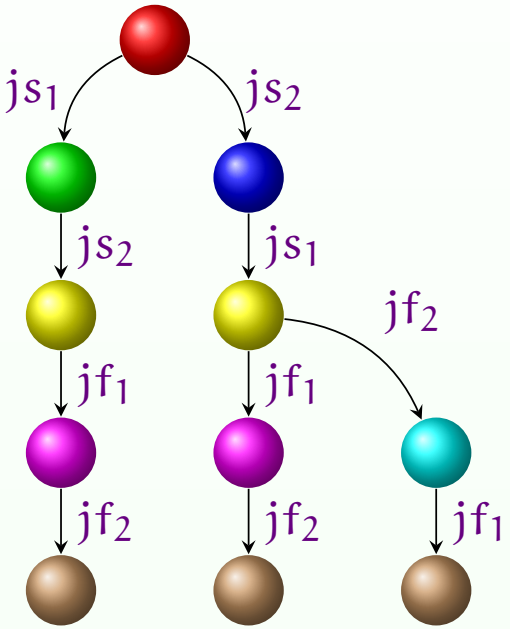
1. Compute successor states (breadth-first)
2. Merge these successor states
3. Remove π_0-incompatible states $(\bullet, C)$ (i.e., such that $\pi_0 \not\models C$) by refining K
4. Go to 1

Return the intersection of all constraints

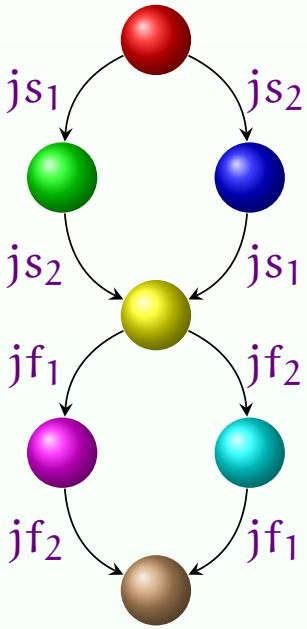
IM_{Mrg}: The Problem

- Example of a jobshop case study

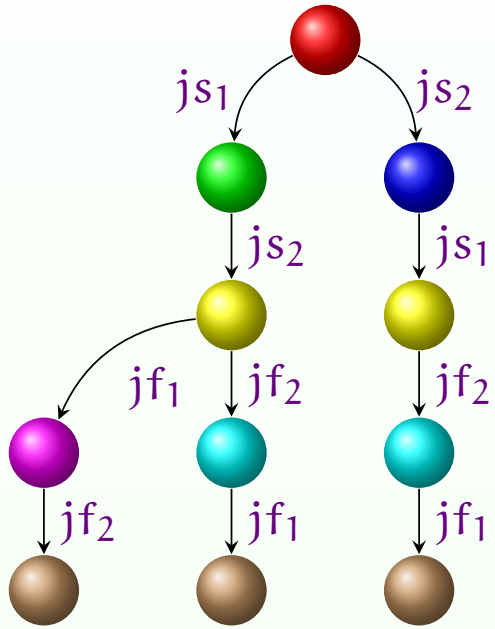
For a given π_0



For $IM_{Mrg}(\mathcal{A}, \pi_0)$



For some $\pi \models IM_{Mrg}(\mathcal{A}, \pi_0)$



- Problems
  - Trace sets are not equal
  - Equality of locations but no equality of actions in general

IM_{Mrg}: Properties

Theorem

Suppose $IM_{Mrg}(\mathcal{A}, \pi_0)$ *terminates with output* K_{Mrg}. *Then*

1. $\pi_0 \models \mathsf{K}_{Mrg}$, *and*
2. *For all* $\pi \models \mathsf{K}_{Mrg}$, *the reachable locations are the same for* π_0 *and* π.

IM_{Mrg}: Properties

Theorem

Suppose $IM_{Mrg}(\mathcal{A}, \pi_0)$ terminates with output K_{Mrg}. Then

1. $\pi_0 \models \mathsf{K}_{Mrg}$, *and*
2. *For all $\pi \models \mathsf{K}_{Mrg}$, the reachable locations are the same for π_0 and π.*

For backward-deterministic PTA, equality of the actions set is guaranteed; but not for general PTA.

Inverse Method With Merging

Starting from the initial state with $K = \texttt{true}$

While there are successor states **do**

1. Compute successor states (breadth-first)
2. Merge these successor states
3. Remove π_0-incompatible states $(\bullet, C)$ (i.e., such that $\pi_0 \not\models C$) by refining K
4. Go to 1

Return the intersection of all constraints

An Improved Inverse Method With Merging IM'_{Mrg}

Starting from the initial state with $K = \texttt{true}$

While there are successor states **do**

1. Compute successor states (breadth-first)
2. Remove π_0-incompatible states $(\bullet, C)$ (i.e., such that $\pi_0 \not\models C$) by refining K
3. Merge these successor states
4. Go to 1

Return the intersection of all constraints

IM'_{Mrg}: Properties

Theorem

Suppose $IM'_{Mrg}(\mathcal{A}, \pi_0)$ *terminates with output* K'_{Mrg}. *Then*

1. $\pi_0 \models \mathsf{K}'_{Mrg}$, *and*
2. *For all* $\pi \models \mathsf{K}'_{Mrg}$, *the reachable locations and executable actions are the same for* π_0 *and* π.

Remarks

- Equality of trace sets is still not guaranteed
- We have $\mathsf{K} \subseteq \mathsf{K}'_{Mrg} \subseteq \mathsf{K}_{Mrg}$
  - IM'_{Mrg} can be seen as a tradeoff between IM without merging (the state space of which may blow up), and IM_{Mrg} (that does not preserve actions)

Outline

Implementation in IMITATOR

- IMITATOR 2.6 [A., Fribourg, Kühne, Soulat, 2012]
  - "Inverse Method for Inferring Time AbstracT BehaviOR"
  - 10,000 lines of OCaml code
  - Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
  - Available under the GNU-GPL license
  - Now integrated in the *CosyVerif* platform [AHHKLLP13]

- Experimental validation by comparing performances when executing IM and IM'_{Mrg}

Experiments

Example	|X|	|P|	IM States	IM Trans.	IM t	IM Cpl	IM'_{Mrg} States	IM'_{Mrg} Trans.	IM'_{Mrg} t	IM'_{Mrg} Cpl	Comparison States	Comparison t	Comparison K
AndOr	4	12	11	11	0.052	√	9	9	0.056	√	82	108	=
Flip-Flop	5	12	11	10	0.060	√	9	9	0.057	√	82	108	=
Latch	8	13	18	17	0.083	?	12	12	0.069	?	67	83	=
SPSMALL	10	26	31	30	0.618	?	31	30	0.613	?	100	99	=
SIMOP	8	7	-	-	loop	-	172	262	2.52	?	0	0	-
BRP	7	6	429	474	3.50	√	426	473	4.30	√	99	123	=
CSMA/CD	3	3	301	462	0.514	√	300	461	0.574	√	100	112	=
CSMA/CD'	3	3	13,365	14,271	18.3	√	13,365	14,271	25.4	√	100	139	=
RCP	5	6	327	518	0.748	√	115	186	0.684	√	35	91	=
WLAN	2	8	-	-	loop	-	8,430	15,152	2,137	√	0	0	-
ABT	7	7	63	62	0.344	?	63	62	0.335	?	100	97	=
AM02	3	4	182	215	0.369	√	53	70	0.112	√	29	30	$\subsetneq$
BB04	6	7	806	827	28.0	?	141	145	3.15	?	17	11	=
CTC	15	21	1,364	1,363	88.9	√	215	264	17.6	√	16	20	=
LA02	3	5	6,290	8,023	751	?	383	533	17.7	√	6.0	2.4	$\subsetneq$
LPPRC10	4	7	78	102	0.39	?	31	40	0.251	?	40	64	=
M2.4	3	8	1,497	1,844	8.89	√	119	181	0.374	√	7.9	4.2	$\subsetneq$

Sources: http://www.lsv.ens-cachan.fr/Software/imitator/merging/

Experiments: Remarks

- From our results, IM'_{Mrg} always has a number of states equal to or less than IM
  - Strictly less states in all but 4 experiments
  - Very efficient for scheduling problems: up to a division by 16
  - Some case studies can only be verified using IM'_{Mrg}
- Time reduction
  - When no states are merged, up to 39 % extra time
    - Mergeability test expensive using polyhedra
    - Reasonable?
  - More surprisingly, despite expensive overhead, IM'_{Mrg} is often faster than IM
    - Up to a division by 42
- Set of parameter valuations output by IM'_{Mrg} often larger than IM

Outline

Conclusion

- Characterization of states merging for parametric timed automata
  - Safety properties preserved, but not linear-time properties
- Characterization of states merging in the inverse method
  - No preservation of the trace set
  - Locations preserved, but not actions in general
  - Improvement to preserve actions too
- Experimental validation
  - Improvement of both memory and time for many case studies
  - Reasonable time overhead in worst case studies
  - Larger set of parameter valuations synthesized

Perspectives

- Improve the heuristics
  - When to merge?
    - May change the properties of merging
  - On which states should the mergeability test be applied?
    - Expensive test to be performed only when its has good chances to be positive
- Combine with other state space reduction techniques
  - In particular (quasi-)equal clock elimination
- Extend to larger classes of models such as hybrid systems

  [Alur et al., 1993a, Fribourg and Kühne, 2013]

Bibliography

References I

Alur, R., Courcoubetis, C., Henzinger, T. A., and Ho, P.-H. (1993a).
Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems.
In Grossman, R. L., Nerode, A., Ravn, A. P., and Rischel, H., editors, *Hybrid Systems 1992*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer.

Alur, R. and Dill, D. L. (1994).
A theory of timed automata.
Theoretical Computer Science, 126(2):183–235.

Alur, R., Henzinger, T. A., and Vardi, M. Y. (1993b).
Parametric real-time reasoning.
In *STOC*, pages 592–601. ACM.

André, É., Chatain, Th., Encrenaz, E., and Fribourg, L. (2009).
An inverse method for parametric timed automata.
International Journal of Foundations of Computer Science, 20(5):819–836.

André, É., Fribourg, L., Kühne, U., and Soulat, R. (2012).
IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.
In *FM*, volume 7436 of *LNCS*, pages 33–36. Springer.

References II

André, É., Hillah, L.-M., Hulin-Hubard, F., Kordon, F., Lembachar, Y., Linard, A., and Petrucci, L. (2013).
CosyVerif: An open source extensible verification environment.
In *ICECCS*, pages 33–36. IEEE Computer Society.

Bagnara, R., Hill, P. M., and Zaffanella, E. (2008).
The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems.
Science of Computer Programming, 72(1–2):3–21.

Clarisó, R. and Cortadella, J. (2007).
The octahedron abstract domain.
Science of Computer Programming, 64(1):115–139.

David, A. (2005).
Merging DBMs efficiently.
In *NWPT*, pages 54–56. DIKU, University of Copenhagen.

David, A. (2006).
Uppaal DBM library programmer's reference.
`http://people.cs.aau.dk/~adavid/UDBM/manual-061023.pdf`.

References III

Fribourg, L. and Kühne, U. (2013).
Parametric verification and test coverage for hybrid automata using the inverse method.
International Journal of Foundations of Computer Science, 24(2):233–249.

Hune, T., Romijn, J., Stoelinga, M., and Vaandrager, F. W. (2002).
Linear parametric model checking of timed automata.
Journal of Logic and Algebraic Programming, 52-53:183–220.

Additional explanation

The Inverse Method: Algorithm

Algorithm 1: $IM(\mathcal{A}, \pi)$

input : PTA $\mathcal{A}$ of initial state s_0, parameter valuation π
output: Constraint K over the parameters

1 $i \leftarrow 0$; $K_c \leftarrow \texttt{true}$; $S_{new} \leftarrow \{s_0\}$; $S \leftarrow \{\}$
2 **while** `true` **do**
3 **while** *there are* π*-incompatible states in* S_{new} **do**
4 Select a π-incompatible state (l, C) of S_{new} (i.e., s.t. $\pi \not\models C$) ;
5 Select a π-incompatible J in $C{\downarrow_P}$ (i.e., s.t. $\pi \not\models J$) ;
6 $K_c \leftarrow K_c \wedge \neg J$; $S \leftarrow \bigcup_{j=0}^{i-1} Post^j_{\mathcal{A}(K_c)}(\{s_0\})$; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$;
7 **if** $S_{new} \sqsubseteq S$ **then return** $K \leftarrow \bigcap_{(l,C)\in S} C{\downarrow_P}$
8 $i \leftarrow i+1$; $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow Post_{\mathcal{A}(K_c)}(S)$

Licensing

Source of the graphics used

Title: Smiley green alien big eyes (aaah)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg
License: public domain

License of this document

Author: Étienne André