UNIVERSITÉ PARIS 13

Habilitation defense

Monday, 25th of June 2018

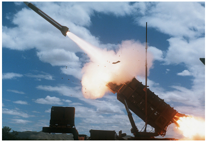
Contributions to parametric timed model checking: Theory and algorithms

Étienne André

LIPN, Université Paris 13, CNRS, France

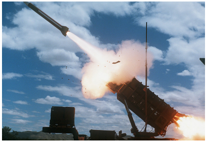
Context: Critical real-time systems

- Real-time systems are everywhere
  - Hard timing constraints and concurrency
  - Criticality: risk for huge damages in case of unexpected behavior (bug)


Context: Critical real-time systems

- Real-time systems are everywhere
  - Hard timing constraints and concurrency
  - Criticality: risk for huge damages in case of unexpected behavior (bug)

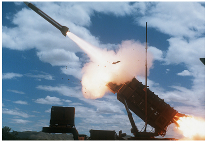




- Verification to ensure absence of bugs is required

- Common techniques
  - Testing
  - Abstract interpretation
  - Theorem proving
  - Model checking

Context: Critical real-time systems

- Real-time systems are everywhere
  - Hard timing constraints and concurrency
  - Criticality: risk for huge damages in case of unexpected behavior (bug)

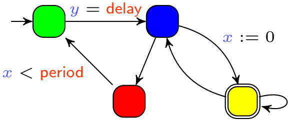




- Verification to ensure absence of bugs is required

- Common techniques
  - Testing
  - Abstract interpretation
  - Theorem proving
  - Model checking

Model checking timed concurrent systems

- Use formal methods [Baier and Katoen, 2008]



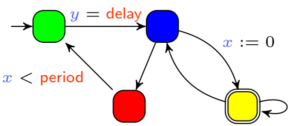
A model of the system


is unreachable

A property to be satisfied

Model checking timed concurrent systems

- Use formal methods [Baier and Katoen, 2008]



A model of the system

$\stackrel{?}{\models}$

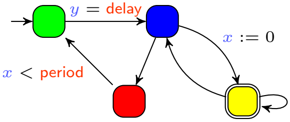
 is unreachable

A property to be satisfied

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

Model checking timed concurrent systems

- Use formal methods [Baier and Katoen, 2008]



A model of the system

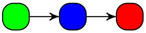
$\overset{?}{\models}$

A property to be satisfied

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

Yes

No

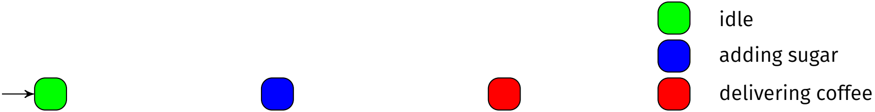


Counterexample

Turing award (2007) to Edmund M. Clarke, Allen Emerson and Joseph Sifakis

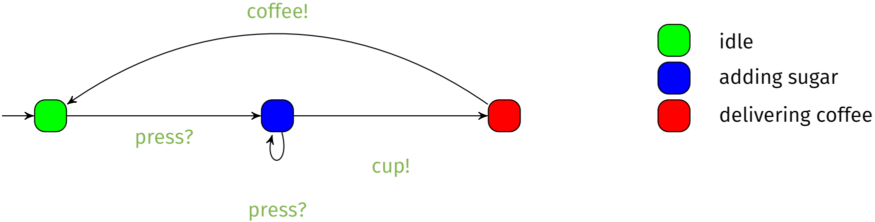
Timed automaton (TA)

- Finite state automaton (sets of locations)



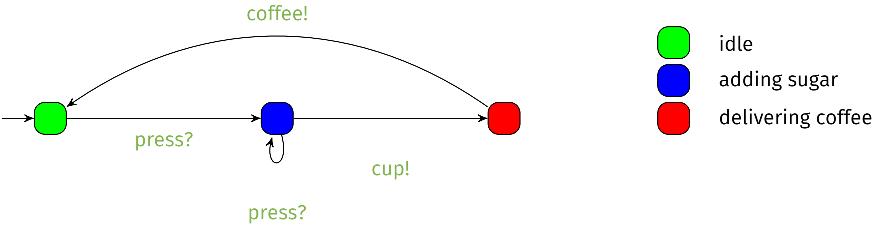
Timed automaton (TA)

- Finite state automaton (sets of locations and actions)



Timed automaton (TA)

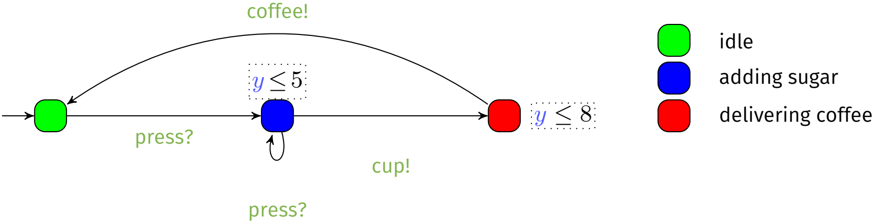
- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate



Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants

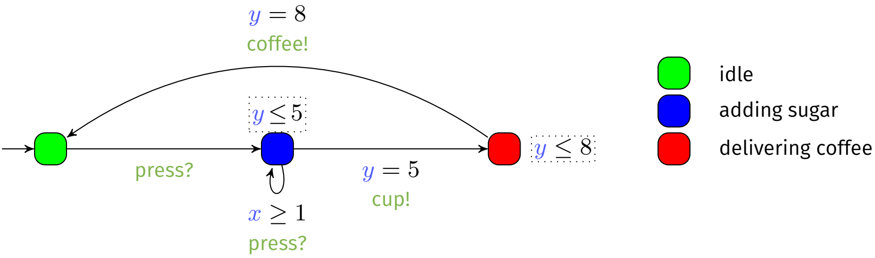
- Features
  - Location invariant: property to be verified to stay at a location



Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants and guards

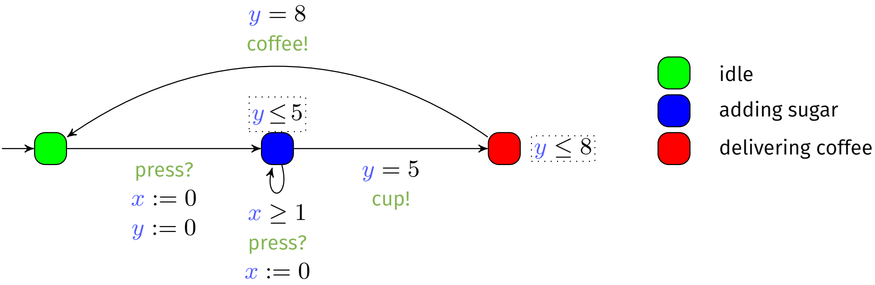
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition



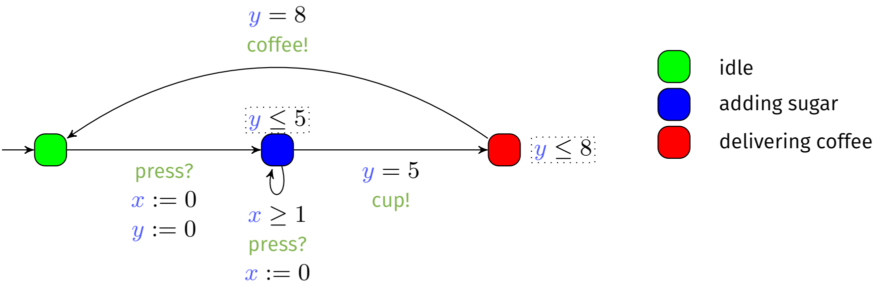
Timed automaton (TA)

- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants and guards

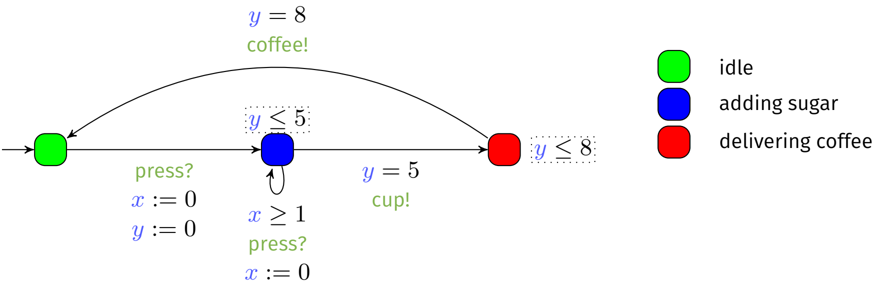
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 along transitions



The most critical system: The coffee machine



The most critical system: The coffee machine

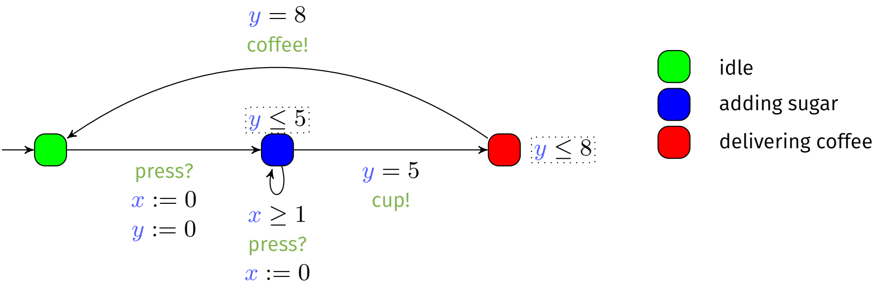


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

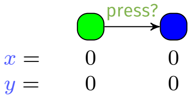
$x = 0$

$y = 0$

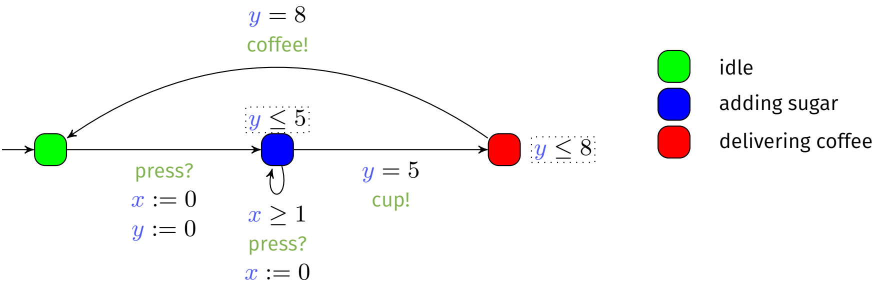
The most critical system: The coffee machine



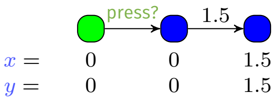
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



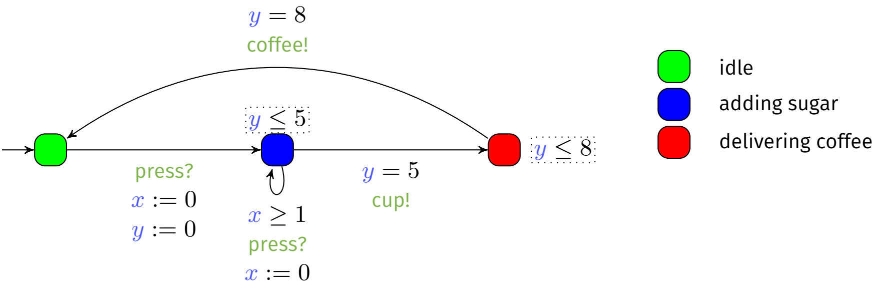
The most critical system: The coffee machine



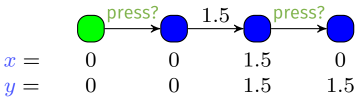
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



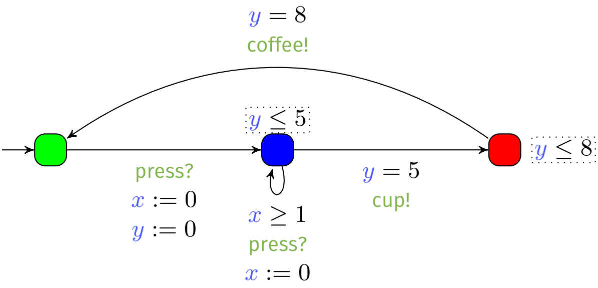
The most critical system: The coffee machine

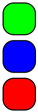


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

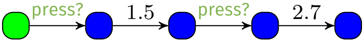


The most critical system: The coffee machine



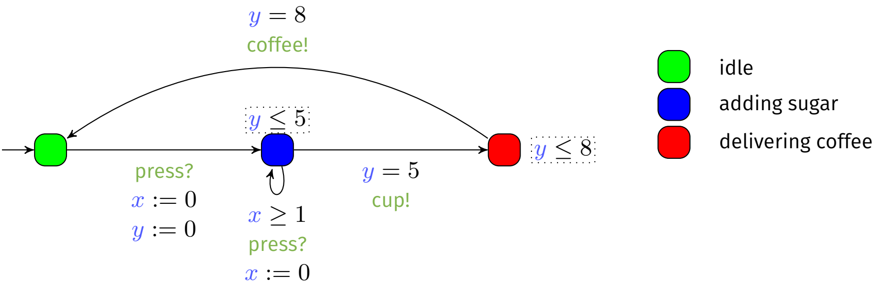


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

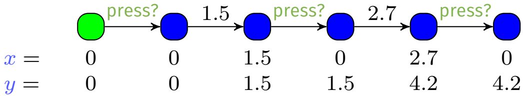


$x =$	0	0	1.5	0	2.7
$y =$	0	0	1.5	1.5	4.2

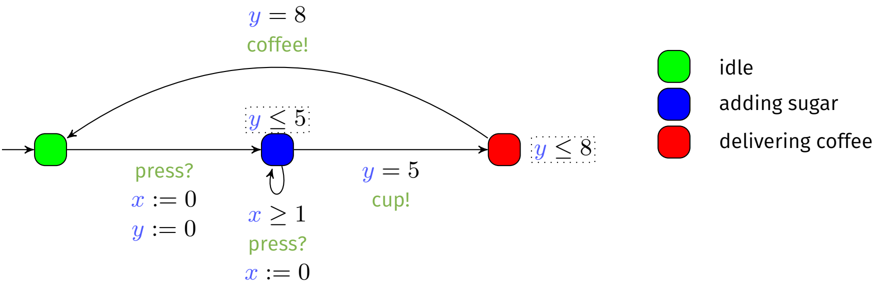
The most critical system: The coffee machine



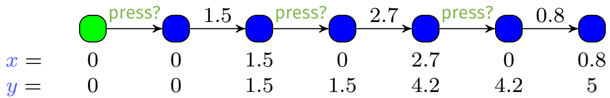
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



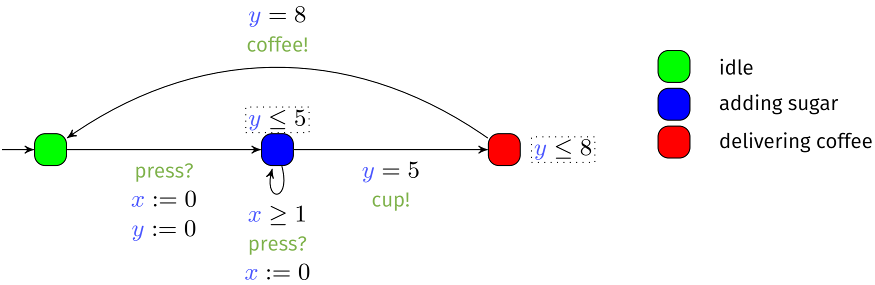
The most critical system: The coffee machine



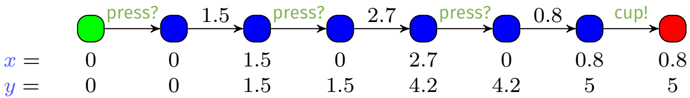
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



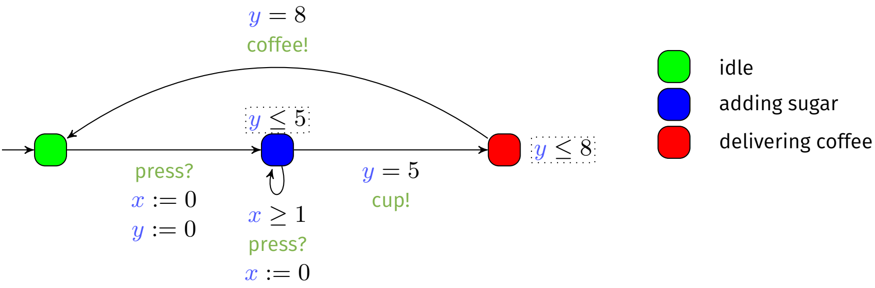
The most critical system: The coffee machine



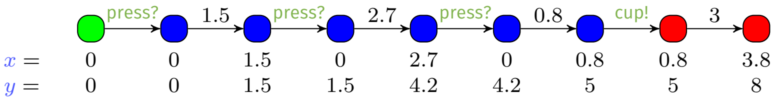
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



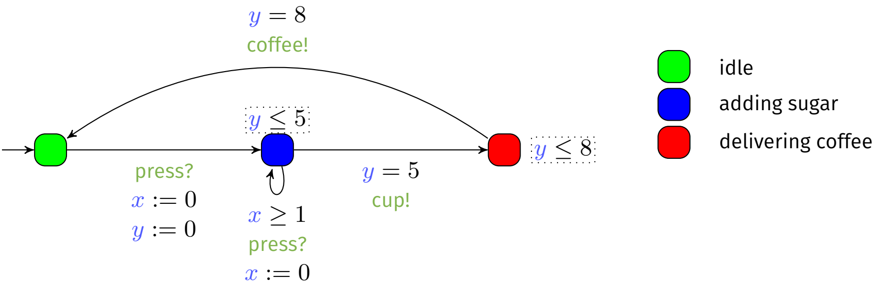
The most critical system: The coffee machine



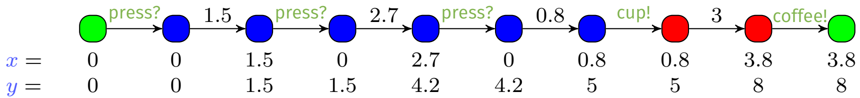
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



The most critical system: The coffee machine



- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



Timed automata: A success story

- An expressive formalism
  - Dense time
  - Concurrency
- A tractable verification in theory
  - Reachability is PSPACE-complete [Alur and Dill, 1994]
- A very efficient verification in practice
  - Symbolic verification: relatively insensitive to constants
  - Several model checkers, notably UPPAAL [Larsen et al., 1997]
  - Long list of successful case studies

Need to allow for abstractions and uncertainty

- Need for abstraction
  - Constants known with limited certainty
  - Unknown constants

Need to allow for abstractions and uncertainty

- Need for abstraction
  - Constants known with limited certainty
  - Unknown constants

- Idea: reason with parameters (unknown constants)
  - Verify the system in presence of uncertain constants
  - Synthesize suitable valuations for unknown parameters
  - Optimize parameter valuations

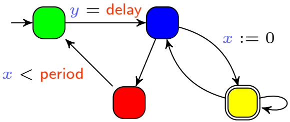
Need to allow for abstractions and uncertainty

- Need for abstraction
  - Constants known with limited certainty
  - Unknown constants

- Idea: reason with parameters (unknown constants)
  - Verify the system in presence of uncertain constants
  - Synthesize suitable valuations for unknown parameters
  - Optimize parameter valuations

- Challenging problems
  - Existence (dually: emptiness): find one valuation for which a property holds
    - “Can I exhibit a valuation for which I am guaranteed to eventually get a coffee?”
  - Synthesis: find some/all parameter valuations for which a property holds
    - “Synthesize all valuations for which I am guaranteed to eventually get a coffee with 2 sugars”

timed model checking



$\overset{?}{\models}$

 is unreachable

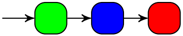
A model of the system

A property to be satisfied

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

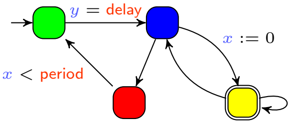
Yes

No



Counterexample

Parametric timed model checking



A model of the system

$\overset{?}{\models}$

 is unreachable

A property to be satisfied

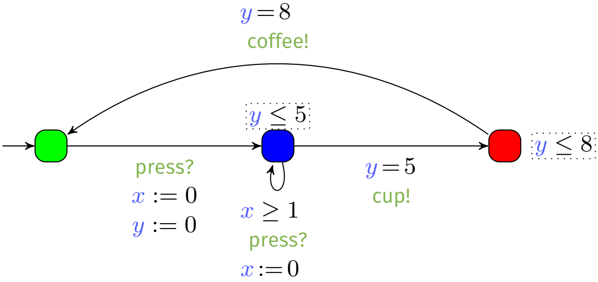
- Question: for what values of the parameters does the model of the system satisfy the property?

Yes if...

$$2 \times \text{delay} > \text{period}$$
$$\wedge\ \text{period} < 20.46$$

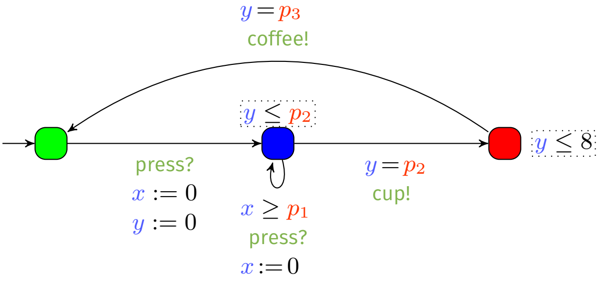
Parametric Timed Automaton (PTA)

- Timed automaton (sets of locations, actions and clocks)



Parametric Timed Automaton (PTA)

- Timed automaton (sets of locations, actions and clocks) augmented with a set P of parameters [Alur et al., 1993]
  - Unknown constants compared to a clock in guards and invariants

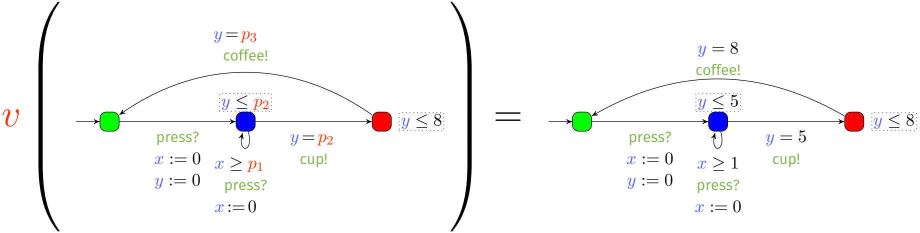


Notation: Valuation of a PTA

- Given a PTA $\mathcal{A}$ and a parameter valuation v, we denote by $v(\mathcal{A})$ the (non-parametric) timed automaton where each parameter p is valuated by $v(p)$

Notation: Valuation of a PTA

- Given a PTA $\mathcal{A}$ and a parameter valuation v, we denote by $v(\mathcal{A})$ the (non-parametric) timed automaton where each parameter p is valuated by $v(p)$



$$\text{with } v : \begin{cases} p_1 & \rightarrow & 1 \\ p_2 & \rightarrow & 5 \\ p_3 & \rightarrow & 8 \end{cases}$$

Objectives

Main objective

Perform efficient parameter synthesis for parametric timed automata

Objectives

Main objective

Perform efficient parameter synthesis for parametric timed automata

Before designing algorithms, one shall first study theory

- Decidability
- Complexity
- Syntactic restrictions

Outline

Outline

1 Decidability

2 Efficient synthesis

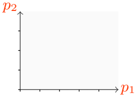
3 Applications to schedulability analysis

4 Perspectives

25 years of (un)decidability results for PTAs

Key problem considered: EF-emptiness

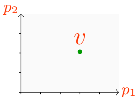
- "given a PTA $\mathcal{A}$ and a location 🟥, is the set of parameter valuations v such that $v(\mathcal{A})$ reaches 🟥 empty"?



25 years of (un)decidability results for PTAs

Key problem considered: EF-emptiness

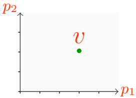
- "given a PTA $\mathcal{A}$ and a location, is the set of parameter valuations v such that $v(\mathcal{A})$ reaches empty"?

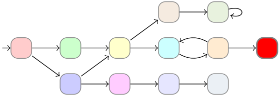


25 years of (un)decidability results for PTAs

Key problem considered: EF-emptiness

- "given a PTA $\mathcal{A}$ and a location , is the set of parameter valuations v such that $v(\mathcal{A})$ reaches  empty"?

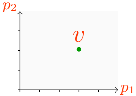


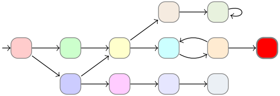


25 years of (un)decidability results for PTAs

Key problem considered: EF-emptiness

- "given a PTA $\mathcal{A}$ and a location , is the set of parameter valuations v such that $v(\mathcal{A})$ reaches empty"?





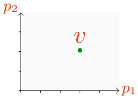
Large collection of results

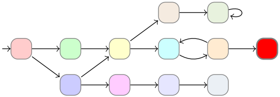
Survey: **[André, STTT (2017)]**

25 years of (un)decidability results for PTAs

Key problem considered: EF-emptiness

- "given a PTA $\mathcal{A}$ and a location , is the set of parameter valuations v such that $v(\mathcal{A})$ reaches empty"?





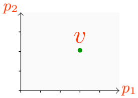
Large collection of results — Survey: **[André, STTT (2017)]**

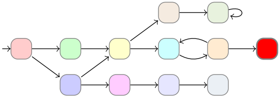
- Undecidable
  - Undecidable for only 3 clocks [Alur et al., 1993]
  - Undecidable for only 1 clock compared to parameters [Miller, 2000]
  - Undecidable with only strict constraints ($<, >$) [Doyen, 2007]
  - Undecidable for only one parameter [Beneš et al., 2015]

25 years of (un)decidability results for PTAs

Key problem considered: EF-emptiness

- "given a PTA $\mathcal{A}$ and a location, is the set of parameter valuations v such that $v(\mathcal{A})$ reaches empty"?





Large collection of results

Survey: **[André, STTT (2017)]**

- Undecidable
  - Undecidable for only 3 clocks [Alur et al., 1993]
  - Undecidable for only 1 clock compared to parameters [Miller, 2000]
  - Undecidable with only strict constraints $(<, >)$ [Doyen, 2007]
  - Undecidable for only one parameter [Beneš et al., 2015]
- Decidable
  - Limiting the number of clocks [Alur et al., 1993, Bundala and Ouaknine, 2014, Beneš et al., 2015]
  - Bounded integer-valued parameters [Jovanović et al., 2015]
  - Restricting the use of parameters [Hune et al., 2002]

Contributions: new (un)decidability results for PTAs

Investigating further problems for parametric timed automata

- Short version: all non-trivial problems are undecidable for PTAs

Contributions: new (un)decidability results for PTAs

Investigating further problems for parametric timed automata

- Short version: all non-trivial problems are undecidable for PTAs
- Long version: see manuscript (Chapter 3)

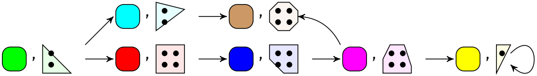
Contributions: new (un)decidability results for PTAs

Investigating further problems for parametric timed automata

- Short version: all non-trivial problems are undecidable for PTAs
- Long version: see manuscript (Chapter 3)

A new subclass: integer-point PTAs (IP-PTAs) **[ICFEM'16]**

- "Each symbolic state (polyhedron) contains an integer point"



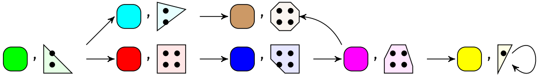
Contributions: new (un)decidability results for PTAs

Investigating further problems for parametric timed automata

- Short version: all non-trivial problems are undecidable for PTAs
- Long version: see manuscript (Chapter 3)

A new subclass: integer-point PTAs (IP-PTAs) **[ICFEM'16]**

- "Each symbolic state (polyhedron) contains an integer point"



- Good news: EF-emptiness is decidable (for bounded parameters)

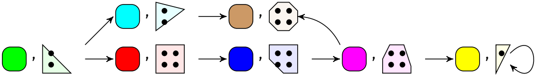
Contributions: new (un)decidability results for PTAs

Investigating further problems for parametric timed automata

- Short version: all non-trivial problems are undecidable for PTAs
- Long version: see manuscript (Chapter 3)

A new subclass: integer-point PTAs (IP-PTAs) **[ICFEM'16]**

- "Each symbolic state (polyhedron) contains an integer point"



- Good news: EF-emptiness is decidable (for bounded parameters)
- Bad news: it is not possible to decide whether a PTA is IP

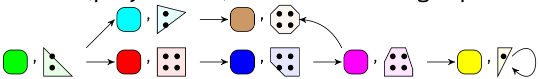
Contributions: new (un)decidability results for PTAs

Investigating further problems for parametric timed automata

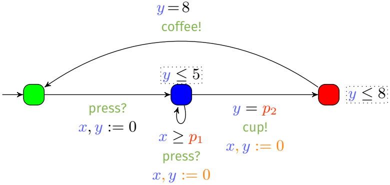
- Short version: all non-trivial problems are undecidable for PTAs
- Long version: see manuscript (Chapter 3)

A new subclass: integer-point PTAs (IP-PTAs) **[ICFEM'16]**

- "Each symbolic state (polyhedron) contains an integer point"

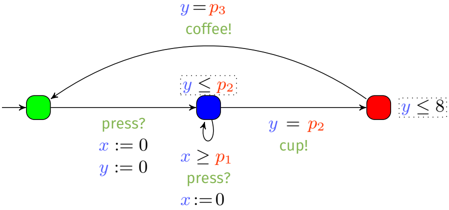


- Good news: EF-emptiness is decidable (for bounded parameters)
- Bad news: it is not possible to decide whether a PTA is IP
- Good news: syntactic subclass: reset-PTA
  - "Whenever a clock is compared to a parameter, all clocks must be reset"



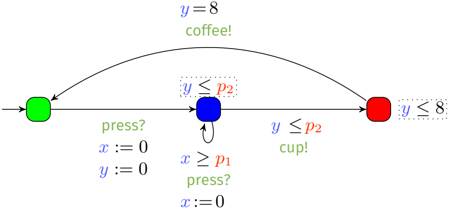
16 years of (un)decidability results for L/U-PTAs

- Subclass of PTAs partitioning parameters into upper-bound parameters ($x \leq p$, $x < p$) and lower-bound parameters ($x \geq p$, $x > p$)



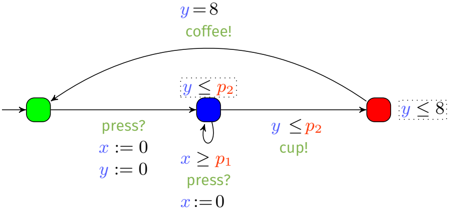
16 years of (un)decidability results for L/U-PTAs

- Subclass of PTAs partitioning parameters into upper-bound parameters ($x \leq p$, $x < p$) and lower-bound parameters ($x \geq p$, $x > p$)



16 years of (un)decidability results for L/U-PTAs

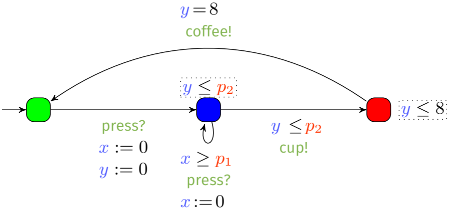
- Subclass of PTAs partitioning parameters into upper-bound parameters ($x \leq p$, $x < p$) and lower-bound parameters ($x \geq p$, $x > p$)



- EF-emptiness and -universality are decidable [Hune et al., 2002]

16 years of (un)decidability results for L/U-PTAs

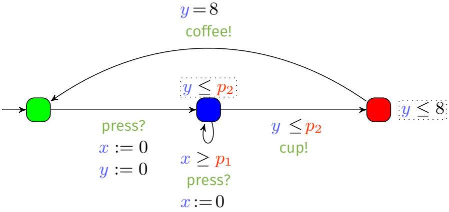
- Subclass of PTAs partitioning parameters into upper-bound parameters ($x \leq p$, $x < p$) and lower-bound parameters ($x \geq p$, $x > p$)



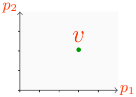
- EF-emptiness and -universality are decidable [Hune et al., 2002]
- Büchi-emptiness is decidable [Bozzelli and La Torre, 2009]

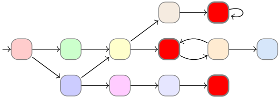
16 years of (un)decidability results for L/U-PTAs

- Subclass of PTAs partitioning parameters into upper-bound parameters ($x \leq p$, $x < p$) and lower-bound parameters ($x \geq p$, $x > p$)



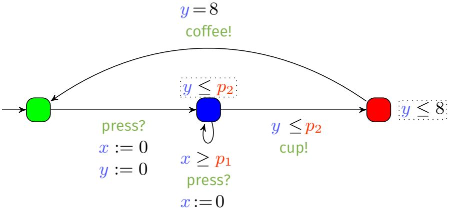
- EF-emptiness and -universality are decidable [Hune et al., 2002]
- Büchi-emptiness is decidable [Bozzelli and La Torre, 2009]
- AF-emptiness is undecidable [Jovanović et al., 2015]



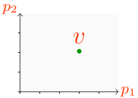


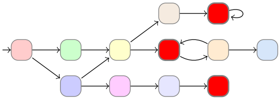
16 years of (un)decidability results for L/U-PTAs

- Subclass of PTAs partitioning parameters into upper-bound parameters ($x \leq p$, $x < p$) and lower-bound parameters ($x \geq p$, $x > p$)



- EF-emptiness and -universality are decidable [Hune et al., 2002]
- Büchi-emptiness is decidable [Bozzelli and La Torre, 2009]
- AF-emptiness is undecidable [Jovanović et al., 2015]

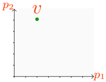


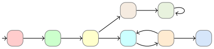


- Seems to allow for interesting applications

Contributions: new (un)decidability results for L/U-PTAs

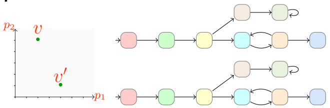
- Language-preservation problem: undecidable **[FORMATS'15]**





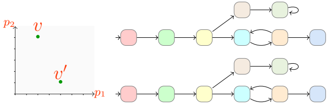
Contributions: new (un)decidability results for L/U-PTAs

- Language-preservation problem: undecidable **[FORMATS'15]**

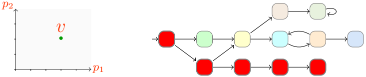


Contributions: new (un)decidability results for L/U-PTAs

- Language-preservation problem: undecidable **[FORMATS'15]**

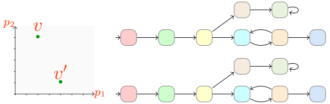


- Deadlock-existence-emptiness: undecidable **[ACSD'17]**

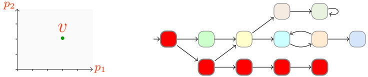


Contributions: new (un)decidability results for L/U-PTAs

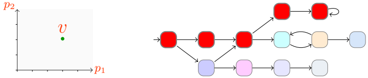
- Language-preservation problem: undecidable **[FORMATS'15]**



- Deadlock-existence-emptiness: undecidable **[ACSD'17]**

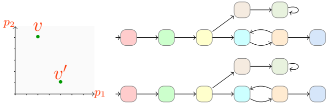


- Cycle-existence-emptiness: decidable **[ACSD'17]**

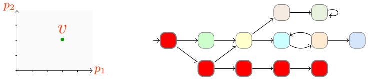


Contributions: new (un)decidability results for L/U-PTAs

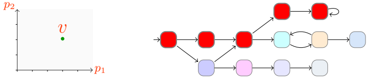
- Language-preservation problem: undecidable [FORMATS'15]



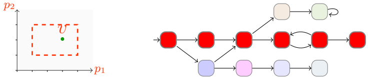
- Deadlock-existence-emptiness: undecidable [ACSD'17]



- Cycle-existence-emptiness: decidable [ACSD'17]

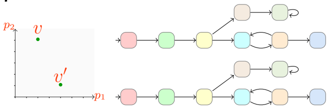


- EG-emptiness: decidable only if the parameters are bounded with closed bounds [ACSD'17]

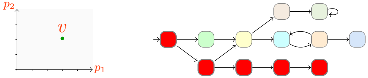


Contributions: new (un)decidability results for L/U-PTAs

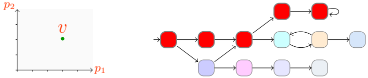
- Language-preservation problem: undecidable **[FORMATS'15]**



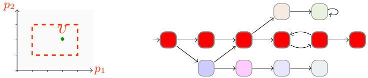
- Deadlock-existence-emptiness: undecidable **[ACSD'17]**



- Cycle-existence-emptiness: decidable **[ACSD'17]**



- EG-emptiness: decidable only if the parameters are bounded with closed bounds **[ACSD'17]**



- Full (T)CTL-emptiness: undecidable even for U-PTAs **[FORMATS'18]**

Mathias Ramparison's PhD thesis

Summary of theoretical contributions

<table>
<tr><th>Class</th><th>U-PTAs</th><th colspan="2">bL/U-PTAs</th><th>L/U-PTAs</th><th>bPTAs</th><th>PTAs</th></tr>
<tr><td>emptiness</td><td></td><td>closed</td><td>open</td><td></td><td></td><td></td></tr>
<tr><td>EF</td><td>[Hune et al., 2002]</td><td>[ICFEM’16]</td><td>(WiP)</td><td>[Hune et al., 2002]</td><td>[Miller, 2000]</td><td>[Alur et al., 1993]</td></tr>
<tr><td>AF</td><td>open</td><td colspan="2">[ICFEM’16]</td><td>[Jovanović et al., 2015]</td><td>[ICFEM’16]</td><td>[Jovanović et al., 2015]</td></tr>
<tr><td>EG</td><td>open</td><td>[ACSD’17]</td><td colspan="4">[ACSD’17]</td></tr>
<tr><td>AG</td><td>[ACSD’17]</td><td>[ICFEM’16]</td><td>(WiP)</td><td>[ACSD’17]</td><td colspan="2">[ICFEM’16]</td></tr>
<tr><td>TCTL</td><td>[FORMATS’18]</td><td colspan="2">[ICFEM’16]</td><td>[Jovanović et al., 2015]</td><td>[Miller, 2000]</td><td>[Alur et al., 1993]</td></tr>
<tr><td>EC</td><td>[ACSD’17]</td><td>[ACSD’17]</td><td>open</td><td>[ACSD’17]</td><td colspan="2">[ACSD’17]</td></tr>
<tr><td>ED</td><td>open</td><td colspan="4">[ACSD’17]</td><td>[ICTAC’16]</td></tr>
<tr><td>Lg-Pres.</td><td>open</td><td colspan="5">[FORMATS’15]</td></tr>
<tr><td>Trace-Pres.</td><td>open</td><td colspan="5">[FORMATS’15]</td></tr>
</table>

[FORMATS’15]	É. André and N. Markey
[ICTAC’16]	É. André
[ICFEM’16]	É. André, D. Lime and O. H. Roux
[ACSD’17]	É. André and D. Lime
[FORMATS’18]	É. André, D. Lime and M. Ramparison
WiP	Work in progress (decidable)

Decidable
Open
Undecidable

bL/U-PTA: bounded L/U-PTAs

Perspectives

Less expressive classes

- A quite unexplored formalism: U-PTA
  - Still able to model interesting systems

More expressive classes

- Extension to hybrid systems
  - Clocks become variables with arbitrary (and different) rates

Outline

Efficient synthesis: Motivation

- Parametric timed automata are very expressive

  [André, Lime, Roux, FORMATS'16]

- But most problems are undecidable
- Still, they represent an excellent opportunity for pragmatic parametric model checking

Efficient synthesis: Motivation

- Parametric timed automata are very expressive

  [André, Lime, Roux, FORMATS'16]

- But most problems are undecidable
- Still, they represent an excellent opportunity for pragmatic parametric model checking

Goal

Design efficient parameter synthesis algorithms

Efficient synthesis: Motivation

- Parametric timed automata are very expressive

  [André, Lime, Roux, FORMATS'16]

- But most problems are undecidable
- Still, they represent an excellent opportunity for pragmatic parametric model checking

Goal

Design efficient parameter synthesis algorithms

Two possible directions:

1. Achieving termination without guarantee on the completeness
2. Achieving exact synthesis without guarantee on termination

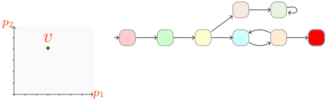
Outline

Parametric reachability preservation

Parametric reachability preservation problem

Input: a PTA $\mathcal{A}$, a goal location 🟥, a parameter valuation v

Problem: synthesize valuations v' such that $v(\mathcal{A})$ reaches 🟥 iff $v'(\mathcal{A})$ reaches 🟥

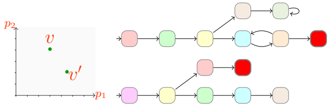


Parametric reachability preservation

Parametric reachability preservation problem

Input: a PTA $\mathcal{A}$, a goal location 🟥, a parameter valuation v

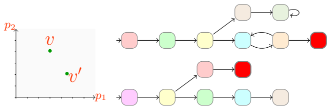
Problem: synthesize valuations v' such that $v(\mathcal{A})$ reaches 🟥 iff $v'(\mathcal{A})$ reaches 🟥



Parametric reachability preservation

Parametric reachability preservation problem

Input: a PTA $\mathcal{A}$, a goal location 🟥, a parameter valuation v
Problem: synthesize valuations v' such that $v(\mathcal{A})$ reaches 🟥 iff $v'(\mathcal{A})$ reaches 🟥

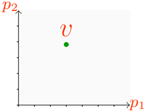


Reachability-preservation-emptiness problem undecidable

[André, Lipari, Nguyen, Sun, NFM'15]

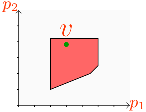
Parametric reachability preservation: An algorithm

A pragmatic procedure: $\mathsf{PRP}(\mathcal{A}, v)$



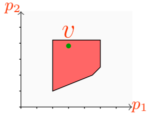
Parametric reachability preservation: An algorithm

A pragmatic procedure: $\mathsf{PRP}(\mathcal{A}, v)$



Parametric reachability preservation: An algorithm

A pragmatic procedure: $\mathsf{PRP}(\mathcal{A}, v)$

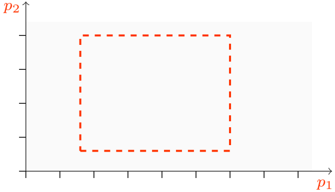


- Built on top of two existing algorithms
  - Reachability synthesis (EFsynth) [Alur et al., 1993, Jovanović et al., 2015]
  - Trace-preservation-synthesis (IM) **[André et al., 2009]**
- Key heuristics: only explore behaviors "similar" to that of $v(\mathcal{A})$

$\rightsquigarrow$ Non-necessarily terminating, incomplete on purpose, but fast in practice

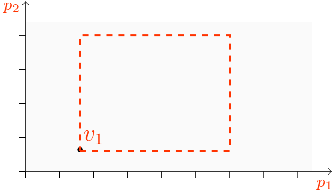
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



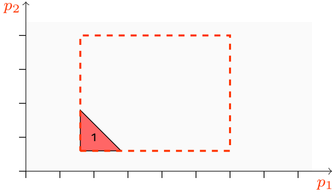
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



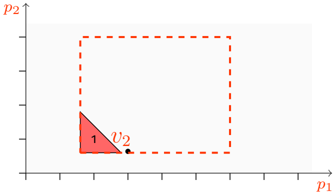
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



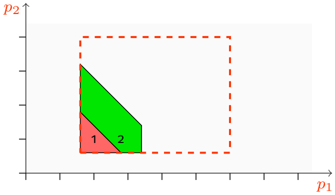
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



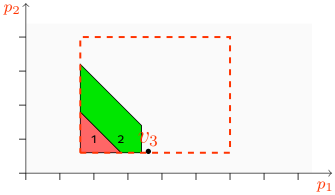
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



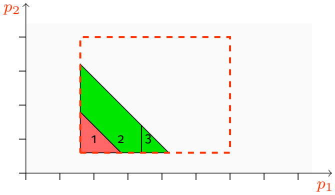
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



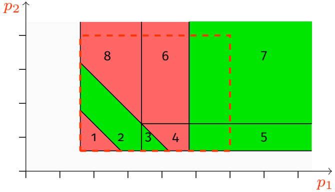
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



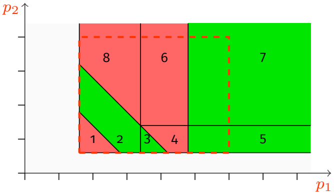
Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



Solving reachability synthesis using PRP

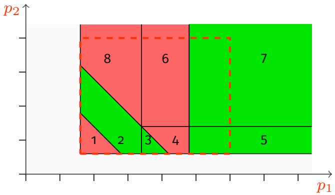
- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



  - Principle: "many small analyses rather than one big analysis" $\rightsquigarrow$ memory gain

Solving reachability synthesis using PRP

- Idea: select valuations in a bounded parameter domain, and call PRP on these valuations, until a sufficient coverage is reached: algorithm PRPC



- Principle: “many small analyses rather than one big analysis” $\rightsquigarrow$ memory gain
- Unexpected: time gain in several cases too!

Case study	Clocks	Points	EFsynth	BC	PRPC
toy	2	2 601	0.401	∞	0.078
Sched1	13	6 561	∞	∞	1 595
Sched2.50.0	6	3 321	9.25	990	14.55
Sched2.50.2	6	3 321	662	∞	213
Sched2.100.0	6	972 971	21.4	2 093	116
Sched2.100.2	6	972 971	3 757	∞	4 557
Sched5	21	1 681	352	∞	∞
SPSMALL	11	3 082	7.49	587	118

BC: former algorithm **[André and Fribourg, 2010]**

Towards distributed parameter synthesis

- Point-based algorithms (that iterate on parameter valuations) can obviously be distributed
  - Distribution over a cluster: many computers with their own memory (communication through a network)

Towards distributed parameter synthesis

- Point-based algorithms (that iterate on parameter valuations) can obviously be distributed
  - Distribution over a cluster: many computers with their own memory (communication through a network)
- Challenge: How to efficiently distribute these algorithms?
  - “Shape” of the result unknown a priori
  - Risk of redundant computations

Towards distributed parameter synthesis

- Point-based algorithms (that iterate on parameter valuations) can obviously be distributed
  - Distribution over a cluster: many computers with their own memory (communication through a network)
- Challenge: How to efficiently distribute these algorithms?
  - "Shape" of the result unknown a priori
  - Risk of redundant computations
- We proposed several distribution policies
  - Most efficient: dynamic domain decomposition (25 times faster on 128 nodes)

[André, Coti, Evangelista, EuroMPI'14] [André, Coti, Nguyễn, ICFEM'15]

Nguyễn Hoàng Gia's PhD thesis

Towards distributed parameter synthesis

- Point-based algorithms (that iterate on parameter valuations) can obviously be distributed
  - Distribution over a cluster: many computers with their own memory (communication through a network)
- Challenge: How to efficiently distribute these algorithms?
  - "Shape" of the result unknown a priori
  - Risk of redundant computations
- We proposed several distribution policies
  - Most efficient: dynamic domain decomposition (25 times faster on 128 nodes)

[André, Coti, Evangelista, EuroMPI'14] [André, Coti, Nguyễn, ICFEM'15]

Nguyễn Hoàng Gia's PhD thesis

- Application to PRPC

Case study	Clocks	Points	EFsynth	BC	PRPC	PRPC distr(12)
toy	2	2 601	0.401	∞	0.078	0.050
Sched1	13	6 561	∞	∞	1 595	219
Sched2.50.0	6	3 321	9.25	990	14.55	4.77
Sched2.50.2	6	3 321	662	∞	213	84
Sched2.100.0	6	972 971	21.4	2 093	116	10.1
Sched2.100.2	6	972 971	3 757	∞	4 557	1 543
Sched5	21	1 681	352	∞	∞	917
SPSMALL	11	3 082	7.49	587	118	11.2

Outline

Compositional verification of timed systems

Learning an unknown timed system via interactions with a teacher

- Membership queries
- Candidate queries

Extension TL^* of the L^* algorithm [Angluin, 1987]

- Using a subclass of timed automata [Alur et al., 1999]
- More efficient than [Grinchtein et al., 2010]

[Lin, André et al., ATVA'11] [Lin, André et al., FM'12]

Compositional verification of timed systems

Learning an unknown timed system via interactions with a teacher

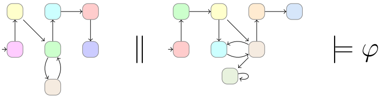
- Membership queries
- Candidate queries

Extension TL* of the L* algorithm [Angluin, 1987]

- Using a subclass of timed automata [Alur et al., 1999]
- More efficient than [Grinchtein et al., 2010]

[Lin, André et al., ATVA'11] [Lin, André et al., FM'12]

Use TL* to learn an abstraction of a component (assume-guarantee reasoning)



Compositional verification of timed systems

Learning an unknown timed system via interactions with a teacher

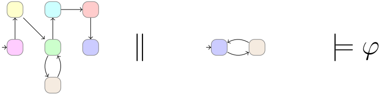
- Membership queries
- Candidate queries

Extension TL^* of the L^* algorithm [Angluin, 1987]

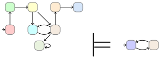
- Using a subclass of timed automata [Alur et al., 1999]
- More efficient than [Grinchtein et al., 2010]

[Lin, André et al., ATVA'11] [Lin, André et al., FM'12]

Use TL^* to learn an abstraction of a component (assume-guarantee reasoning)



with



Compositional verification of timed systems

Learning an unknown timed system via interactions with a teacher

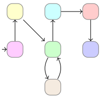
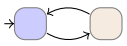
- Membership queries
- Candidate queries

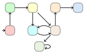
Extension TL* of the L* algorithm [Angluin, 1987]

- Using a subclass of timed automata [Alur et al., 1999]
- More efficient than [Grinchtein et al., 2010]

[Lin, André et al., ATVA'11] [Lin, André et al., FM'12]

Use TL* to learn an abstraction of a component (assume-guarantee reasoning)

 $\parallel$  $\models \varphi$

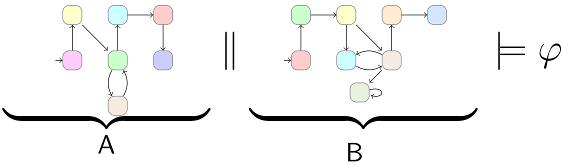
with  $\models$ 

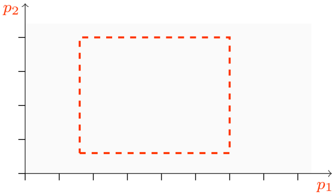
- Membership and candidate queries are performed by model checking
- Much faster than monolithic verification **[Lin, André et al., TSE 2014]**

Compositional parameter synthesis

[André and Lin, FORTE'17]

Given a parametric component A and a non-parametric component B



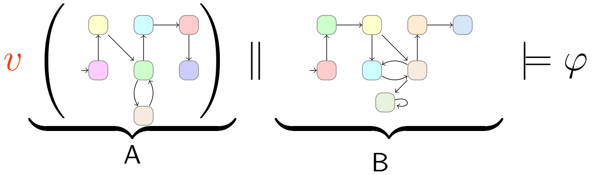


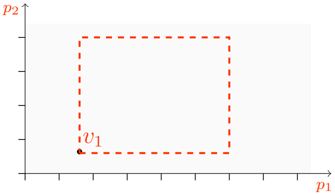
Compositional parameter synthesis

[André and Lin, FORTE'17]

Given a parametric component A and a non-parametric component B

1. Pick a parameter valuation v

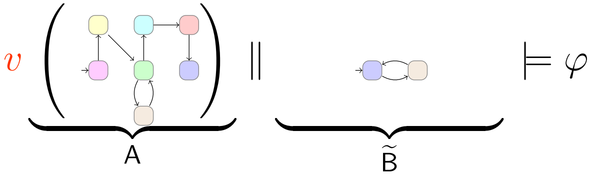


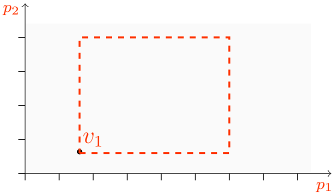


Compositional parameter synthesis [André and Lin, FORTE'17]

Given a parametric component A and a non-parametric component B

1. Pick a parameter valuation v
2. Compute an abstraction $\widetilde{B}$ of B



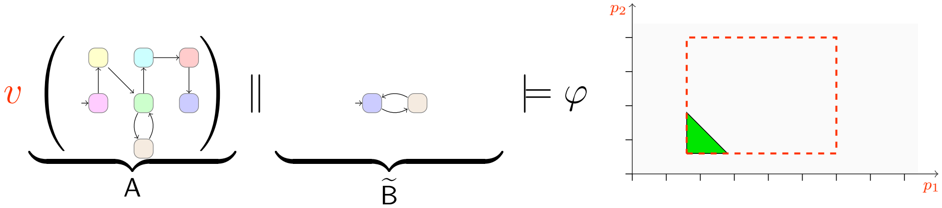


Compositional parameter synthesis

[André and Lin, FORTE'17]

Given a parametric component A and a non-parametric component B

1. Pick a parameter valuation v
2. Compute an abstraction $\widetilde{\mathsf{B}}$ of B
3. If $v(\mathsf{A}) \parallel \widetilde{\mathsf{B}} \models \varphi$, synthesize $\mathsf{PRP}(\mathsf{A} \parallel \widetilde{\mathsf{B}}, v)$



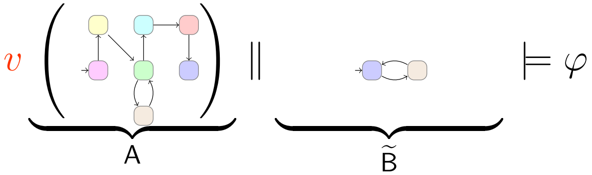
Compositional parameter synthesis

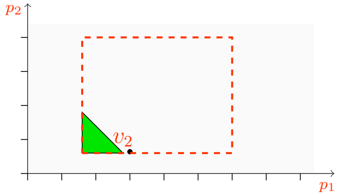
[André and Lin, FORTE'17]

Given a parametric component A and a non-parametric component B

1. Pick a parameter valuation v
2. Compute an abstraction $\widetilde{\mathsf{B}}$ of B
3. If $v(\mathsf{A}) \parallel \widetilde{\mathsf{B}} \models \varphi$, synthesize $\mathsf{PRP}(\mathsf{A} \parallel \widetilde{\mathsf{B}}, v)$

4. Find another point and restart



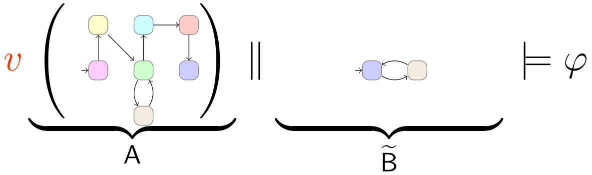


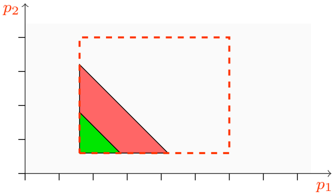
Compositional parameter synthesis

[André and Lin, FORTE'17]

Given a parametric component A and a non-parametric component B

1. Pick a parameter valuation v
2. Compute an abstraction $\widetilde{B}$ of B
3. If $v(A) \parallel \widetilde{B} \models \varphi$, synthesize $PRP(A \parallel \widetilde{B}, v)$
   Else generalize the counter-example (cheap)
4. Find another point and restart



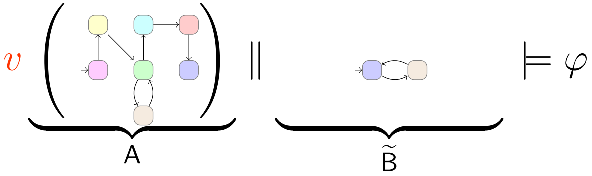


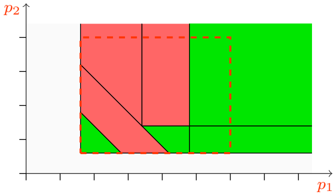
Compositional parameter synthesis

[André and Lin, FORTE'17]

Given a parametric component A and a non-parametric component B

1. Pick a parameter valuation v
2. Compute an abstraction $\widetilde{\mathsf{B}}$ of B
3. If $v(\mathsf{A}) \parallel \widetilde{\mathsf{B}} \models \varphi$, synthesize $\mathsf{PRP}(\mathsf{A} \parallel \widetilde{\mathsf{B}}, v)$
   Else generalize the counter-example (cheap)
4. Find another point and restart





Compositional parameter synthesis: experiments

Toolkit made of IMITATOR and CV, interfaced by a Python script

Case study	$\#\mathcal{A}$	$\#X$	$\#P$	Spec	EFsynth	CompSynth			
						#abs	#c.-ex.	learning	total
FMS-1	6	18	2	1	0.299	1	1	0.074	0.136
				2	0.010	0	1	0.038	0.046
				3	0.282	1	0	0.090	0.242
FMS-2	11	37	2	1	∞	1	1	84.2	88.9
				2	∞	1	0	81.4	85.2
				3	0.051	0	2	1.10	2.44
				4	0.062	0	1	1.42	1.53
				5	∞	1	0	31.4	40.8
				6	∞	1	0	37.2	42.4
AIP	11	46	2	1	0.551	0	1	0.086	0.114
				2	2.11	0	1	1.22	1.25
				3	3.91	0	1	8.50	8.54
				4	0.235	1	1	8.39	8.42
				5	∞	1	0	0.394	0.871
				6	∞	1	0	5.32	9.58
				7	∞	1	0	1.76	3.19
				8	∞	1	0	1.13	4.35
				9	∞	1	1	0.762	1.84
				10	0.022	0	1	0.072	0.094
Fischer-3	5	12	2		2.76	0	1	-	∞
Fischer-4	6	16	2		∞	0	1	-	∞

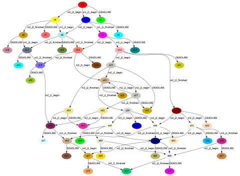
Works well when loosely synchronized and loosely timed

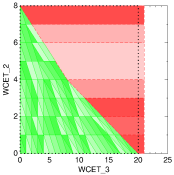
Outline

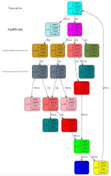
IMITATOR

- A tool for modeling and verifying timed concurrent systems with unknown constants modeled with parametric timed automata
  - Communication through (strong) broadcast synchronization
  - Rational-valued shared discrete variables
  - Stopwatches, to model schedulability problems with preemption

- Synthesis algorithms
  - (non-Zeno) parametric model checking (using a subset of TCTL)
  - Language and trace preservation, and robustness analysis
  - Parametric deadlock-freeness checking







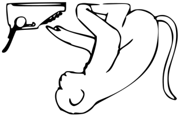
IMITATOR

Under continuous development since 2008 [André et al., FM'12]

A library of benchmarks

- Communication protocols
- Schedulability problems
- Asynchronous circuits
- ...and more

Free and open source software: Available under the GNU-GPL license



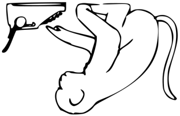
IMITATOR

Under continuous development since 2008 [André et al., FM'12]

A library of benchmarks

- Communication protocols
- Schedulability problems
- Asynchronous circuits
- ...and more

Free and open source software: Available under the GNU-GPL license



Try it!

www.imitator.fr

Some success stories

- Modeled and verified an asynchronous memory circuit by ST-Microelectronics
  - Project ANR Valmem
- Parametric schedulability analysis of a prospective architecture for the flight control system of the next generation of spacecrafts designed at ASTRIUM Space Transportation [Fribourg et al., 2012]
- Verification of software product lines [Luthmann et al., 2017]
- Formal timing analysis of music scores [Fanchon and Jacquemard, 2013]
- Solution to a challenge related to a distributed video processing system by Thales

Perspectives

- Beyond distributed verification
  - Multicore verification
  - Swarm verification
- Combine non-parametric and parametric analyses
  - Machine learning
    - "Learn" a constraint by repeated call to a non-parametric model checker (much faster)
    - Preliminary works in **[Li, Sun, Gao, André, ICFEM'17]**

Outline

Context: schedulability analysis

Real-time system:

- Set of tasks (with a period, a WCET and a deadline)
- One processor (uniprocessor) or more (multiprocessor)
- Scheduling policies: fixed priority (FPS), earliest deadline first (EDF)...

Definition (Schedulability analysis)

Given a real-time system and a scheduling policy, certify that no deadline miss will ever occur

Context: schedulability analysis

Real-time system:

- Set of tasks (with a period, a WCET and a deadline)
- One processor (uniprocessor) or more (multiprocessor)
- Scheduling policies: fixed priority (FPS), earliest deadline first (EDF)...

Definition (Schedulability analysis)

Given a real-time system and a scheduling policy, certify that no deadline miss will ever occur

Solved in [Liu and Layland, 1973]*

Context: schedulability analysis

Real-time system:

- Set of tasks (with a period, a WCET and a deadline)
- One processor (uniprocessor) or more (multiprocessor)
- Scheduling policies: fixed priority (FPS), earliest deadline first (EDF)...

Definition (Schedulability analysis)

Given a real-time system and a scheduling policy, certify that no deadline miss will ever occur

Solved in [Liu and Layland, 1973]*

*for fixed priority, for a single processor, without jitter, without sporadic tasks, without preemption, without precedence constraints, without resource sharing, without uncertainty...

Context: schedulability analysis

Real-time system:

- Set of tasks (with a period, a WCET and a deadline)
- One processor (uniprocessor) or more (multiprocessor)
- Scheduling policies: fixed priority (FPS), earliest deadline first (EDF)...

Definition (Schedulability analysis)

Given a real-time system and a scheduling policy, certify that no deadline miss will ever occur

Solved in [Liu and Layland, 1973]*

*for fixed priority, for a single processor, without jitter, without sporadic tasks, without preemption, without precedence constraints, without resource sharing, without uncertainty...

In general, schedulability analysis is hard

Outline

Schedulability analysis with parametric model checking

Goal: parametric schedulability analysis

Given a real-time system and a scheduling policy, synthesize valuations (deadlines, periods…) such that the system is schedulable.

Modeling a real-time system with PTAs

- Each task or chain of task: one PTA
- Each scheduler: one PTA
- Use stopwatches to model preemption

Schedulability analysis with parametric model checking

Goal: parametric schedulability analysis

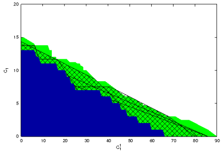
Given a real-time system and a scheduling policy, synthesize valuations (deadlines, periods…) such that the system is schedulable.

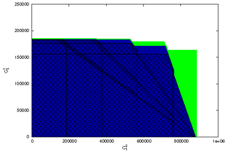
Modeling a real-time system with PTAs

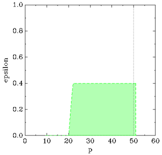
- Each task or chain of task: one PTA
- Each scheduler: one PTA
- Use stopwatches to model preemption

Comparison with analytical methods **[Sun, Soulat, Lipari, André, Fribourg, FTSCS'13]**

- Much better in terms of completeness
- And can evaluate robustness







Romain Soulat's PhD thesis

Outline

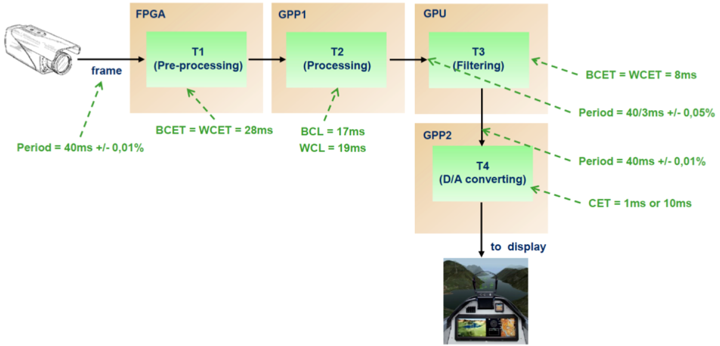
The Thales challenge (1/2)

"FMTV challenge" by Thales proposed during the WATERS 2014 workshop

Solutions presented at WATERS 2015

System: an unmanned aerial video system

- Architecture: 4 processors, 4 tasks, 2 buffers
- ...with uncertain periods
  - Period constant but with a small uncertainty (typically 0.01 %)
  - Not a jitter!



The Thales challenge (2/2)

Goal

Compute the end-to-end BCET and WCET times for a buffer size of 1 and 3

The Thales challenge (2/2)

Goal

Compute the end-to-end BCET and WCET times for a buffer size of 1 and 3

Challenging!

- Distributed system (multiprocessor)
- Buffers
- Dependencies between tasks
- Uncertain periods

The Thales challenge (2/2)

Goal

Compute the end-to-end BCET and WCET times for a buffer size of 1 and 3

Challenging!

- Distributed system (multiprocessor)
- Buffers
- Dependencies between tasks
- Uncertain periods

A typical parameter synthesis problem

- The end-to-end time can be set as a parameter… to be synthesized
- The uncertain period is typically a parameter (with some constraint, e. g., $P1 \in [40 - 0.004, 40 + 0.004]$)

Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time

Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame

Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location

Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location
4. Gather all constraints (in as many dimensions as uncertain periods + the end-to-end time)

Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location
4. Gather all constraints (in as many dimensions as uncertain periods + the end-to-end time)
5. Eliminate all parameters but the end-to-end time

Note: not eliminating parameters allows one to know for which values of the periods the best / worst case execution times are obtained.

Methodology

1. Propose a PTA model with parameters for uncertain periods and the end-to-end time
2. Add a specific location corresponding to the correct transmission of the frame
3. Run the reachability synthesis algorithm EFsynth (implemented in IMITATOR) w.r.t. that location
4. Gather all constraints (in as many dimensions as uncertain periods + the end-to-end time)
5. Eliminate all parameters but the end-to-end time
6. Exhibit the minimum and the maximum

Note: not eliminating parameters allows one to know for which values of the periods the best / worst case execution times are obtained.

Results obtained by IMITATOR

E2E latency results for the two buffer sizes

Buffer size $\rightarrow$	1	3
min E2E	63 ms	63 ms
max E2E	145.008 ms	225.016 ms

Results obtained using IMITATOR in a few seconds

[André, Lipari, Sun, WATERS'15]

Perspectives

- Better scalability
  - Design dedicated synthesis algorithms
  - Compositional synthesis
- Better integration
  - Parametric task automata **[André, FMICS'17]**
  - Support of existing industrial formalisms in IMITATOR

Outline

Summary of contributions

- Theory
  - New decidable subclasses of parametric timed automata
- Efficient synthesis algorithms
  - Implementation in IMITATOR
- Application to real-time systems
  - Application to industrial case studies

Also (not presented)

- Robustness of timed concurrent systems
- Formal specification of (timed) concurrent systems
  - **Mahdi Benmoussa's PhD thesis**

General perspectives

- Timing parameters in more complex settings
  - Probabilities:
    - preliminary works in **[André and Delahaye, TIME'16]**
  - Hybrid systems
- More parameters
  - Probabilistic parameters
  - Discrete parameters (networks of identical processes)
- Beyond parameter synthesis: controller synthesis
  - More abstraction
  - More uncertainty
- More applications
  - Real-time systems
  - Biological systems
  - Cybersecurity

Bibliography

References I

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

Alur, R., Fix, L., and Henzinger, T. A. (1999).
Event-clock automata: A determinizable class of timed automata.
Theoretical Computer Science, 211(1-2):253–273.

Alur, R., Henzinger, T. A., and Vardi, M. Y. (1993).
Parametric real-time reasoning.
In Kosaraju, S. R., Johnson, D. S., and Aggarwal, A., editors, *STOC*, pages 592–601, New York, NY, USA. ACM.

André, É. (2016).
Parametric deadlock-freeness checking timed automata.
In Sampaio, A. C. A. and Wang, F., editors, *ICTAC*, volume 9965 of *Lecture Notes in Computer Science*, pages 469–478. Springer.

André, É. (2017a).
A unified formalism for monoprocessor schedulability analysis under uncertainty.
In Cavalcanti, A., Petrucci, L., and Seceleanu, C., editors, *FMICS-AVoCS*, volume 10471 of *Lecture Notes in Computer Science*, pages 100–115. Springer.
Best paper award.

References II

André, É. (2017b).
What's decidable about parametric timed automata?
International Journal on Software Tools for Technology Transfer.
To appear.

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., Coti, C., and Evangelista, S. (2014).
Distributed behavioral cartography of timed automata.
In Dongarra, J., Ishikawa, Y., and Atsushi, H., editors, *EuroMPI/ASIA*, pages 109–114. ACM.

André, É., Coti, C., and Nguyen, H. G. (2015a).
Enhanced distributed behavioral cartography of parametric timed automata.
In Butler, M., Conchon, S., and Zaïdi, F., editors, *ICFEM*, volume 9407 of *Lecture Notes in Computer Science*, pages 319–335. Springer.

André, É. and Delahaye, B. (2016).
Consistency in parametric interval probabilistic timed automata.
In Dyreson, C. E., Hansen, M. R., and Hunsberger, L., editors, *TIME*, pages 110–119. IEEE Computer Society.

André, É. and Fribourg, L. (2010).
Behavioral cartography of timed automata.
In Kučera, A. and Potapov, I., editors, *RP*, volume 6227 of *Lecture Notes in Computer Science*, pages 76–90. Springer.

References III

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

André, É. and Lime, D. (2017).
Liveness in L/U-parametric timed automata.
In Legay, A. and Schneider, K., editors, *ACSD*, pages 9–18. IEEE.

André, É., Lime, D., and Ramparison, M. (2018).
TCTL model checking lower/upper-bound parametric timed automata without invariants.
In *FORMATS*.
To appear.

André, É., Lime, D., and Roux, O. H. (2016a).
Decision problems for parametric timed automata.
In Ogata, K., Lawford, M., and Liu, S., editors, *ICFEM*, volume 10009 of *Lecture Notes in Computer Science*, pages 400–416. Springer.

André, É., Lime, D., and Roux, O. H. (2016b).
On the expressiveness of parametric timed automata.
In Fränzle, M. and Markey, N., editors, *FORMATS*, volume 9984 of *Lecture Notes in Computer Science*, pages 19–34. Springer.

References IV

André, É. and Lin, S.-W. (2017).
Learning-based compositional parameter synthesis for event-recording automata.
In Bouajjani, A. and Alexandra, S., editors, *FORTE*, volume 10321 of *Lecture Notes in Computer Science*, pages 17–32. Springer.
Best FORTE paper award and best DisCoTec paper award.

André, É., Lipari, G., Nguyen, H. G., and Sun, Y. (2015b).
Reachability preservation based parameter synthesis for timed automata.
In Havelund, K., Holzmann, G. J., and Joshi, R., editors, *NFM*, volume 9058 of *Lecture Notes in Computer Science*, pages 50–65. Springer.

André, É., Lipari, G., and Sun, Y. (2015c).
Verification of two real-time systems using parametric timed automata.
In Quinton, S. and Vardanega, T., editors, *WATERS*.

André, É. and Markey, N. (2015).
Language preservation problems in parametric timed automata.
In Sankaranarayanan, S. and Vicario, E., editors, *FORMATS*, volume 9268 of *Lecture Notes in Computer Science*, pages 27–43. Springer.

André, É. and Soulat, R. (2011).
Synthesis of timing parameters satisfying safety properties.
In Delzanno, G. and Potapov, I., editors, *RP*, volume 6945 of *Lecture Notes in Computer Science*, pages 31–44. Springer.

References V

Angluin, D. (1987).
Learning regular sets from queries and counterexamples.
Information and Computation, 75(2):87–106.

Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

Beneš, N., Bezděk, P., Larsen, K. G., and Srba, J. (2015).
Language emptiness of continuous-time parametric timed automata.
In *ICALP, Part II*, volume 9135 of *LNCS*, pages 69–81. Springer.

Bozzelli, L. and La Torre, S. (2009).
Decision problems for lower/upper bound parametric timed automata.
Formal Methods in System Design, 35(2):121–151.

Bundala, D. and Ouaknine, J. (2014).
Advances in parametric real-time reasoning.
In Csuhaj-Varjú, E., Dietzfelbinger, M., and Ésik, Z., editors, *MFCS, Part I*, volume 8634 of *Lecture Notes in Computer Science*, pages 123–134. Springer.

Doyen, L. (2007).
Robust parametric reachability for timed automata.
Information Processing Letters, 102(5):208–213.

References VI

Fanchon, L. and Jacquemard, F. (2013).
Formal timing analysis of mixed music scores.
In *ICMC*. Michigan Publishing.

Fersman, E., Krcál, P., Pettersson, P., and Yi, W. (2007).
Task automata: Schedulability, decidability and undecidability.
Information and Computation, 205(8):1149–1172.

Fribourg, L., Lesens, D., Moro, P., and Soulat, R. (2012).
Robustness analysis for scheduling problems using the inverse method.
In Reynolds, M., Terenziani, P., and Moszkowski, B., editors, *TIME*, pages 73–80. IEEE Computer Society Press.

Grinchtein, O., Jonsson, B., and Leucker, M. (2010).
Learning of event-recording automata.
Theoretical Computer Science, 411(47):4029–4054.

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.

Jovanović, A., Lime, D., and Roux, O. H. (2015).
Integer parameter synthesis for timed automata.
IEEE Transactions on Software Engineering, 41(5):445–461.

References VII

Larsen, K. G., Pettersson, P., and Yi, W. (1997).
UPPAAL in a nutshell.
International Journal on Software Tools for Technology Transfer, 1(1-2):134–152.

Li, J., Sun, J., Gao, B., and André, É. (2017).
Classification based parameter synthesis for parametric timed automata.
In Duan, Z. and Ong, L., editors, *ICFEM*. Springer.
To appear.

Lin, S.-W., André, É., Dong, J. S., Sun, J., and Liu, Y. (2011).
An efficient algorithm for learning event-recording automata.
In Bultan, T. and Hsiung, P.-A., editors, *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 463–472. Springer.

Lin, S.-W., André, É., Liu, Y., Sun, J., and Dong, J. S. (2014).
Learning assumptions for compositional verification of timed systems.
Transactions on Software Engineering, 40(2):137–153.

Lin, S.-W., Liu, Y., Sun, J., Dong, J. S., and André, É. (2012).
Automatic compositional verification of timed systems.
In Giannakopoulou, D. and Méry, D., editors, *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 272–276. Springer.

Liu, C. L. and Layland, J. W. (1973).
Scheduling algorithms for multiprogramming in a hard-real-time environment.
Journal of the ACM, 20(1):46–61.

References VIII

Luthmann, L., Stephan, A., Bürdek, J., and Lochau, M. (2017).
Modeling and testing product lines with unbounded parametric real-time constraints.
In Cohen, M. B., Acher, M., Fuentes, L., Schall, D., Bosch, J., Capilla, R., Bagheri, E., Xiong, Y., Troya, J., Cortés, A. R., and Benavides, D., editors, *SPLC, Volume A*, pages 104–113. ACM.

Miller, J. S. (2000).
Decidability and complexity results for timed automata and semi-linear hybrid automata.
In Lynch, N. A. and Krogh, B. H., editors, *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer.

Norström, C., Wall, A., and Yi, W. (1999).
Timed automata as task models for event-driven systems.
In *RTCSA*, pages 182–189. IEEE Computer Society.

Sun, Y., Soulat, R., Lipari, G., André, É., and Fribourg, L. (2013).
Parametric schedulability analysis of fixed priority real-time distributed systems.
In Artho, C. and Ölveczky, P., editors, *FSTCS*, volume 419 of *Communications in Computer and Information Science*, pages 212–228. Springer.

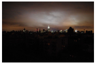
Summary of publications

Summary of publications

	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	Total
Books						1						1
Proceedings							2	1				3
International journals		1				2	2		2	1	1	9
International conferences	1	3	2	2	6	11	8	7	8	7	2	57
Other publications		2	2		2	1	2	1				10
Total	1	6	4	2	8	15	14	9	10	8	3	80

Additional explanation

Explanation for the 4 pictures in the beginning



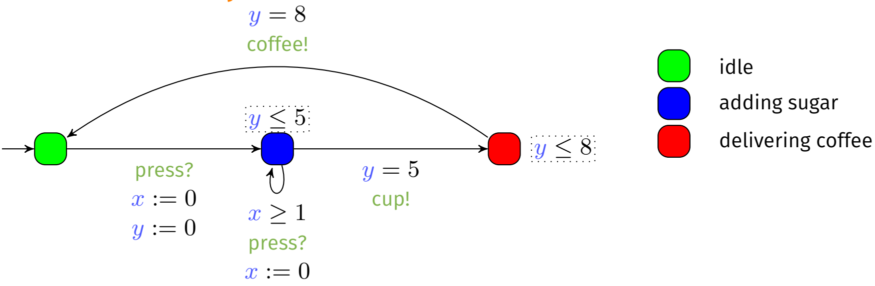
Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

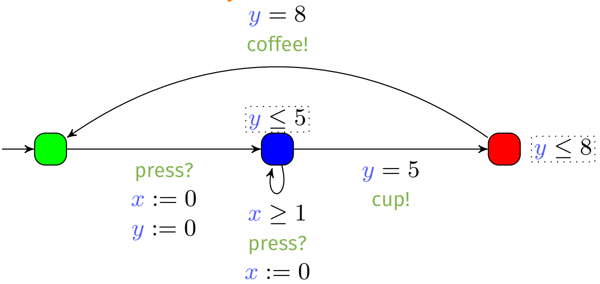
Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)

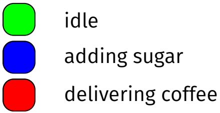
Varying the coffee machine

The most critical system: the coffee machine

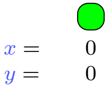


The most critical system: the coffee machine

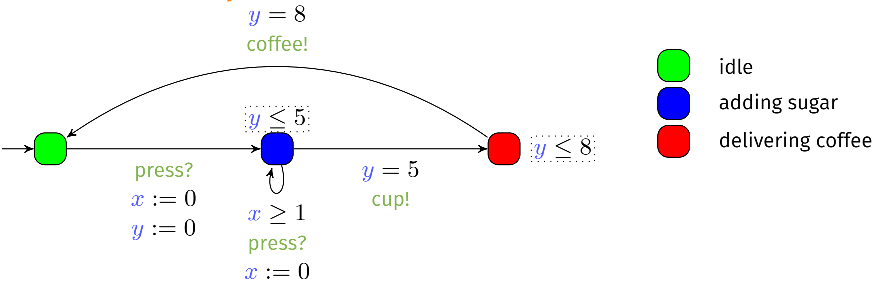




- Example of concrete run for the coffee machine
  - Coffee with no sugar



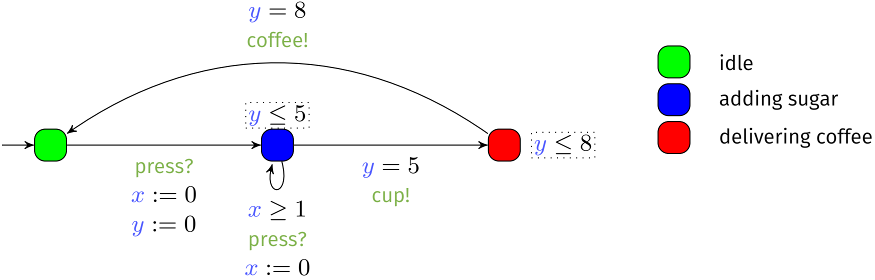
The most critical system: the coffee machine



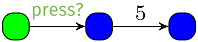
- Example of concrete run for the coffee machine
  - Coffee with no sugar

$x =$	0	0
$y =$	0	0

The most critical system: the coffee machine

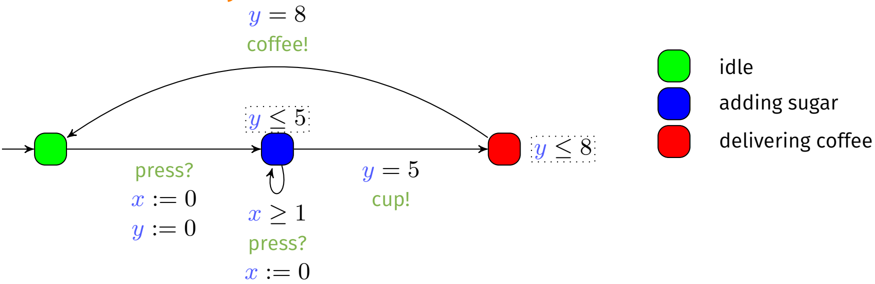


- Example of concrete run for the coffee machine
  - Coffee with no sugar

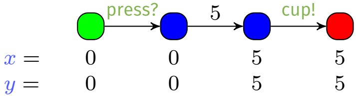


$x =$	0	0	5
$y =$	0	0	5

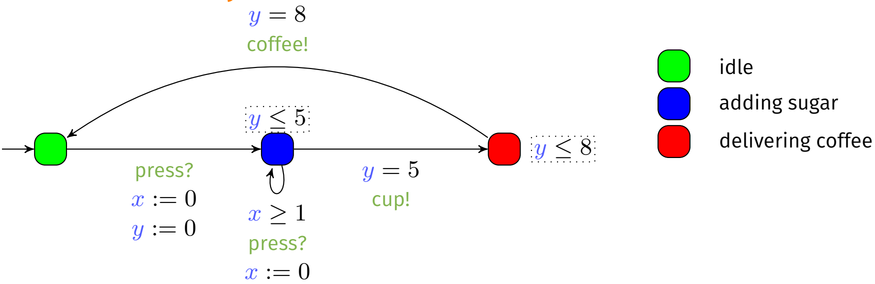
The most critical system: the coffee machine



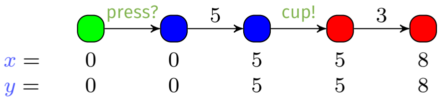
- Example of concrete run for the coffee machine
  - Coffee with no sugar



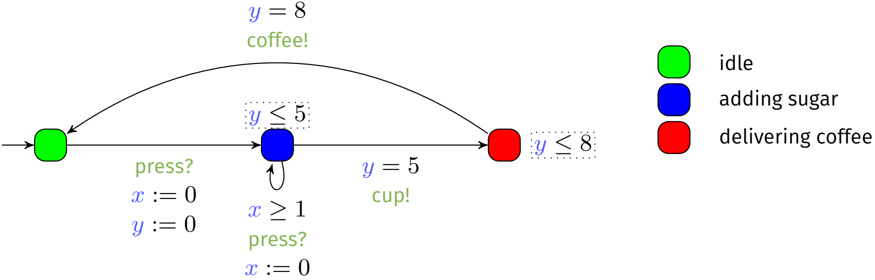
The most critical system: the coffee machine



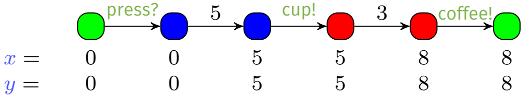
- Example of concrete run for the coffee machine
  - Coffee with no sugar



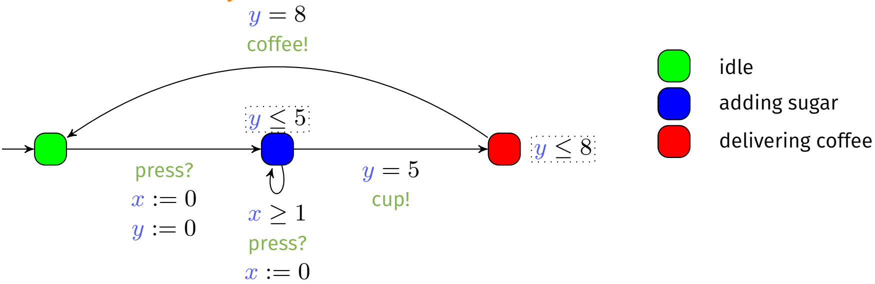
The most critical system: the coffee machine



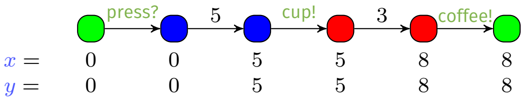
- Example of concrete run for the coffee machine
  - Coffee with no sugar



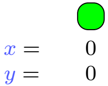
The most critical system: the coffee machine



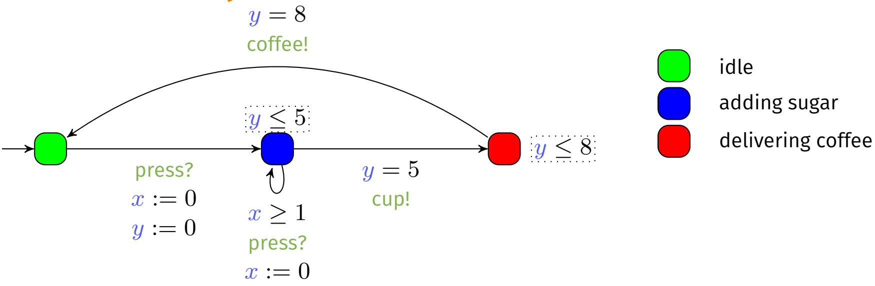
- Example of concrete run for the coffee machine
  - Coffee with no sugar



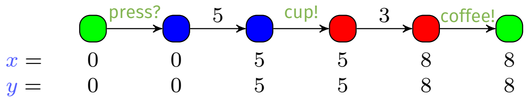
  - Coffee with 2 doses of sugar



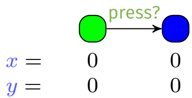
The most critical system: the coffee machine



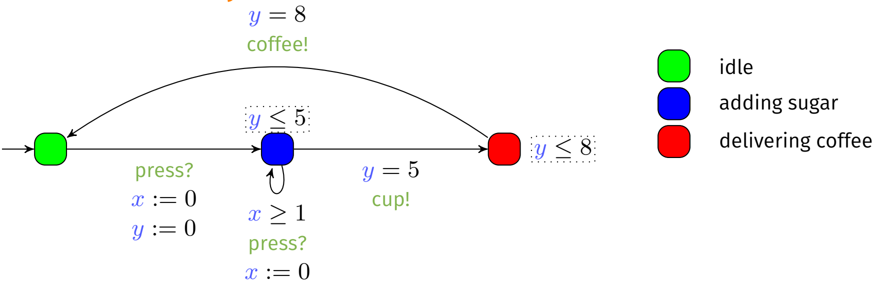
- Example of concrete run for the coffee machine
  - Coffee with no sugar



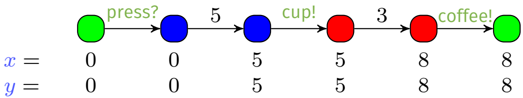
  - Coffee with 2 doses of sugar



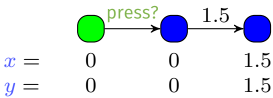
The most critical system: the coffee machine



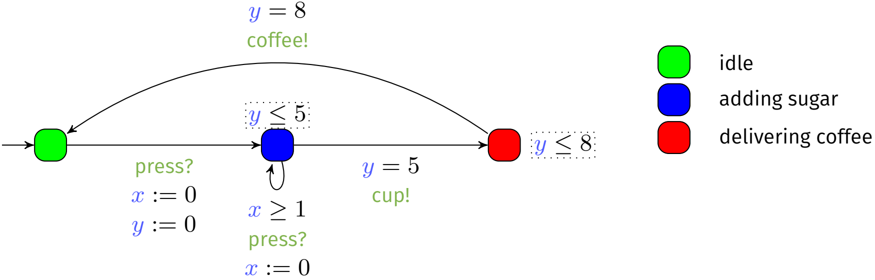
- Example of concrete run for the coffee machine
  - Coffee with no sugar



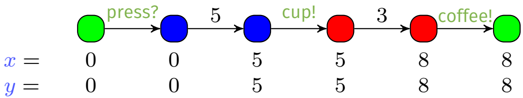
  - Coffee with 2 doses of sugar



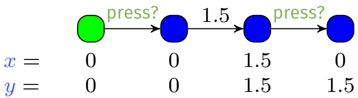
The most critical system: the coffee machine



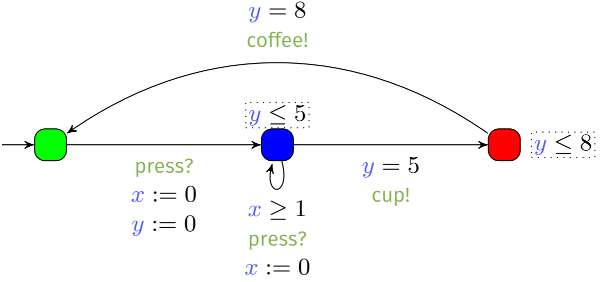
- Example of concrete run for the coffee machine
  - Coffee with no sugar

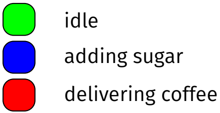


  - Coffee with 2 doses of sugar

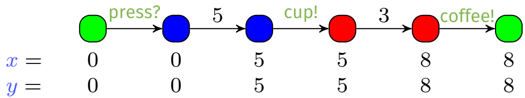


The most critical system: the coffee machine

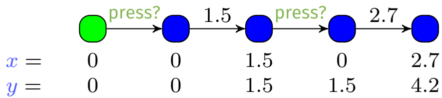




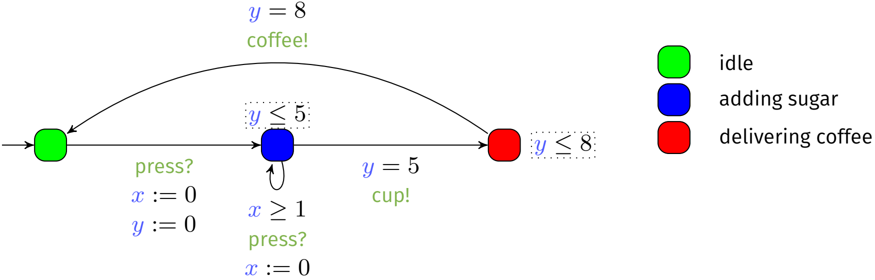
- Example of concrete run for the coffee machine
  - Coffee with no sugar



  - Coffee with 2 doses of sugar



The most critical system: the coffee machine



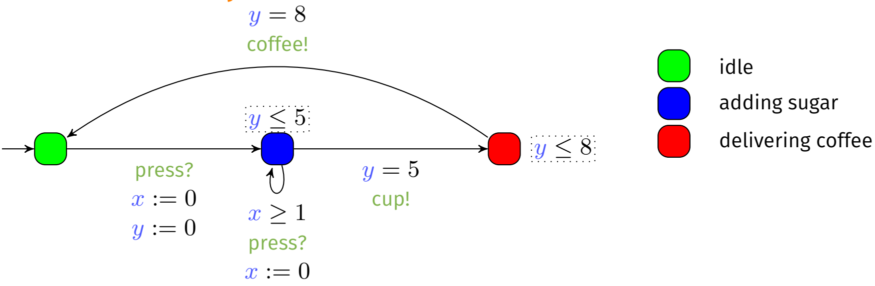
- Example of concrete run for the coffee machine
  - Coffee with no sugar

$x =$	0	0	5	5	8	8
$y =$	0	0	5	5	8	8

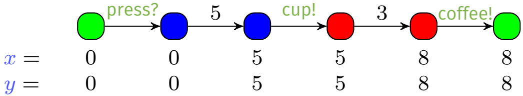
  - Coffee with 2 doses of sugar

$x =$	0	0	1.5	0	2.7	0
$y =$	0	0	1.5	1.5	4.2	4.2

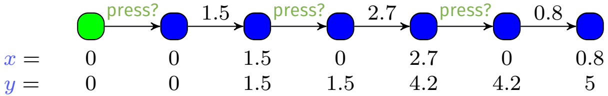
The most critical system: the coffee machine



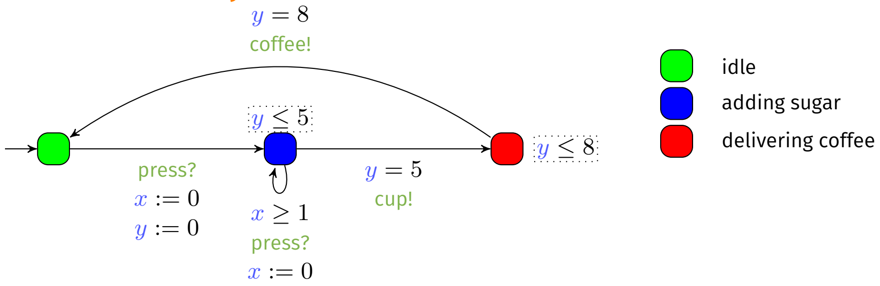
- Example of concrete run for the coffee machine
  - Coffee with no sugar



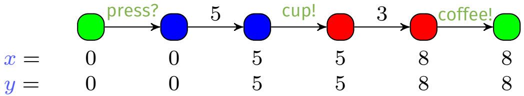
  - Coffee with 2 doses of sugar



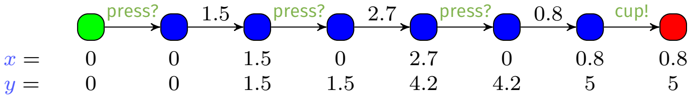
The most critical system: the coffee machine



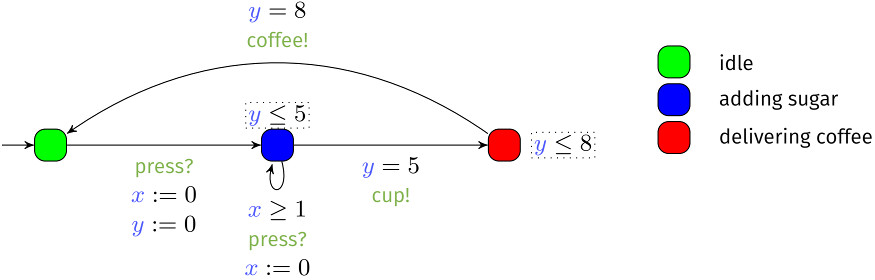
- Example of concrete run for the coffee machine
  - Coffee with no sugar



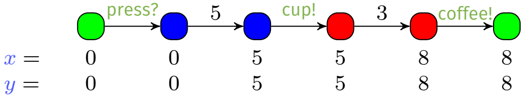
  - Coffee with 2 doses of sugar



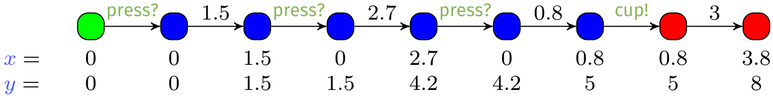
The most critical system: the coffee machine



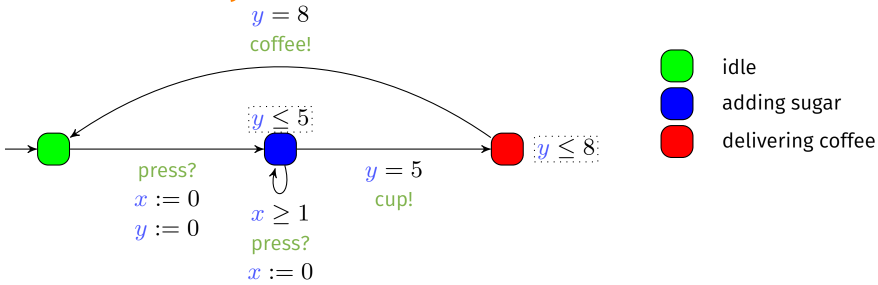
- Example of concrete run for the coffee machine
  - Coffee with no sugar



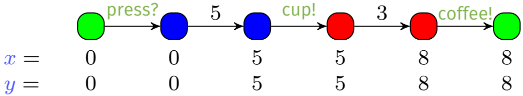
  - Coffee with 2 doses of sugar



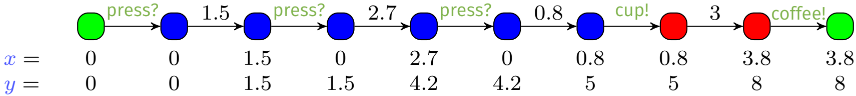
The most critical system: the coffee machine



- Example of concrete run for the coffee machine
  - Coffee with no sugar

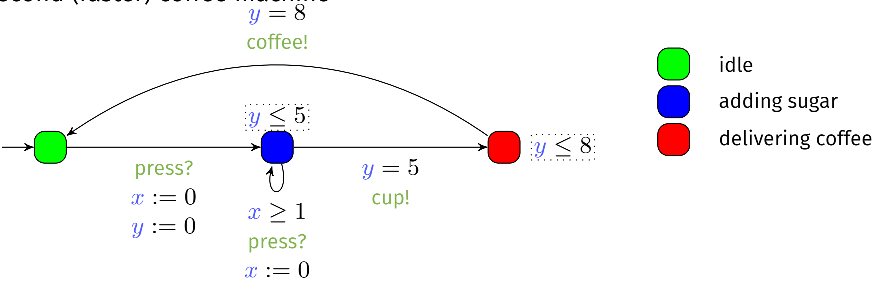


  - Coffee with 2 doses of sugar



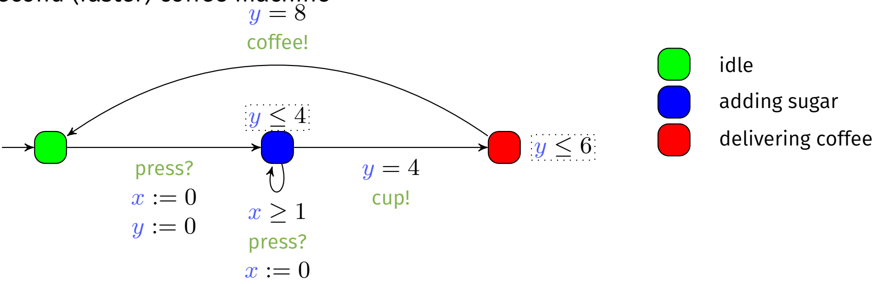
The most critical system: the coffee machine (2/2)

A second (faster) coffee machine



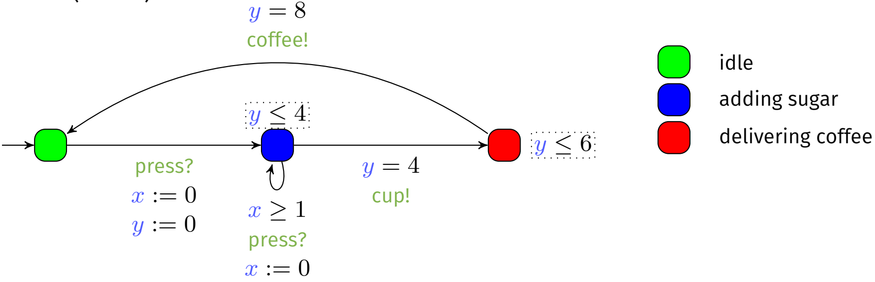
The most critical system: the coffee machine (2/2)

A second (faster) coffee machine



The most critical system: the coffee machine (2/2)

A second (faster) coffee machine



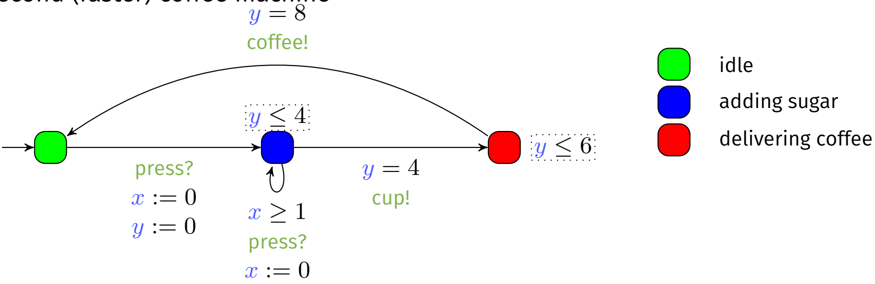
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

$x =$ 0

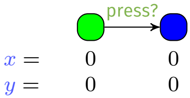
$y =$ 0

The most critical system: the coffee machine (2/2)

A second (faster) coffee machine

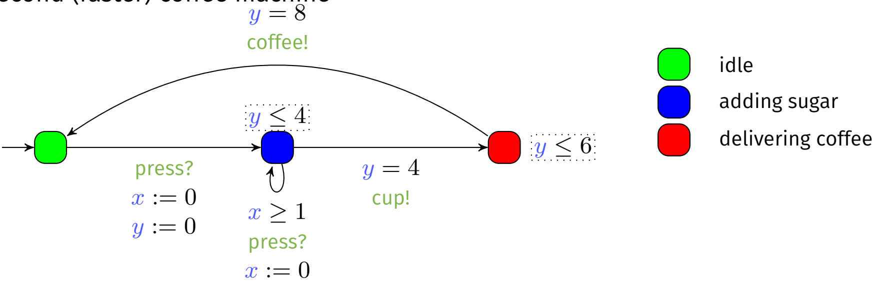


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

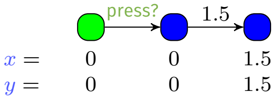


The most critical system: the coffee machine (2/2)

A second (faster) coffee machine

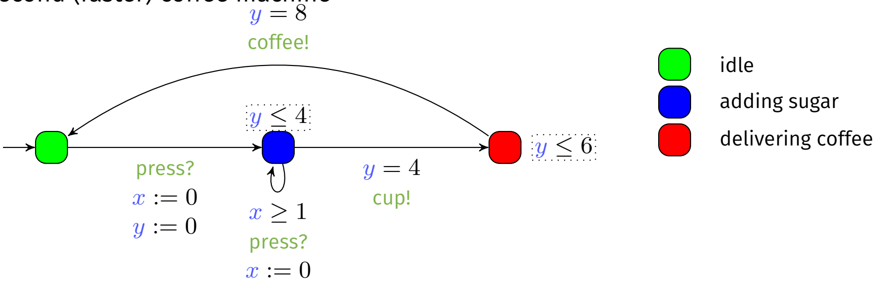


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

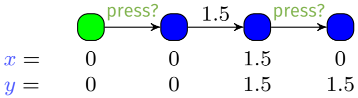


The most critical system: the coffee machine (2/2)

A second (faster) coffee machine

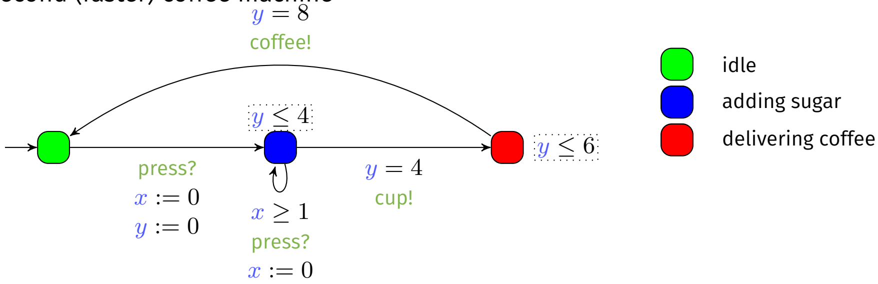


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

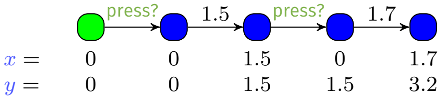


The most critical system: the coffee machine (2/2)

A second (faster) coffee machine

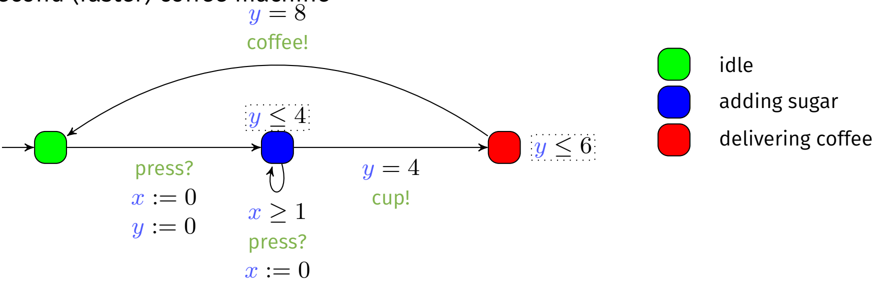


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



The most critical system: the coffee machine (2/2)

A second (faster) coffee machine

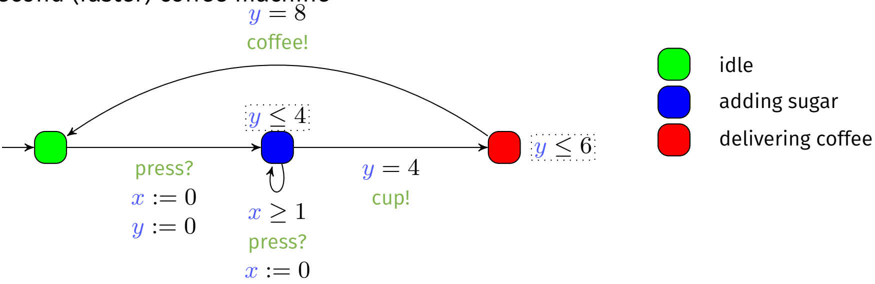


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

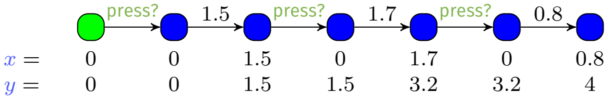
$x =$	0	0	1.5	0	1.7	0
$y =$	0	0	1.5	1.5	3.2	3.2

The most critical system: the coffee machine (2/2)

A second (faster) coffee machine

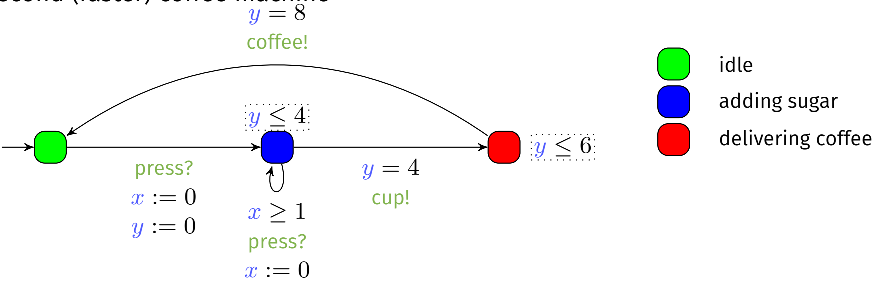


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

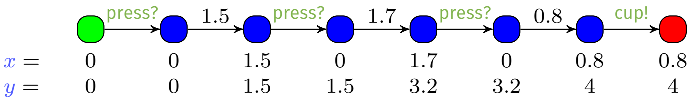


The most critical system: the coffee machine (2/2)

A second (faster) coffee machine

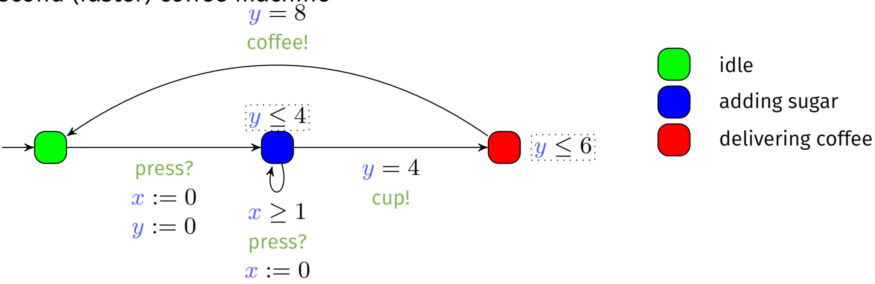


- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

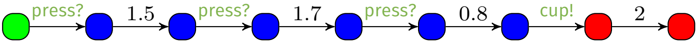


The most critical system: the coffee machine (2/2)

A second (faster) coffee machine



- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



$x =$	0	0	1.5	0	1.7	0	0.8	0.8	2.8
$y =$	0	0	1.5	1.5	3.2	3.2	4	4	6

Decidability of PTAs

Surveying EF-emptiness for PTAs

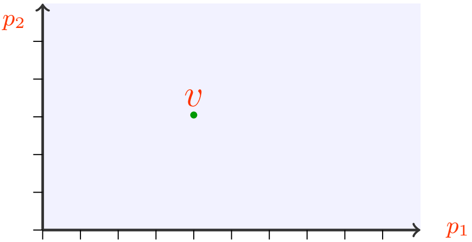
$\mathbb{T}$	$\mathbb{P}$	**Guards**	**Invariants**	**P-clocks**	**NP-clocks**	**Params**	**Decidability**	**Main ref.**
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p \mid d$		1	0	fixed	(at most) PTIME	[Miller, 2000] (consequence)
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p \mid d$		1	0	any	(at most) NP-complete	[Miller, 2000] (consequence)
$\mathbb{N}$	$\mathbb{N}$	$x \leq\geq p \mid d^+$		1	any	any	NEXPTIME-complete	[Bundala and Ouaknine, 2014]
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p \mid d$	$x \preceq p \mid d^+$	1	any	any	(at most) NEXPTIME	[Beneš et al., 2015] (consequence)
$\mathbb{N}$	$\mathbb{N}$	$x \leq\geq p \mid d^+$		2	any	1	PSPACE$^{\text{NEXP}}$-hard	[Bundala and Ouaknine, 2014]
$\mathbb{N}$	$\mathbb{N}$	any		2	any	> 1	open	
$\mathbb{N}$	$\mathbb{N}$	$x \bowtie p \mid d$	None	3	0	1	undecidable	[Beneš et al., 2015]
$\mathbb{N}$	$\mathbb{N}$	$x = p \mid d$	None	3	0	6	undecidable	[Alur et al., 1993]
$\mathbb{N}$	$\mathbb{N}$	$x <> p$		any	any	any	open	
$\mathbb{N}$	$\mathbb{N}$ bounded	$x \bowtie plt$	$x \preceq plt$	any	any	any	(at most) PSPACE-complete	[Jovanović et al., 2015] (consequence)
$\mathbb{R}^+$	$\mathbb{N}$	$x \bowtie p \mid d$		1	0	fixed	(at most) PTIME	[Miller, 2000] (consequence)
$\mathbb{R}^+$	$\mathbb{N}$	$x \bowtie p \mid d$		1	0	any	(at most) NP-complete	[Miller, 2000] (consequence)
$\mathbb{R}^+$	$\mathbb{N}$	$x \bowtie p \mid d$	$x \preceq p \mid d^+$	1	any	any	NEXPTIME	[Beneš et al., 2015]
$\mathbb{R}^+$	$\mathbb{N}$	any		2	any	any	open	
$\mathbb{R}^+$	$\mathbb{N}$	$x \bowtie p \mid d$	None	3	0	1	undecidable	[Beneš et al., 2015]
$\mathbb{R}^+$	$\mathbb{N}$	$x = p \mid d$	None	3	0	6	undecidable	[Alur et al., 1993] (consequence)
$\mathbb{R}^+$	$\mathbb{N}$	$x <> p$		any	any	any	open	
$\mathbb{R}^+$	$\mathbb{N}$ bounded	$x \bowtie plt$	$x \preceq plt$	any	any	any	PSPACE-complete	[Jovanović et al., 2015]
$\mathbb{R}^+$	$\mathbb{Q}^+$	$x \bowtie p \mid d$		1	0	fixed	PTIME	[Miller, 2000]
$\mathbb{R}^+$	$\mathbb{Q}^+$	$x \bowtie p \mid d$		1	0	any	NP-complete	[Miller, 2000]
$\mathbb{R}^+$	$\mathbb{Q}^+$	any		1	1 or 2	any	open	
$\mathbb{R}^+$	$\mathbb{Q}^+[1;2]$	$x \bowtie p \mid d$		1	3	1	undecidable	[Miller, 2000]
$\mathbb{R}^+$	$\mathbb{Q}^+$	any		2	0 or 1	any	open	
$\mathbb{R}^+$	$\mathbb{Q}^+[1;2]$	$x \bowtie p \mid d$		2	2	1	undecidable	[Miller, 2000] (consequence)
$\mathbb{R}^+$	$\mathbb{Q}^+[1;2]$	$x \bowtie p \mid d$		3	0	1	undecidable	[Miller, 2000]
$\mathbb{R}^+$	$\mathbb{R}^+$	$x = p \mid d$	None	3	0	6	undecidable	[Alur et al., 1993]
$\mathbb{R}^+$	$\mathbb{Q}^+$	$x <> p$		< 2	3	2	open	
$\mathbb{R}^+$	$\mathbb{Q}^+$	$x <> p$		2	< 3	2	open	
$\mathbb{R}^+$	$\mathbb{Q}^+$	$x <> p$		2	3	< 2	open	
$\mathbb{Q}^+/\mathbb{R}^+$	$\mathbb{Q}^+/\mathbb{R}^+$	$x <> p$		2	3	2	undecidable	[Doyen, 2007]

PRP in details

Reachability Preservation

Key idea

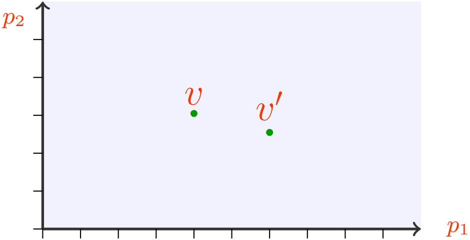
"If we know a parameter valuation v that reaches (resp. does not reach) 🟥, can we find other valuations around v that reach (resp. do not reach) 🟥?"



Reachability Preservation

Key idea

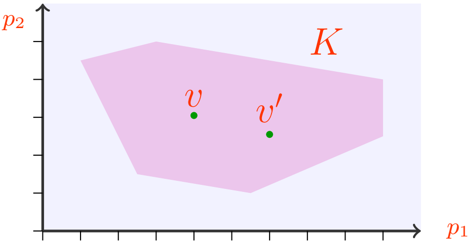
"If we know a parameter valuation v that reaches (resp. does not reach) 🟥, can we find other valuations around v that reach (resp. do not reach) 🟥?"



Reachability Preservation

Key idea

"If we know a parameter valuation v that reaches (resp. does not reach) 🟥, can we find other valuations around v that reach (resp. do not reach) 🟥?"



Reachability Preservation: Undecidability

Problem (PREACH-emptiness)

Let $\mathcal{A}$ *be a PTA, and* v *a parameter valuation. Does there exist* $v' \neq v$ *such that* $v'(\mathcal{A})$ *preserves the reachability of* 🟥 *in* $v(\mathcal{A})$*?*

Reachability Preservation: Undecidability

Problem (PREACH-emptiness)

Let $\mathcal{A}$ be a PTA, and v a parameter valuation. Does there exist $v' \neq v$ such that $v'(\mathcal{A})$ preserves the reachability of in $v(\mathcal{A})$?

Theorem ([André, Lipari, Nguyen, Sun, NFM'15])

PREACH-emptiness is undecidable.

Reachability Preservation: Undecidability

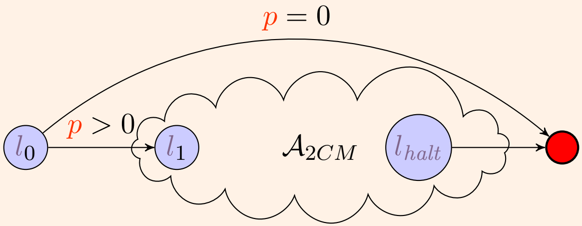
Problem (PREACH-emptiness)

Let $\mathcal{A}$ be a PTA, and v a parameter valuation. Does there exist $v' \neq v$ such that $v'(\mathcal{A})$ preserves the reachability of ● *in $v(\mathcal{A})$?*

Theorem ([André, Lipari, Nguyen, Sun, NFM'15])

PREACH-emptiness is undecidable.

Proof.



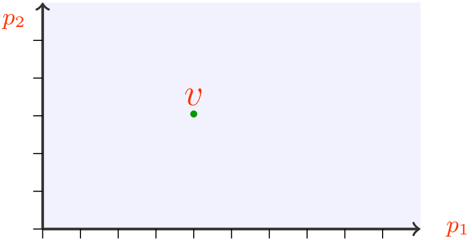
□

PRP: Parametric Reachability Preservation

Input: parameter valuation v

Output: constraint K such that

1. $v \models K$, and
2. $\forall v' \models K$, $v'(\mathcal{A})$ preserves the reachability of ● in $v(\mathcal{A})$



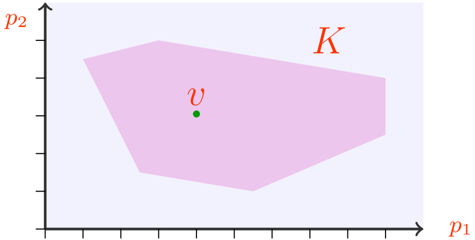
Inspired by EFsynth [Alur et al., 1993, Jovanović et al., 2015] and a variant of IM in [André and Soulat, 2011]

PRP: Parametric Reachability Preservation

Input: parameter valuation v

Output: constraint K such that

1. $v \models K$, and
2. $\forall v' \models K$, $v'(\mathcal{A})$ preserves the reachability of 🟥 in $v(\mathcal{A})$

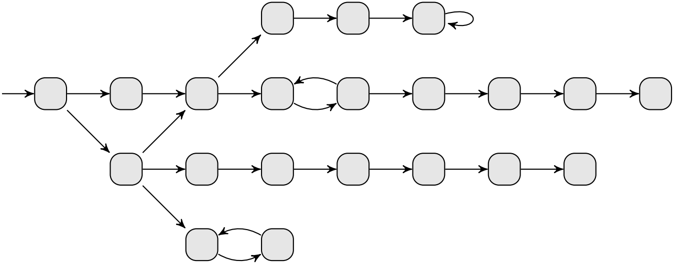


Inspired by EFsynth [Alur et al., 1993, Jovanović et al., 2015] and a variant of IM in [André and Soulat, 2011]

PRP: Case 1

As long as 🟥 is not met…

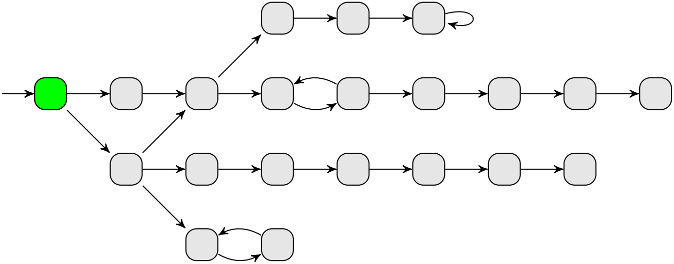
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met...

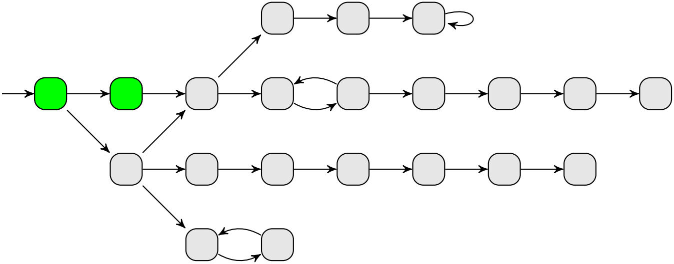
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as [red state] is not met…

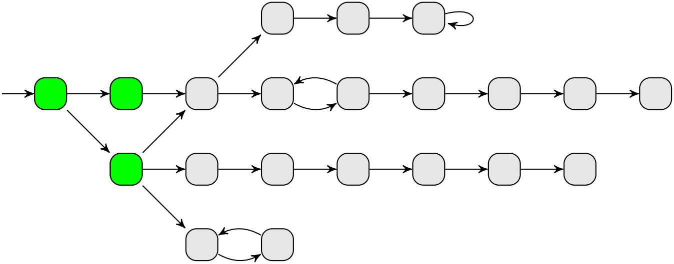
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

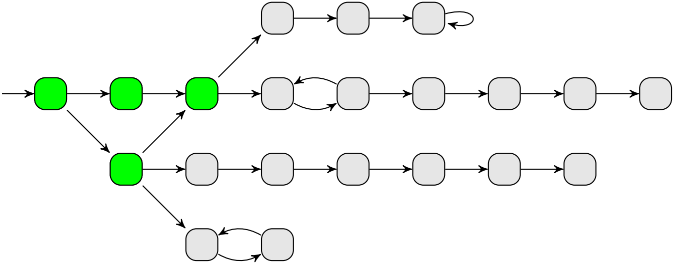
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

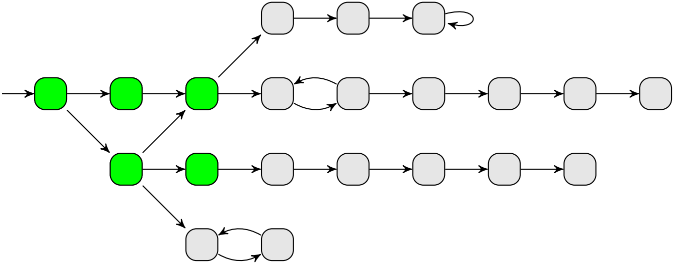
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

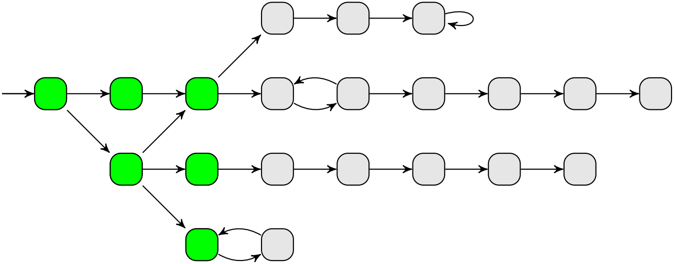
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as [red state] is not met…

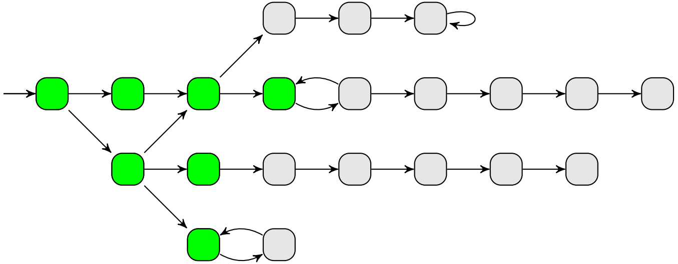
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

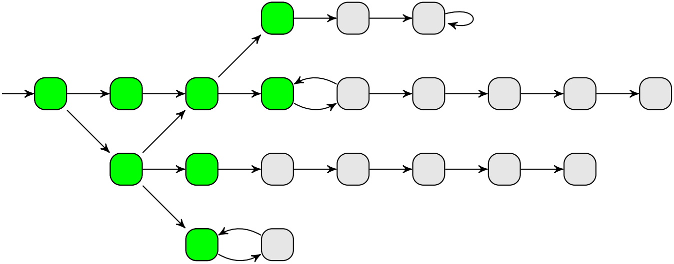
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

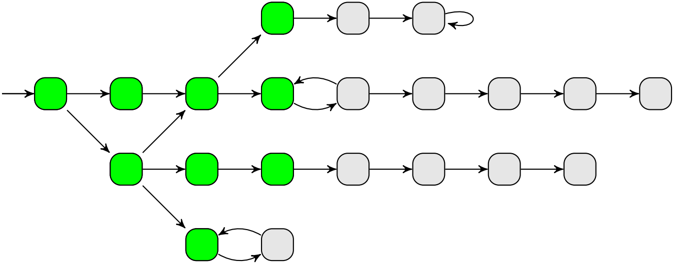
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as [red state] is not met...

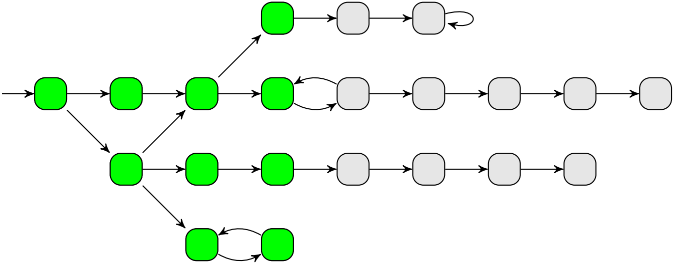
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as [red state] is not met...

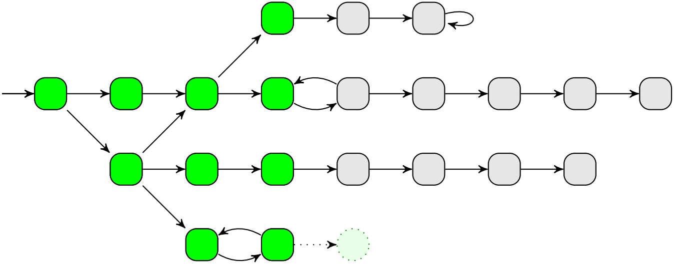
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as [red state] is not met...

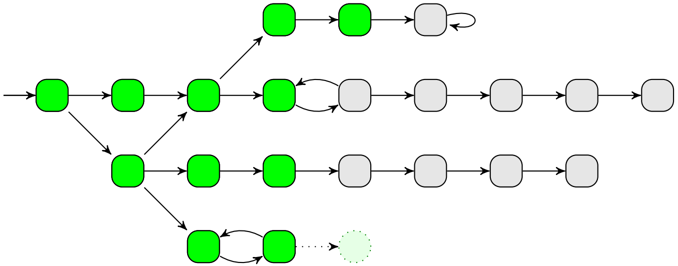
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as [red state] is not met…

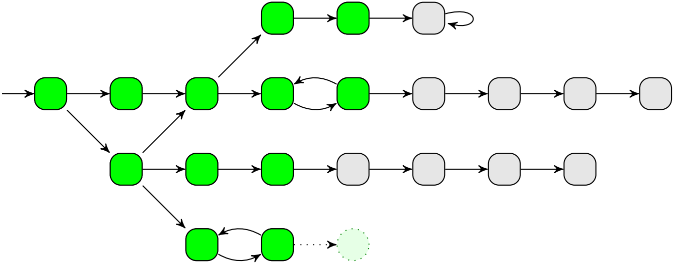
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

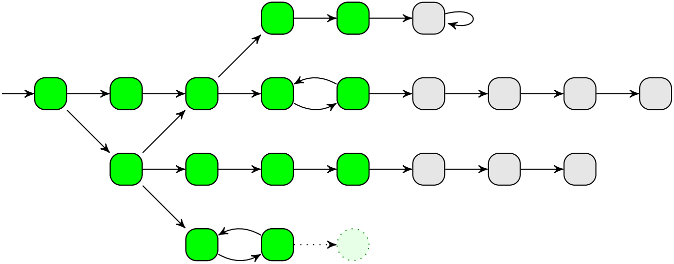
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

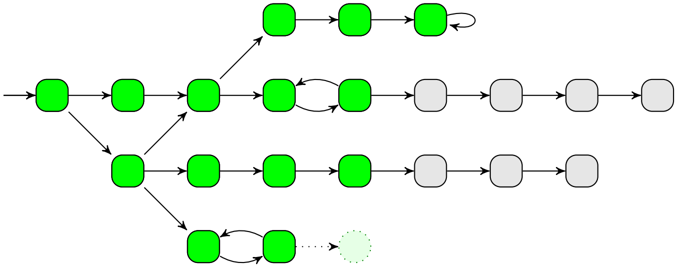
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as ■ is not met...

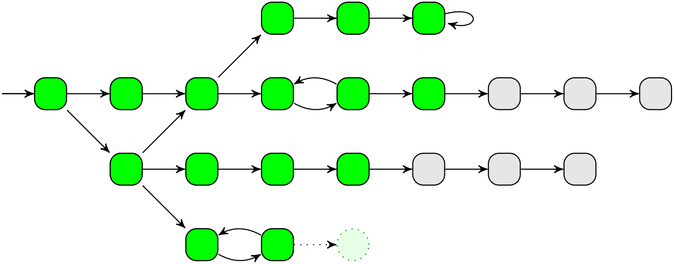
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as 🟥 is not met…

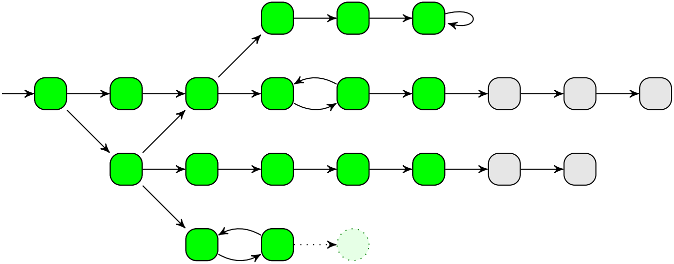
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

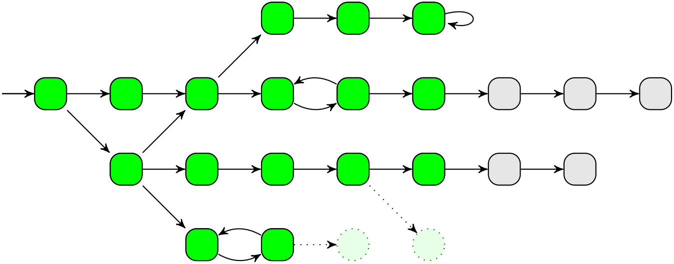
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as  is not met…

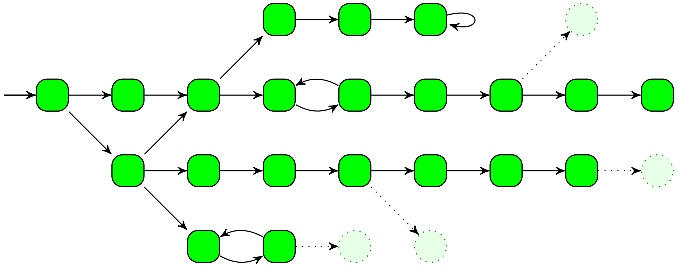
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as [red state] is not met…

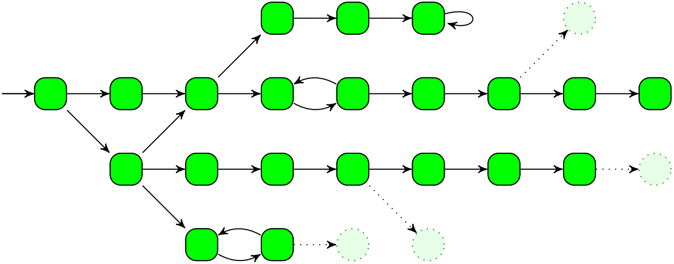
- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



PRP: Case 1

As long as is not met…

- Explore the symbolic state space
- But do not explore the behaviors not present in $v(\mathcal{A})$!



When no successors, and if was never met:

- return $\neg \wedge \cdots \wedge \neg$
- Ensures a subset of the behaviors of $v(\mathcal{A})$, and hence guarantees the unreachability of

PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $v(\mathcal{A})$?
How do we know that a symbolic state of $\mathcal{A}$ corresponds to a behavior of $v(\mathcal{A})$?

PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $v(\mathcal{A})$?
How do we know that a symbolic state of $\mathcal{A}$ corresponds to a behavior of $v(\mathcal{A})$?

We could compute the zone graph of $v(\mathcal{A})$.
But this is not necessary.
In fact, we do not even need to know whether $v(\mathcal{A})$ reaches or not.

PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $v(\mathcal{A})$?
How do we know that a symbolic state of $\mathcal{A}$ corresponds to a behavior of $v(\mathcal{A})$?

We could compute the zone graph of $v(\mathcal{A})$.
But this is not necessary.
In fact, we do not even need to know whether $v(\mathcal{A})$ reaches  or not.

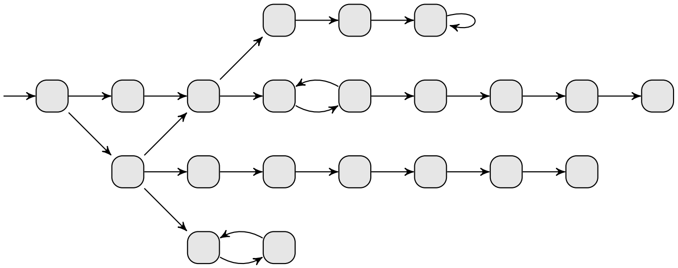
Trick

A symbolic state (l, C) corresponds to a behavior of $v(\mathcal{A})$ iff $v \models C$.

PRP: Case 2

When [red state] is met, switch to an EFsynth-like algorithm…

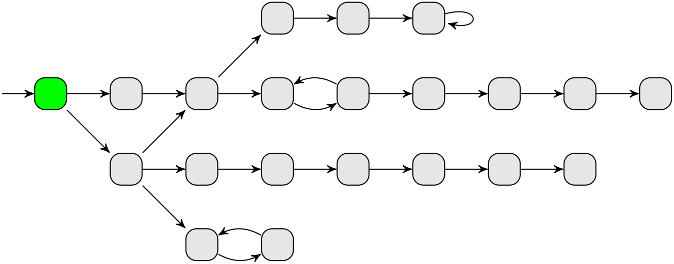
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When 🟥 is met, switch to an EFsynth-like algorithm…

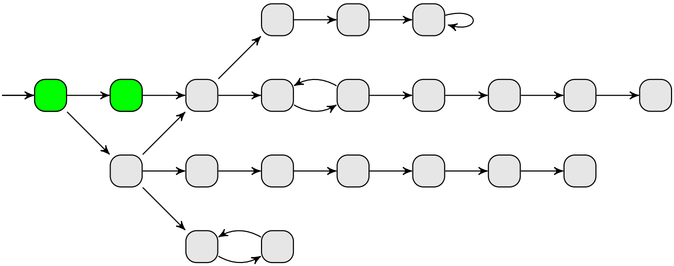
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm. . .

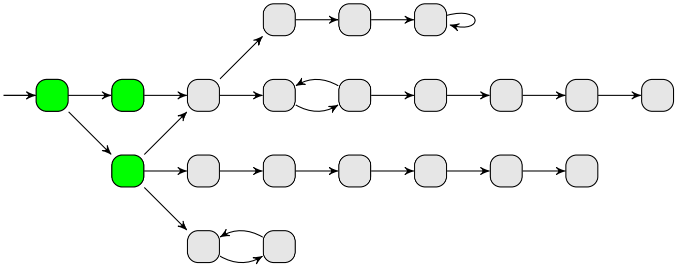
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When is met, switch to an EFsynth-like algorithm…

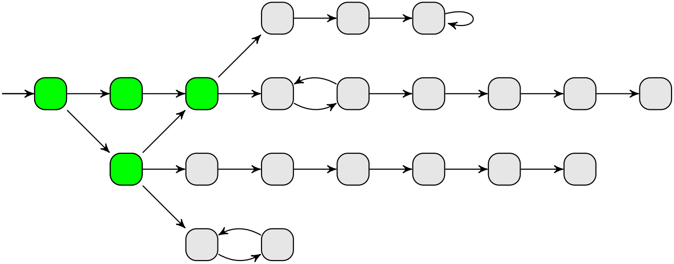
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm...

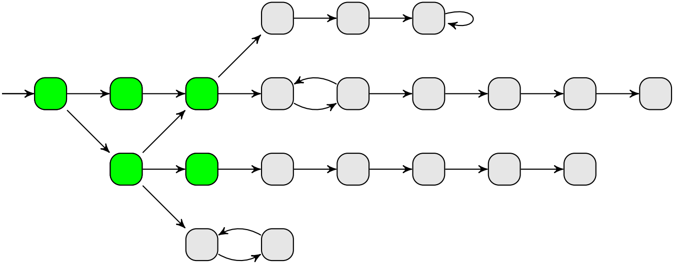
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm...

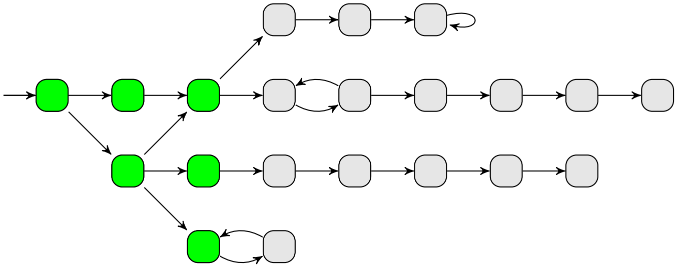
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When [red state] is met, switch to an EFsynth-like algorithm…

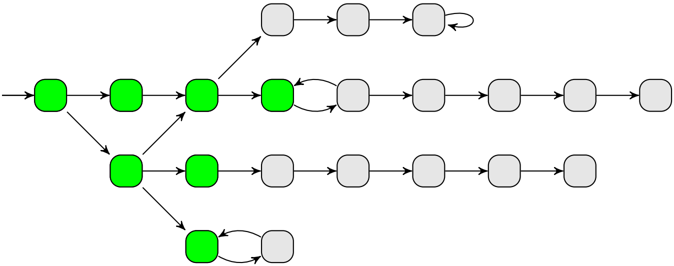
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When [red state] is met, switch to an EFsynth-like algorithm…

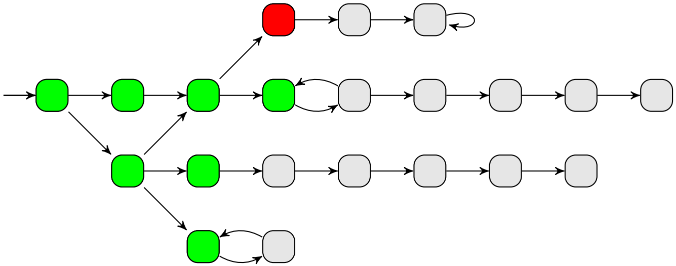
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm…

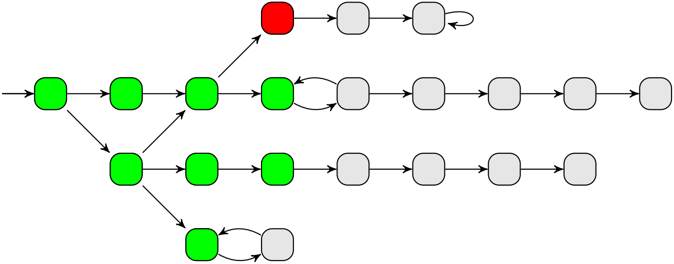
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm...

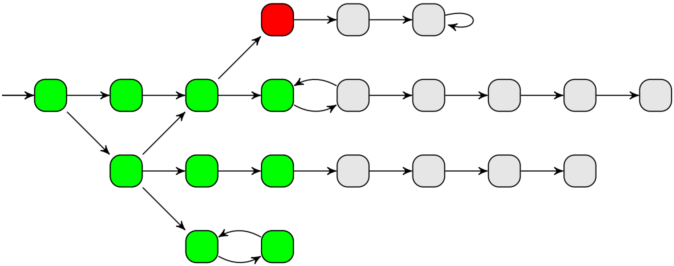
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm…

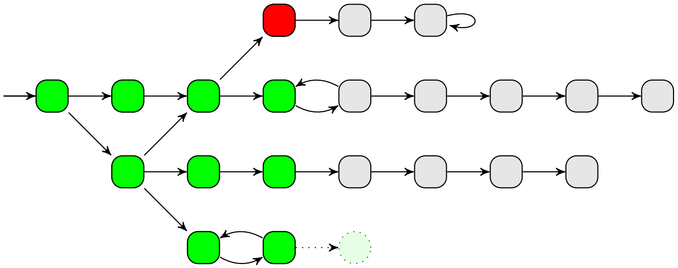
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When [red state] is met, switch to an EFsynth-like algorithm…

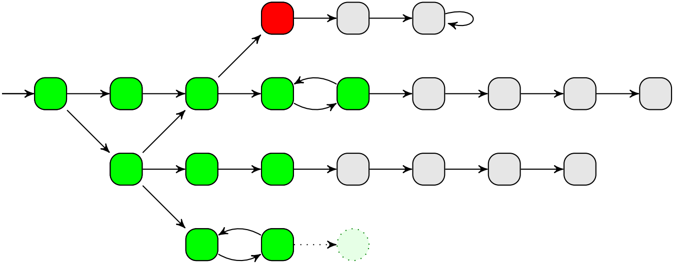
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm...

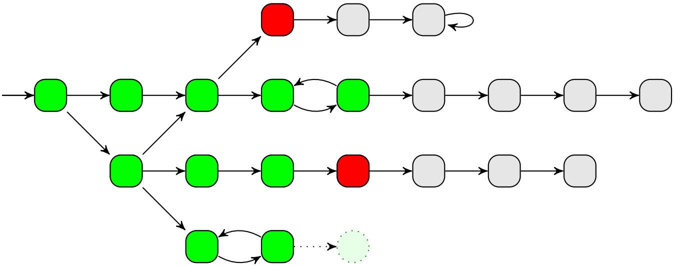
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When 🔴 is met, switch to an EFsynth-like algorithm...

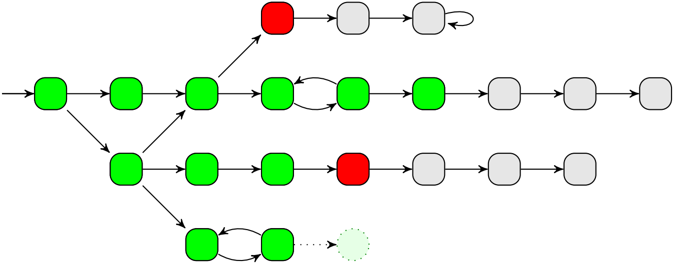
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm…

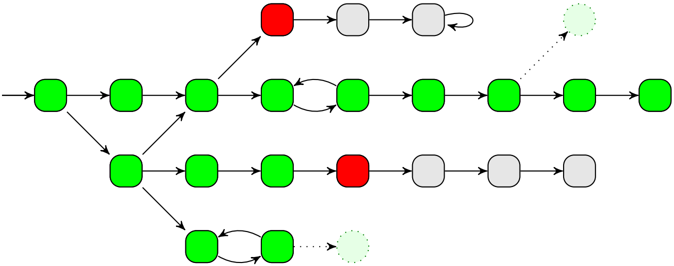
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When  is met, switch to an EFsynth-like algorithm…

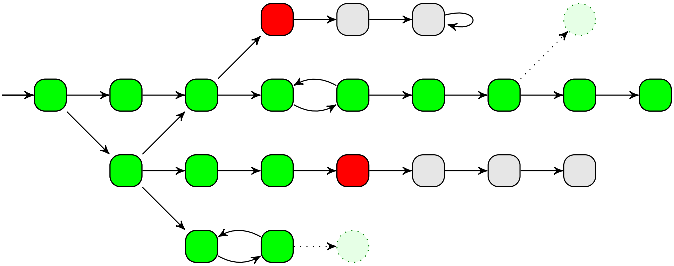
- But still without exploring the behaviors not present in $v(\mathcal{A})$



PRP: Case 2

When ■ is met, switch to an EFsynth-like algorithm. . .

- But still without exploring the behaviors not present in $v(\mathcal{A})$



When no successors, and if ■ was met:

- return ■ $\vee \cdots \vee$ ■
- Guarantees the reachability of ■

Compositional parameter synthesis

Learning an abstraction: $\mathsf{LearnAbstr}(\mathsf{B}, v(\mathsf{A}), \mathsf{AG}\neg L^{☹})$

Input: $v(\mathsf{A}) \parallel \mathsf{B}$
Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example

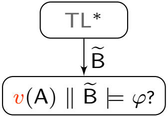
TL*

TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B
[Lin et al., 2014]

Learning an abstraction: $\mathsf{LearnAbstr}(\mathsf{B}, v(\mathsf{A}), \mathsf{AG}\neg L^{☹})$

Input: $v(\mathsf{A}) \parallel \mathsf{B}$

Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B

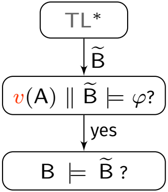
[Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr$(B, v(A), AG\neg L^{☹})$

Input: $v(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B

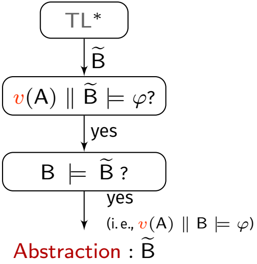
[Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr(B, v(A), AG$\neg L^{☹}$)

Input: v(A) $\parallel$ B

Output: an abstraction $\widetilde{\text{B}}$ or a counter-example



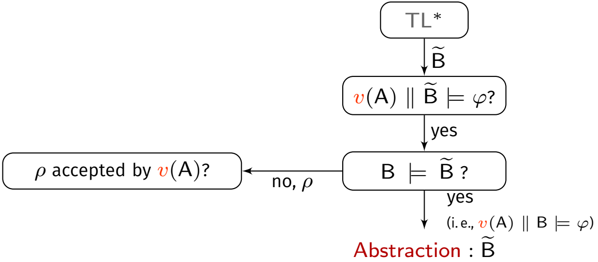
TL*: learning algorithm to compute a candidate abstraction $\widetilde{\text{B}}$ of an ERA B [Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr$(\mathsf{B}, v(\mathsf{A}), \mathsf{AG}\neg L^{☹})$

Input: $v(\mathsf{A}) \parallel \mathsf{B}$

Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B

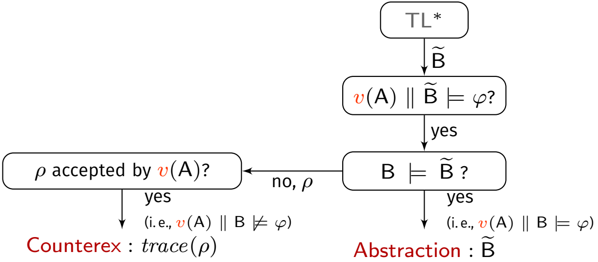
[Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: LearnAbstr$(\mathsf{B}, v(\mathsf{A}), \mathsf{AG}\neg L^{☹})$

Input: $v(\mathsf{A}) \parallel \mathsf{B}$

Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B

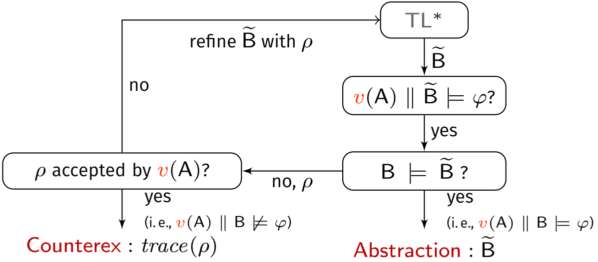
[Lin et al., 2014]

$\models$: can be checked using model checking

Learning an abstraction: $\mathsf{LearnAbstr}(B, v(A), \mathsf{AG}\neg L^{☹})$

Input: $v(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B

[Lin et al., 2014]

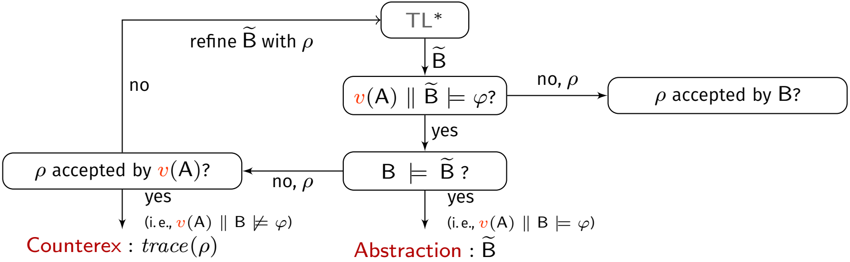
$\models$: can be checked using model checking

Refinement: can be performed using learning

Learning an abstraction: LearnAbstr(B, v(A), AG¬$L^{☹}$)

Input: v(A) $\parallel$ B

Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B

[Lin et al., 2014]

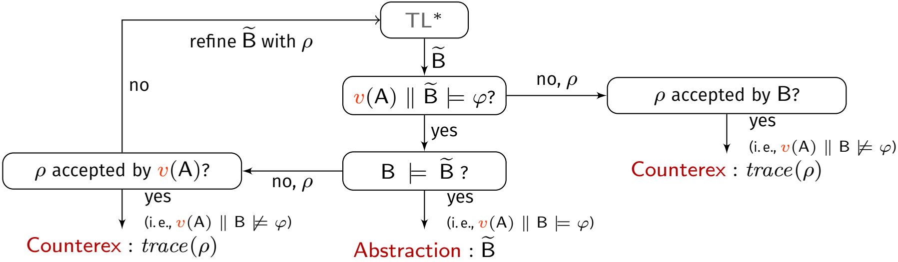
$\models$: can be checked using model checking

Refinement: can be performed using learning

Learning an abstraction: LearnAbstr$(\mathsf{B}, v(\mathsf{A}), \mathsf{AG}\neg L^{☹})$

Input: $v(\mathsf{A}) \parallel \mathsf{B}$

Output: an abstraction $\widetilde{\mathsf{B}}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{\mathsf{B}}$ of an ERA B

[Lin et al., 2014]

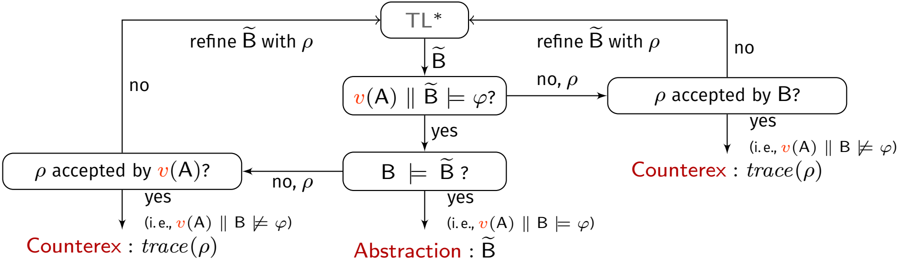
$\models$: can be checked using model checking

Refinement: can be performed using learning

Learning an abstraction: LearnAbstr$(B, v(A), AG\neg L^{☹})$

Input: $v(A) \parallel B$

Output: an abstraction $\widetilde{B}$ or a counter-example



TL*: learning algorithm to compute a candidate abstraction $\widetilde{B}$ of an ERA B

[Lin et al., 2014]

$\models$: can be checked using model checking

Refinement: can be performed using learning

Replaying a trace

Given a finite trace (i. e., a sequence of actions), we can replay it in the parametric framework

- i. e., find all parameter valuations for which this trace is feasible
- Using a symbolic semantics defined for PERAs (see paper)

☺ Very cheap

Our overall procedure CompSynth

Key ideas:

- Iterate on integer points v
- Try to compute an abstraction $\widetilde{\mathsf{B}}$ of the non-parametric component w.r.t. $v(\mathsf{A})$ and φ
  - If succeed, synthesize "similar" valuations using PRP on $\mathsf{A} \parallel \widetilde{\mathsf{B}}$
  - If fail, synthesize the valuations corresponding to the counterex.

Our overall procedure CompSynth

Key ideas:

- Iterate on integer points v
- Try to compute an abstraction $\widetilde{\mathsf{B}}$ of the non-parametric component w.r.t. $v(\mathsf{A})$ and φ
  - If succeed, synthesize "similar" valuations using PRP on $\mathsf{A} \parallel \widetilde{\mathsf{B}}$
  - If fail, synthesize the valuations corresponding to the counterex.

$K_{bad} \leftarrow \bot$; $K_{good} \leftarrow \bot$

while *there is an integer point not covered by* K_{bad} *or* K_{good} **do**

  Pick such a point v

  switch LearnAbstr$(\mathsf{B}, v(\mathsf{A}), AG\neg L^{☹})$ **do**

    case Abstraction$(\widetilde{\mathsf{B}})$ **do**

      $K_{good} \leftarrow K_{good} \cup \mathrm{PRP}(\mathsf{A} \parallel \widetilde{\mathsf{B}}, v, L^{☹})$

    case Counterex(τ) **do**

      $K_{bad} \leftarrow K_{bad} \cup \mathrm{ReplayTrace}(\mathsf{A} \parallel \mathsf{B}, \tau)$

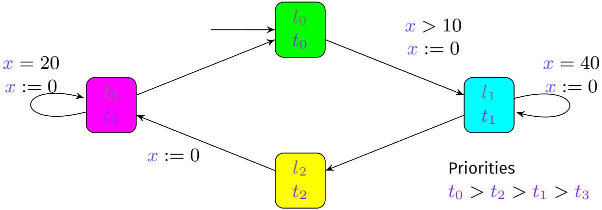
return (K_{good}, K_{bad})

Parametric task automata

A unified formalism: Parametric task automata

Extension of task automata [Norström et al., 1999, Fersman et al., 2007] with parameters

[André, FMICS'17]

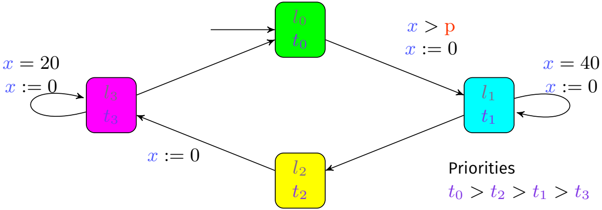


Task	B	W	D
t_0	0	1	2
t_1	4	4	20
t_2	0	1	4
t_3	2	2	10

A unified formalism: Parametric task automata

Extension of task automata [Norström et al., 1999, Fersman et al., 2007] with parameters

[André, FMICS'17]

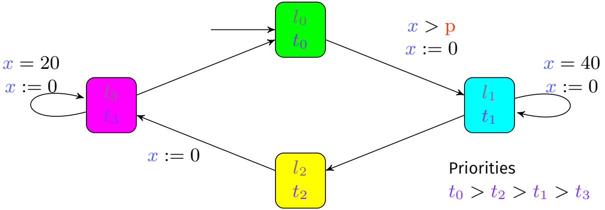


Task	B	W	D
t_0	0	1	2
t_1	4	4	20
t_2	0	1	p'
t_3	2	2	10

A unified formalism: Parametric task automata

Extension of task automata [Norström et al., 1999, Fersman et al., 2007] with parameters

[André, FMICS'17]



Task	B	W	D
t_0	0	1	2
t_1	4	4	20
t_2	0	1	p'
t_3	2	2	10

Parametric task automata can model

- Preemption
- Periodic tasks, sporadic tasks, pseudo-periodic tasks...
- Dependencies between tasks
- Offset, jitter
- Uncertainty
- Uniprocessor only

Parametric task automata: theory and practice

Schedulability-emptiness ("is the set of valuations for which the system is schedulable empty?")

- Undecidable in general
- Decidable under some assumptions **[André, FMICS'17]**

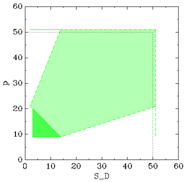
Parametric task automata: theory and practice

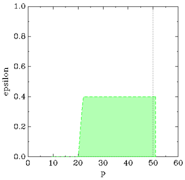
Schedulability-emptiness ("is the set of valuations for which the system is schedulable empty?")

- Undecidable in general
- Decidable under some assumptions **[André, FMICS'17]**

Implementation in IMITATOR

- Translation into a network of parametric stopwatch automata
- Schedulability analysis
- Parametric and/or robust schedulability analysis





The FMTV Challenge in details

To build the PTA model

- Uncertainties in the system:
  - $P1 \in [40 - 0.004, 40 + 0.004]$
  - $P3 \in [\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}]$
  - $P4 \in [40 - 0.004, 40 + 0.004]$

To build the PTA model

- Uncertainties in the system:
  - $P1 \in [40 - 0.004, 40 + 0.004]$
  - $P3 \in [\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}]$
  - $P4 \in [40 - 0.004, 40 + 0.004]$
- Parameters:
  - P1_uncertain
  - P3_uncertain
  - P4_uncertain

To build the PTA model

- Uncertainties in the system:
  - $P1 \in [40 - 0.004, 40 + 0.004]$
  - $P3 \in [\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}]$
  - $P4 \in [40 - 0.004, 40 + 0.004]$

- Parameters:
  - P1_uncertain
  - P3_uncertain
  - P4_uncertain

- The end-to-end latency (another parameter): E2E

To build the PTA model

- Uncertainties in the system:
  - $P1 \in [40 - 0.004, 40 + 0.004]$
  - $P3 \in [\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}]$
  - $P4 \in [40 - 0.004, 40 + 0.004]$
- Parameters:
  - P1_uncertain
  - P3_uncertain
  - P4_uncertain
- The end-to-end latency (another parameter): E2E
- Others:
  - the register between task 2 and task 3: discrete variable $\mathrm{reg}_{2,3}$
  - the buffer between task 3 and task 4: $n = 1$ or $n = 3$

Simplification

- T1 and T2 are synchronised; T1, T3 and T4 are asynchronised
  - (exact modeling of the system behaviour is too heavy)

Simplification

- T1 and T2 are synchronised; T1, T3 and T4 are asynchronised
  - (exact modeling of the system behaviour is too heavy)
- We choose a single arbitrary frame, called the target one
- We assume the system is initially in an arbitrary status
  - This is our only uncertain assumption (in other words, can the periods deviate from each other so as to yield any arbitrary deviation?)

Outline

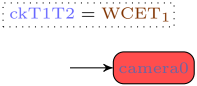
1 Decidability

2 Efficient synthesis

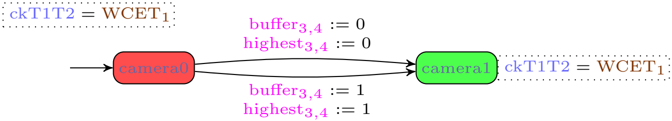
3 Applications to schedulability analysis

4 Perspectives
- The PTA model for $n = 1$

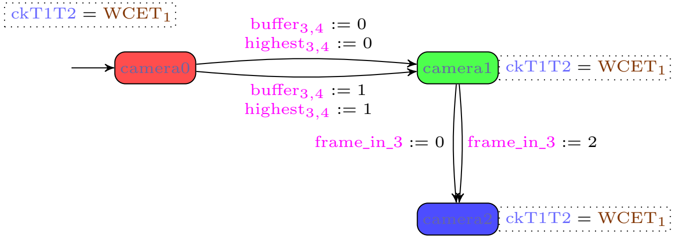
The initialization automaton



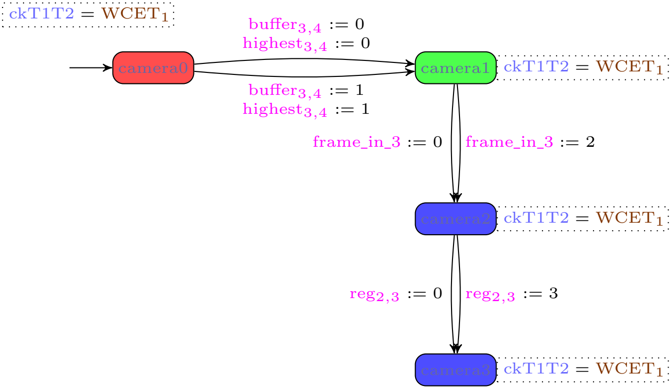
The initialization automaton



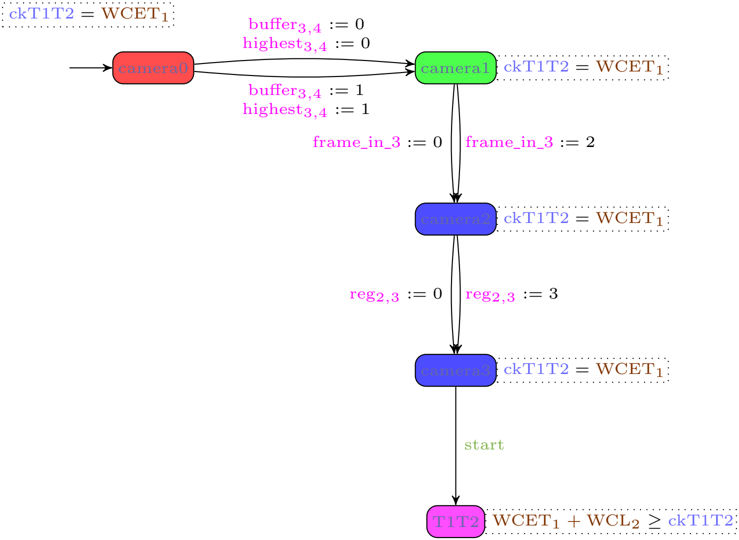
The initialization automaton



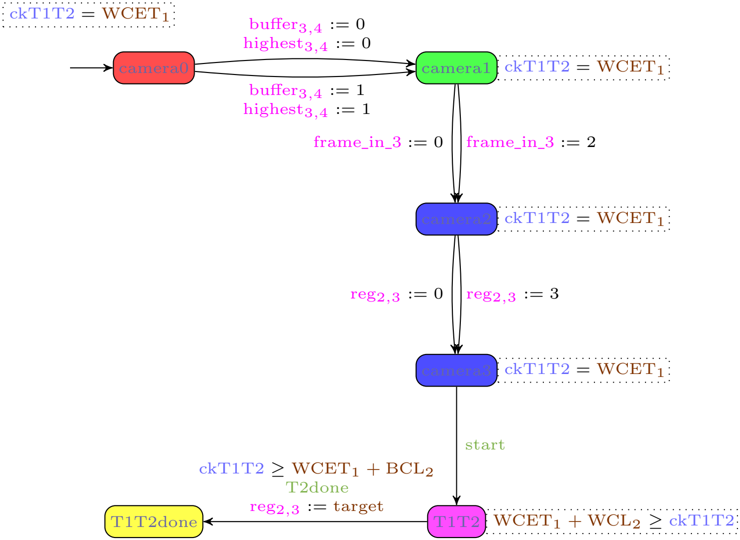
The initialization automaton



The initialization automaton

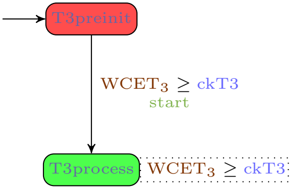


The initialization automaton

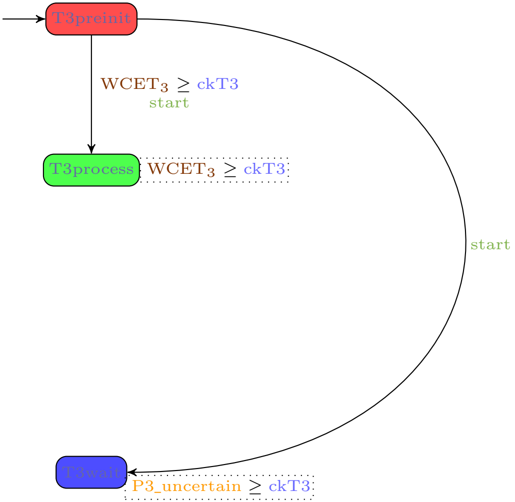


Task T3

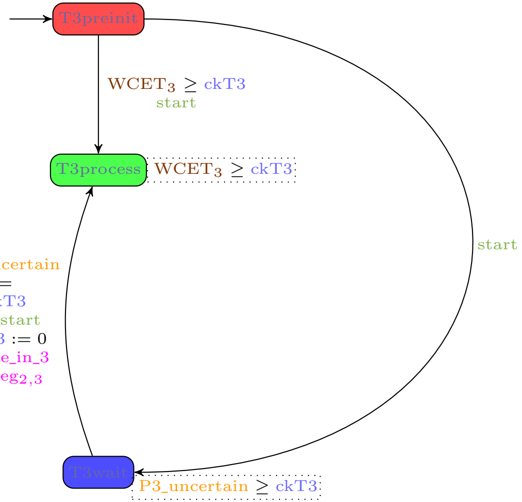
Task T3



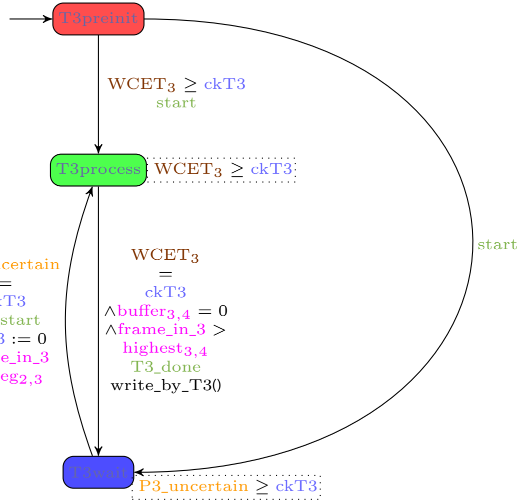
Task T3


T3preinit
WCET₃ ≥ ckT3
start
T3process
WCET₃ ≥ ckT3
start
T3wait
P3_uncertain ≥ ckT3

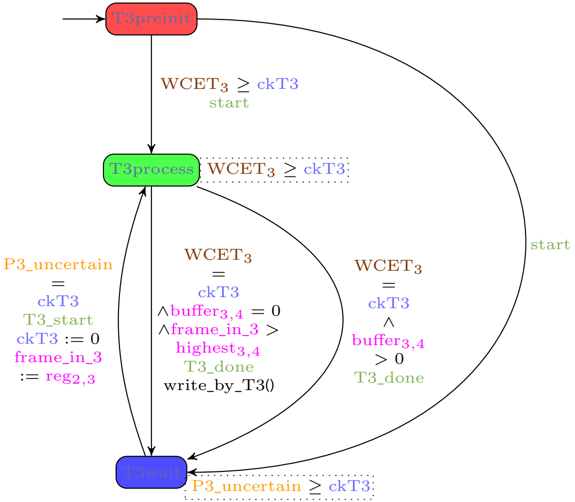
Task T3


T3preinit
WCET$_3$ ≥ ckT3
start
T3process
WCET$_3$ ≥ ckT3
start
P3_uncertain
=
ckT3
T3_start
ckT3 := 0
frame_in_3
:= reg$_{2,3}$
T3wait
P3_uncertain ≥ ckT3

Task T3

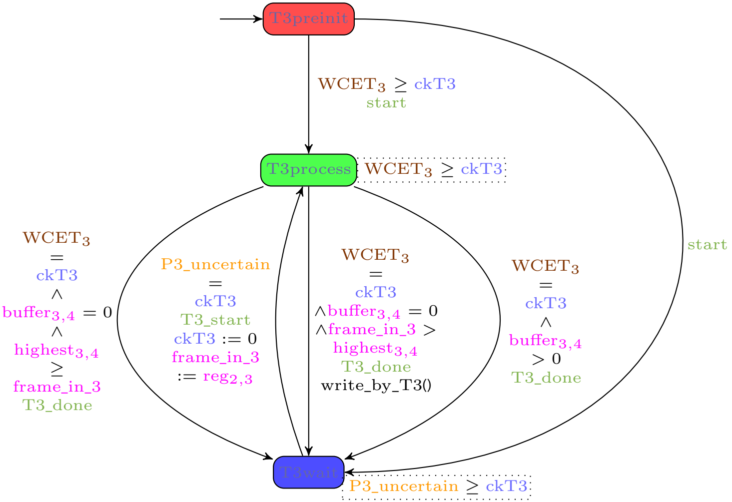

T3preinit
WCET3 ≥ ckT3
start
T3process
WCET3 ≥ ckT3
start
P3_uncertain
=
ckT3
T3_start
ckT3 := 0
frame_in_3
:= reg2,3
WCET3
=
ckT3
∧buffer3,4 = 0
∧frame_in_3 >
highest3,4
T3_done
write_by_T3()
T3wait
P3_uncertain ≥ ckT3

Task T3



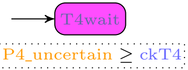
T3preinit
WCET$_3$ ≥ ckT3
start
T3process
WCET$_3$ ≥ ckT3
start
P3_uncertain = ckT3
T3_start
ckT3 := 0
frame_in_3 := reg$_{2,3}$
WCET$_3$ = ckT3 ∧buffer$_{3,4}$ = 0 ∧frame_in_3 > highest$_{3,4}$
T3_done
write_by_T3()
WCET$_3$ = ckT3 ∧ buffer$_{3,4}$ > 0
T3_done
T3wait
P3_uncertain ≥ ckT3

Task T3

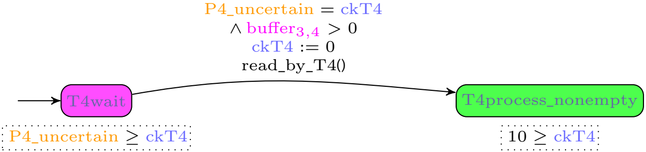


T3preinit
WCET3 ≥ ckT3
start
T3process
WCET3 ≥ ckT3
start
WCET3 = ckT3 ∧ buffer3,4 = 0 ∧ highest3,4 ≥ frame_in_3
T3_done
P3_uncertain = ckT3
T3_start
ckT3 := 0
frame_in_3 := reg2,3
WCET3 = ckT3 ∧buffer3,4 = 0 ∧frame_in_3 > highest3,4
T3_done
write_by_T3()
WCET3 = ckT3 ∧ buffer3,4 > 0
T3_done
T3wait
P3_uncertain ≥ ckT3

Task T4

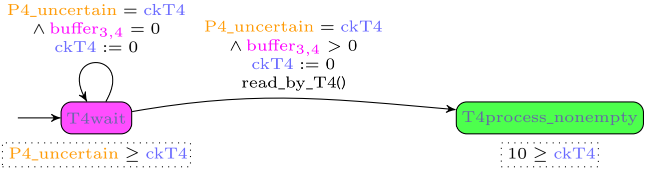


Task T4



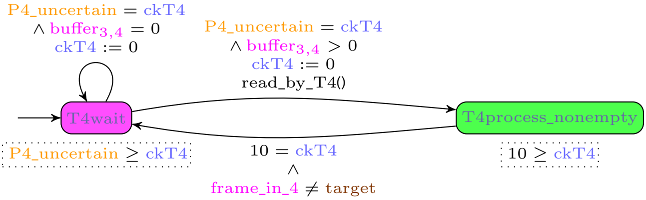
P4_uncertain = ckT4
∧ buffer3,4 > 0
ckT4 := 0
read_by_T4()
T4wait
P4_uncertain ≥ ckT4
T4process_nonempty
10 ≥ ckT4

Task T4

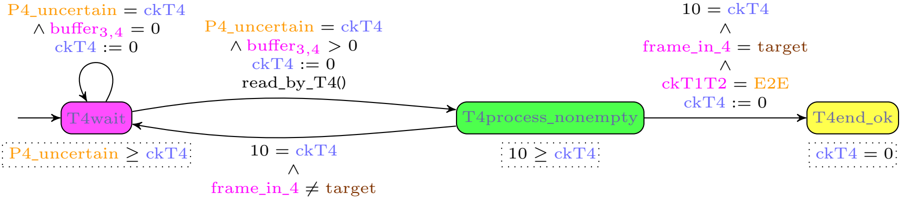


P4_uncertain = ckT4
∧ buffer3,4 = 0
ckT4 := 0
P4_uncertain = ckT4
∧ buffer3,4 > 0
ckT4 := 0
read_by_T4()
T4wait
T4process_nonempty
P4_uncertain ≥ ckT4
10 ≥ ckT4

Task T4

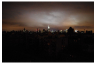

P4_uncertain = ckT4
∧ buffer3,4 = 0
ckT4 := 0
P4_uncertain = ckT4
∧ buffer3,4 > 0
ckT4 := 0
read_by_T4()
T4wait
T4process_nonempty
P4_uncertain ≥ ckT4
10 = ckT4
∧
frame_in_4 ≠ target
10 ≥ ckT4

Task T4


P4_uncertain = ckT4
∧ buffer3,4 = 0
ckT4 := 0
P4_uncertain = ckT4
∧ buffer3,4 > 0
ckT4 := 0
read_by_T4()
10 = ckT4
∧
frame_in_4 = target
∧
ckT1T2 = E2E
ckT4 := 0
T4wait
T4process_nonempty
T4end_ok
P4_uncertain ≥ ckT4
10 = ckT4
∧
frame_in_4 ≠ target
10 ≥ ckT4
ckT4 = 0

Licensing

Source of the graphics used I



Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: `https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.JPG`
License: CC BY 3.0

Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: `https://secure.flickr.com/photos/ideum/4711481781/`
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: `https://secure.flickr.com/photos/imcomkorea/3017886760/`
License: CC BY-NC-ND 2.0

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

Source of the graphics used II

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