NASA Formal Methods 2015

April 27th, 2015
Pasadena, CA, USA

Reachability Preservation Based Parameter Synthesis for Timed Automata

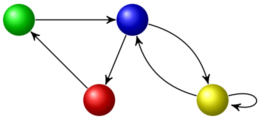
Étienne André[1], Giuseppe Lipari[2], Hoang Gia Nguyen[1], Youcheng Sun[3]

[1]LIPN, Université Paris 13, Sorbonne Paris Cité, CNRS, France
[2]CRIStAL – UMR 9189, Université de Lille, USR 3380 CNRS, France
[3]Scuola Superiore Sant'Anna, Pisa, Italy

PROJET FINANCÉ PAR L'ANR - PROJECT FUNDED BY THE ANR

Context: Formal Verification of Timed Systems

- Model checking



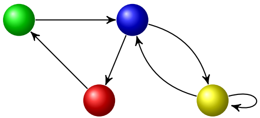
A model of the system

 is unreachable

A property to be satisfied

Context: Formal Verification of Timed Systems

- Model checking



$\models^{?}$

 is unreachable

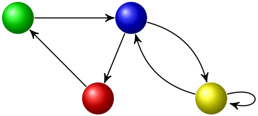
A model of the system

A property to be satisfied

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

Context: Formal Verification of Timed Systems

- 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

Beyond Model Checking: Parameter Synthesis

- Timed systems are characterized by a set of timing constants
  - "The packet transmission lasts for $50\,ms$"
  - "The sensor reads the value every $10\,s$"
- Verification for one set of constants does not usually guarantee the correctness for other values
- Challenges
  - Numerous verifications: is the system correct for any value within $[40; 60]$?
  - Optimization: until what value can we increase 10?
  - Robustness [Markey, 2011]: What happens if 50 is implemented with 49.99?

Beyond Model Checking: Parameter Synthesis

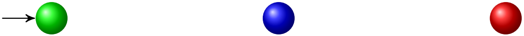
- Timed systems are characterized by a set of timing constants
  - "The packet transmission lasts for 50 ms"
  - "The sensor reads the value every 10 s"
- Verification for one set of constants does not usually guarantee the correctness for other values
- Challenges
  - Numerous verifications: is the system correct for any value within $[40; 60]$?
  - Optimization: until what value can we increase 10?
  - Robustness [Markey, 2011]: What happens if 50 is implemented with 49.99?
- Parameter synthesis
  - Consider that timing constants are unknown constants (parameters)
  - Find good values for the parameters

Outline

Outline

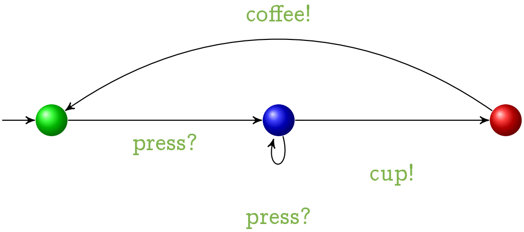
Timed Automaton (TA)

- Finite state automaton (sets of locations)



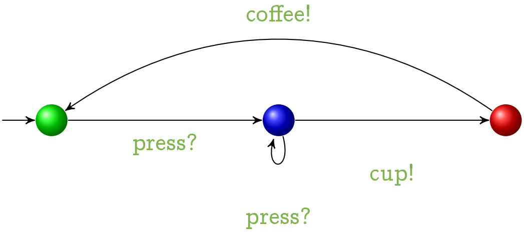
Timed Automaton (TA)

- Finite state automaton (sets of locations and actions)



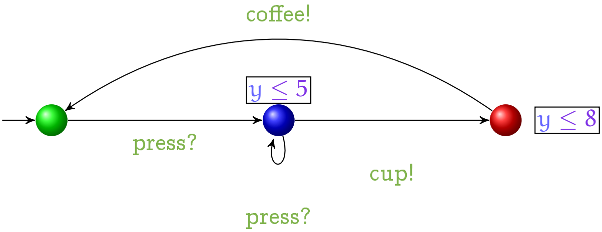
Timed Automaton (TA)

- 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)

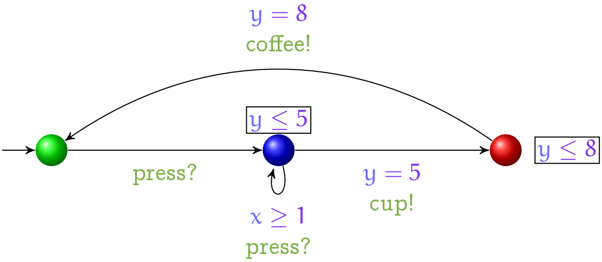
- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
- Features
  - Location invariant: property to be verified to stay at a location



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

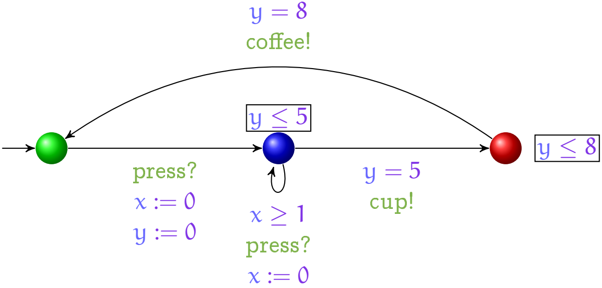
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition



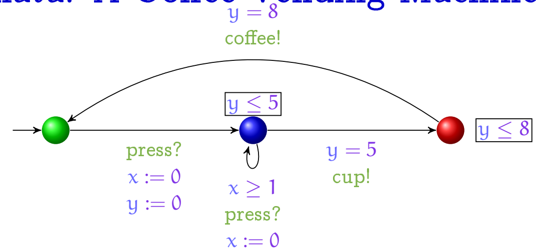
Timed Automaton (TA)

- 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

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

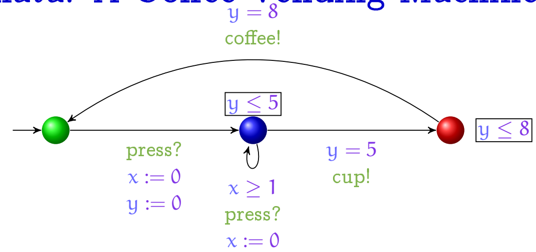


Timed Automata: A Coffee Vending Machine

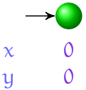


- Examples of concrete runs

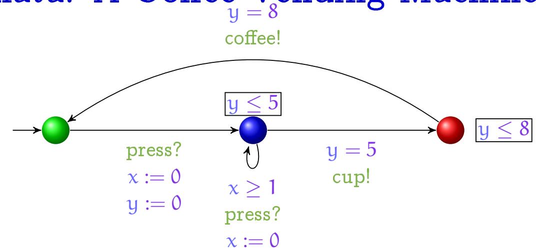
Timed Automata: A Coffee Vending Machine



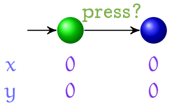
- Examples of concrete runs
  - Coffee with no sugar



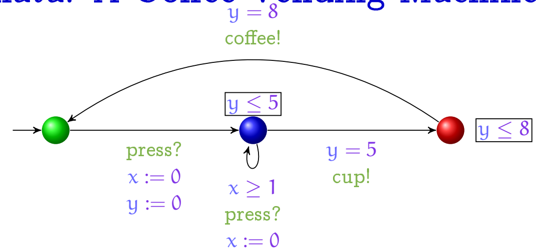
Timed Automata: A Coffee Vending Machine



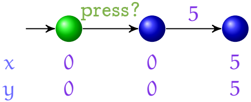
- Examples of concrete runs
  - Coffee with no sugar



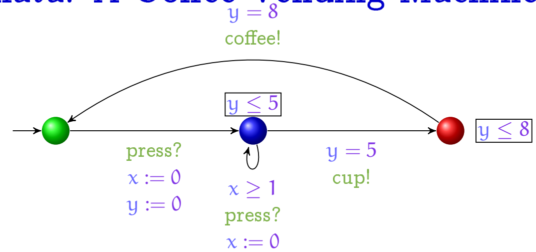
Timed Automata: A Coffee Vending Machine



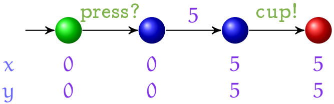
- Examples of concrete runs
  - Coffee with no sugar



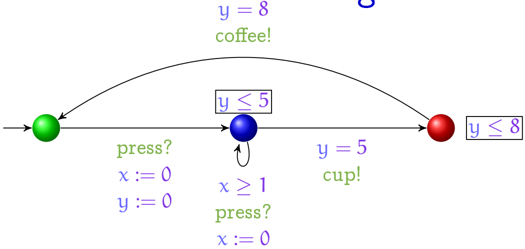
Timed Automata: A Coffee Vending Machine



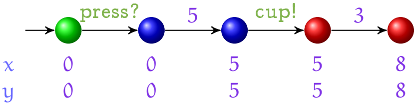
- Examples of concrete runs
  - Coffee with no sugar



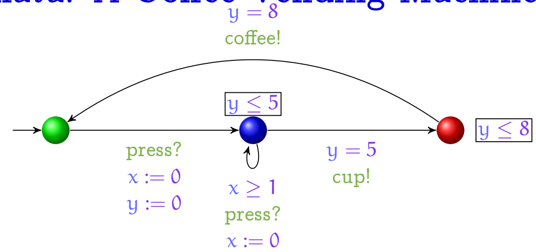
Timed Automata: A Coffee Vending Machine



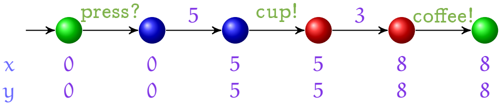
- Examples of concrete runs
  - Coffee with no sugar



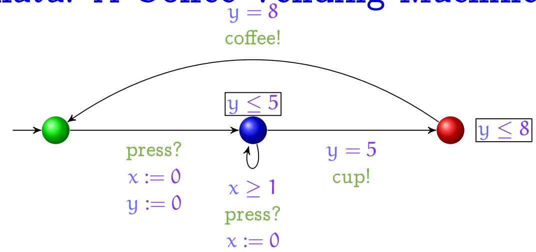
Timed Automata: A Coffee Vending Machine



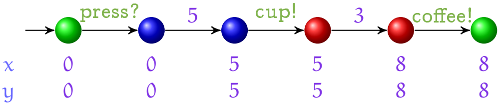
- Examples of concrete runs
  - Coffee with no sugar



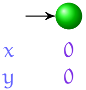
Timed Automata: A Coffee Vending Machine



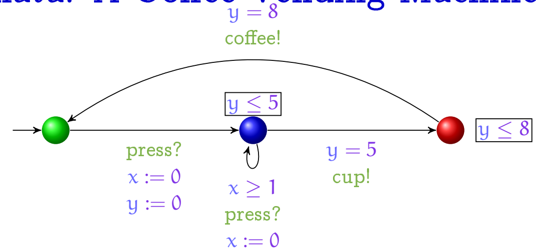
- Examples of concrete runs
  - Coffee with no sugar



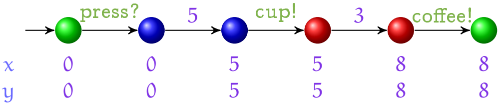
  - Coffee with 2 doses of sugar



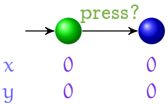
Timed Automata: A Coffee Vending Machine



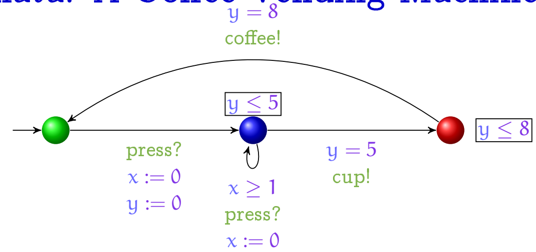
- Examples of concrete runs
  - Coffee with no sugar



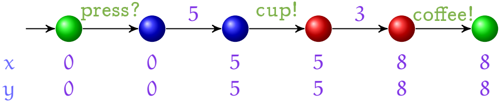
  - Coffee with 2 doses of sugar



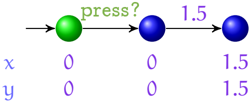
Timed Automata: A Coffee Vending Machine



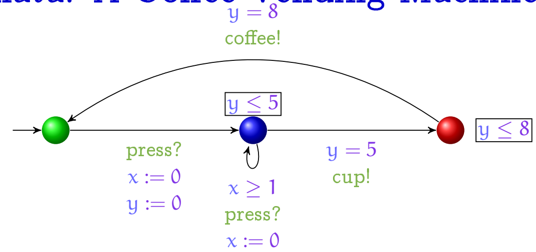
- Examples of concrete runs
  - Coffee with no sugar



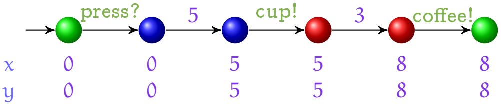
  - Coffee with 2 doses of sugar



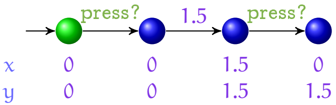
Timed Automata: A Coffee Vending Machine



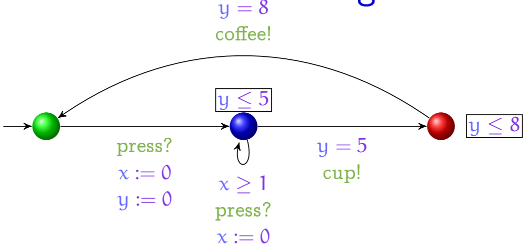
- Examples of concrete runs
  - Coffee with no sugar



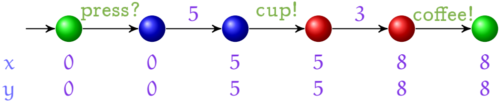
  - Coffee with 2 doses of sugar



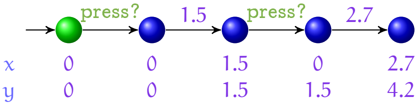
Timed Automata: A Coffee Vending Machine



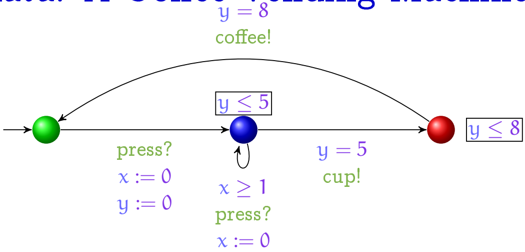
- Examples of concrete runs
  - Coffee with no sugar



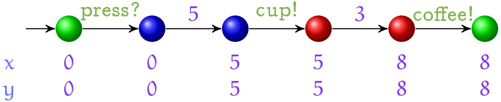
  - Coffee with 2 doses of sugar



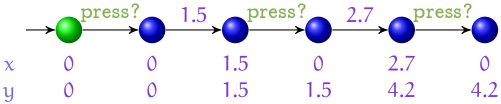
Timed Automata: A Coffee Vending Machine



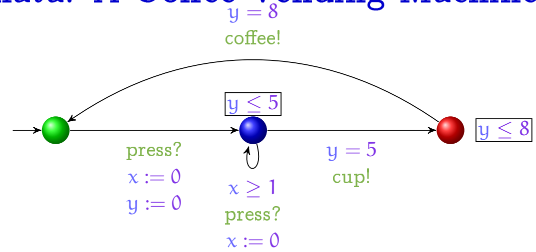
- Examples of concrete runs
  - Coffee with no sugar



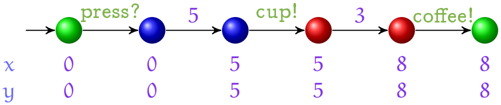
  - Coffee with 2 doses of sugar



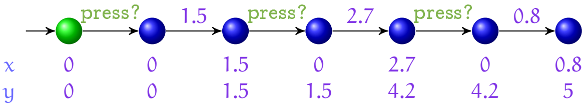
Timed Automata: A Coffee Vending Machine



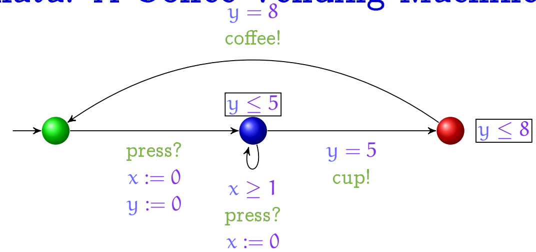
- Examples of concrete runs
  - Coffee with no sugar



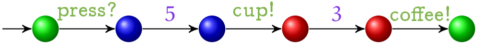
  - Coffee with 2 doses of sugar



Timed Automata: A Coffee Vending Machine

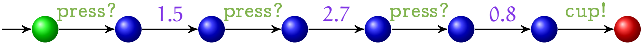


- Examples of concrete runs
  - Coffee with no sugar



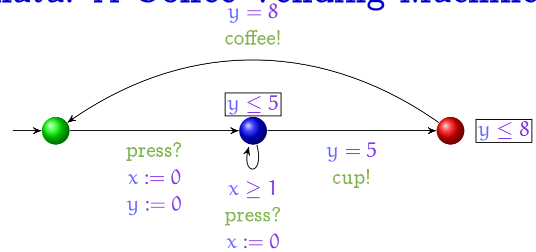
x	0	0	5	5	8	8
y	0	0	5	5	8	8

  - Coffee with 2 doses of sugar

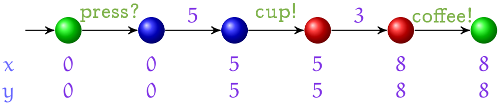


x	0	0	1.5	0	2.7	0	0.8	0.8
y	0	0	1.5	1.5	4.2	4.2	5	5

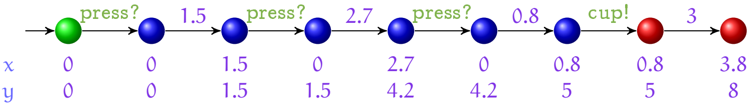
Timed Automata: A Coffee Vending Machine



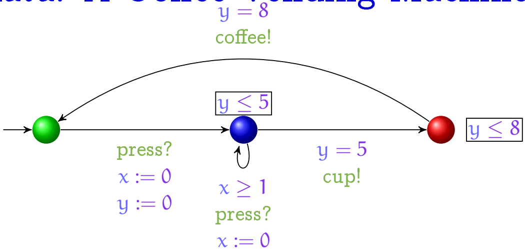
- Examples of concrete runs
  - Coffee with no sugar



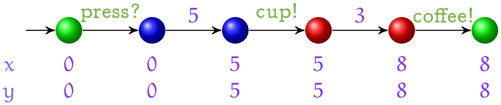
  - Coffee with 2 doses of sugar



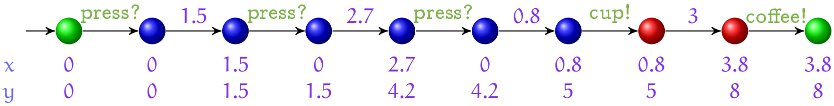
Timed Automata: A Coffee Vending Machine



- Examples of concrete runs
  - Coffee with no sugar

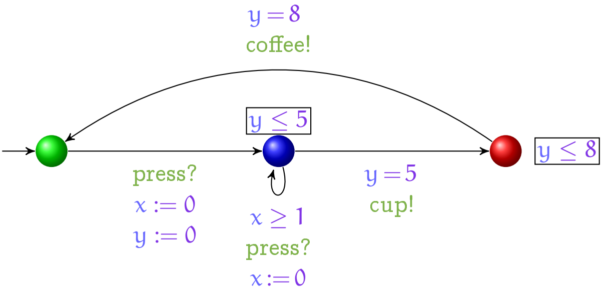


  - Coffee with 2 doses of sugar



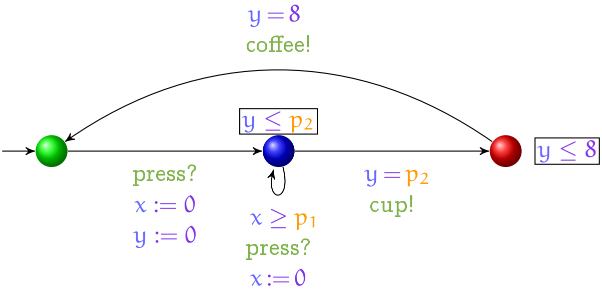
Parametric Timed Automaton (PTA)

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



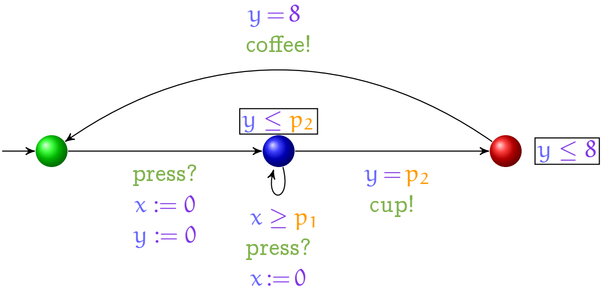
Parametric Timed Automaton (PTA)

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



Parametric Timed Automaton (PTA)

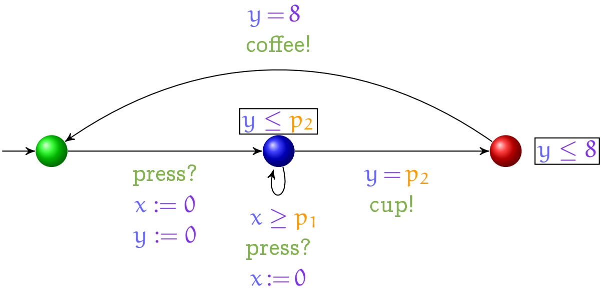
- Timed automaton (sets of locations, actions and clocks) augmented with a set P of parameters [Alur et al., 1993]
  - Unknown constants used in guards and invariants



- Examples of problems
  - "Do there exist parameter valuations such that one can never get a coffee?"

Parametric Timed Automaton (PTA)

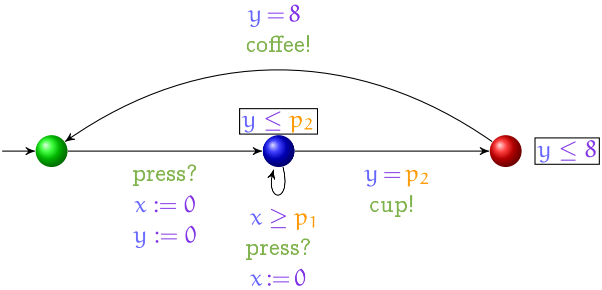
- Timed automaton (sets of locations, actions and clocks) augmented with a set P of parameters [Alur et al., 1993]
  - Unknown constants used in guards and invariants



- Examples of problems
  - "Do there exist parameter valuations such that one can never get a coffee?" Yes! e.g.: $p_1 = 2, p_2 = 10$

Parametric Timed Automaton (PTA)

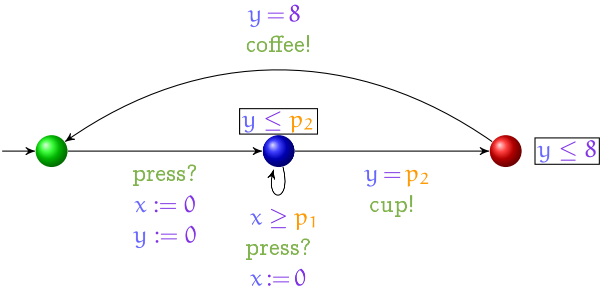
- Timed automaton (sets of locations, actions and clocks) augmented with a set P of parameters [Alur et al., 1993]
  - Unknown constants used in guards and invariants



- Examples of problems
  - "Do there exist parameter valuations such that one can never get a coffee?" Yes! e.g.: $p_1 = 2, p_2 = 10$
  - "What are all possible parameter valuations such that one can get a coffee with 3 doses of sugar?"

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 used in guards and invariants



- Examples of problems
  - "Do there exist parameter valuations such that one can never get a coffee?" Yes! e.g.: $p_1 = 2, p_2 = 10$
  - "What are all possible parameter valuations such that one can get a coffee with 3 doses of sugar?" $p_2 \leq 8 \wedge p_2 \geq 3 \times p_1$

Valuation of a PTA

- A valuation π of all the parameters of P is called a point
- Given a PTA $\mathcal{A}$ and a point π, we denote by $\mathcal{A}[\pi]$ the (non-parametric) timed automaton where all parameters are valuated by π

Objective: Reachability Synthesis

Problem (EF-emptiness)

Let $\mathcal{A}$ be a PTA. Is the set of parameter valuations π such that $\mathcal{A}[\pi]$ reaches l_{bad} empty?

Objective: Reachability Synthesis

Problem (EF-emptiness)

Let $\mathcal{A}$ be a PTA. Is the set of parameter valuations π such that $\mathcal{A}[\pi]$ reaches l_{bad} empty?

Theorem

The EF-emptiness problem is undecidable. [Alur et al., 1993]

Objective: Reachability Synthesis

Problem (EF-emptiness)

Let $\mathcal{A}$ be a PTA. Is the set of parameter valuations π such that $\mathcal{A}[\pi]$ reaches l_{bad} empty?

Theorem

The EF-emptiness problem is undecidable. [Alur et al., 1993]

Problem (EF-synthesis)

Let $\mathcal{A}$ be a PTA. Compute the set of parameter valuations π such that $\mathcal{A}[\pi]$ reaches l_{bad}.

Previous Works

- Semi-algorithm EFsynth proposed in [Alur et al., 1993]
- Synthesis of integer parameter valuations
  - Enumerative terminating algorithm for 2 subclasses of PTA ("L-PTA and U-PTA") [Bozzelli and La Torre, 2009]
  - Symbolic terminating algorithm for general PTA with a bounded parameter domain [Jovanović et al., 2014]

Previous Works

- Semi-algorithm EFsynth proposed in [Alur et al., 1993]
- Synthesis of integer parameter valuations
  - Enumerative terminating algorithm for 2 subclasses of PTA ("L-PTA and U-PTA") [Bozzelli and La Torre, 2009]
  - Symbolic terminating algorithm for general PTA with a bounded parameter domain [Jovanović et al., 2014]
- Here: reachability preservation-based approach
  - For rational-valued parameter valuations

Previous Works

- Semi-algorithm EFsynth proposed in [Alur et al., 1993]

- Synthesis of integer parameter valuations
  - Enumerative terminating algorithm for 2 subclasses of PTA ("L-PTA and U-PTA") [Bozzelli and La Torre, 2009]
  - Symbolic terminating algorithm for general PTA with a bounded parameter domain [Jovanović et al., 2014]

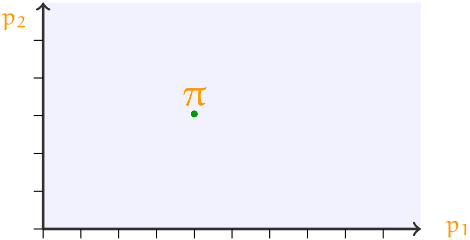
- Here: reachability preservation-based approach
  - For rational-valued parameter valuations
  - ☺ . . . and that can be distributed

Outline

Reachability Preservation

Key idea

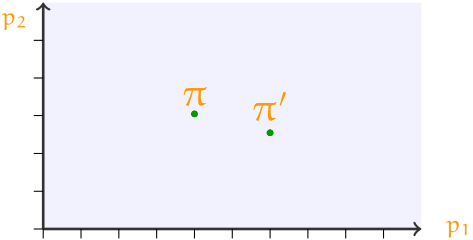
"If we know a parameter valuation π that reaches (resp. does not reach) l_{bad}, can we find other valuations around π that reach (resp. do not reach) l_{bad}?"



Reachability Preservation

Key idea

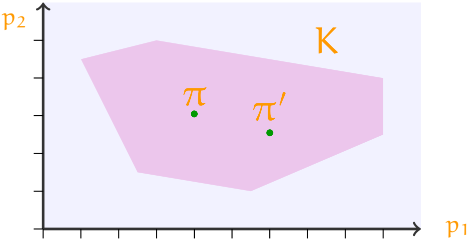
"If we know a parameter valuation π that reaches (resp. does not reach) l_{bad}, can we find other valuations around π that reach (resp. do not reach) l_{bad}?"



Reachability Preservation

Key idea

"If we know a parameter valuation π that reaches (resp. does not reach) l_{bad}, can we find other valuations around π that reach (resp. do not reach) l_{bad}?"



Reachability Preservation: Undecidability

Problem (PREACH-emptiness)

Let $\mathcal{A}$ be a PTA, and π a parameter valuation. Does there exist $\pi' \neq \pi$ such that $\mathcal{A}[\pi']$ preserves the reachability of l_{bad} in $\mathcal{A}[\pi]$?

Reachability Preservation: Undecidability

Problem (PREACH-emptiness)

Let $\mathcal{A}$ *be a PTA, and* π *a parameter valuation. Does there exist* $\pi' \neq \pi$ *such that* $\mathcal{A}[\pi']$ *preserves the reachability of* l_{bad} *in* $\mathcal{A}[\pi]$*?*

Theorem

PREACH-emptiness is undecidable.

Reachability Preservation: Undecidability

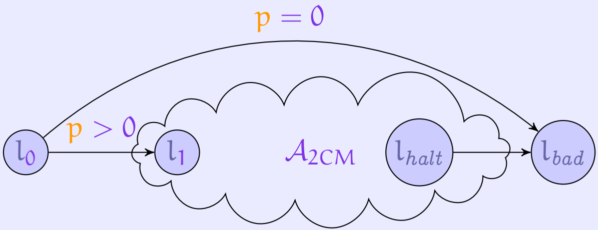
Problem (PREACH-emptiness)

Let $\mathcal{A}$ be a PTA, and π a parameter valuation. Does there exist $\pi' \neq \pi$ such that $\mathcal{A}[\pi']$ preserves the reachability of l_{bad} in $\mathcal{A}[\pi]$?

Theorem

PREACH-emptiness is undecidable.

Proof.

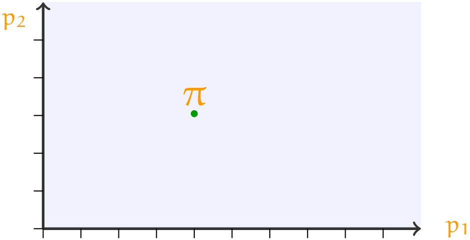


PRP: Parametric Reachability Preservation

Input: parameter valuation π

Output: constraint K such that

1. $\pi \models K$, and
2. $\forall \pi' \models K$, $\mathcal{A}[\pi']$ preserves the reachability of l_{bad} in $\mathcal{A}[\pi]$



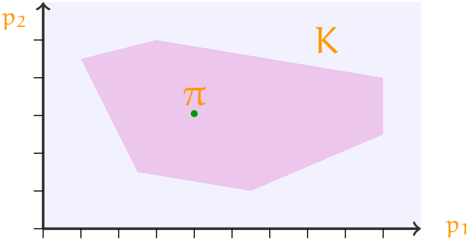
Inspired by EFsynth [Alur et al., 1993, Jovanović et al., 2014] and IM^K [André and Soulat, 2011]

PRP: Parametric Reachability Preservation

Input: parameter valuation π

Output: constraint K such that

1. $\pi \models K$, and
2. $\forall \pi' \models K$, $\mathcal{A}[\pi']$ preserves the reachability of l_{bad} in $\mathcal{A}[\pi]$

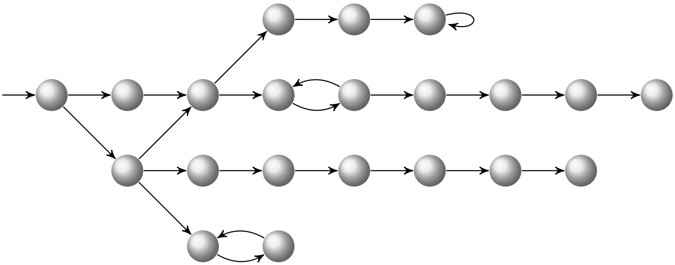


Inspired by EFsynth [Alur et al., 1993, Jovanović et al., 2014] and IM^K [André and Soulat, 2011]

PRP: Case 1

As long as l_{bad} is not met...

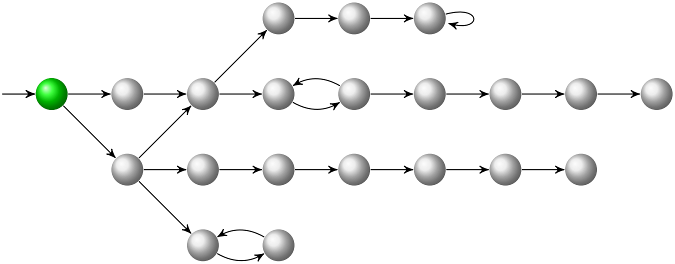
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

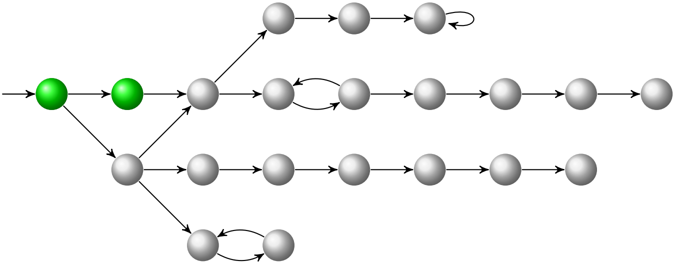
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

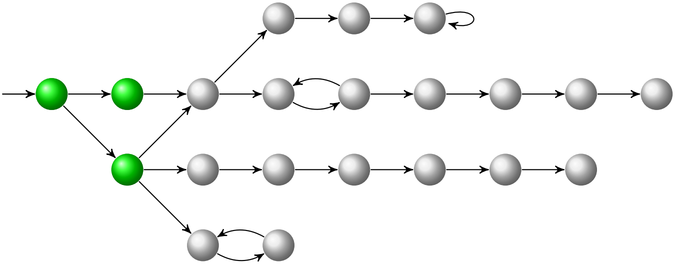
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

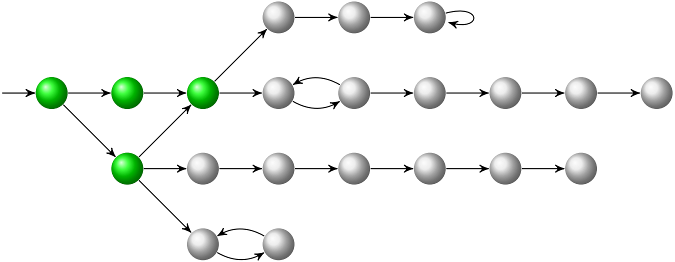
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

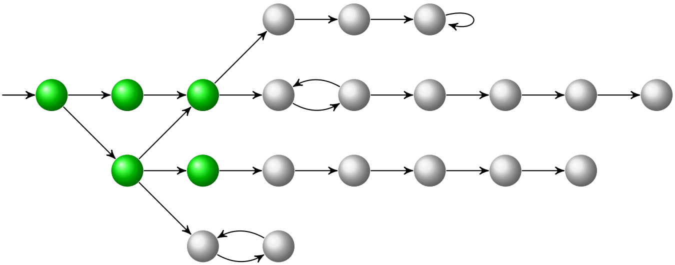
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

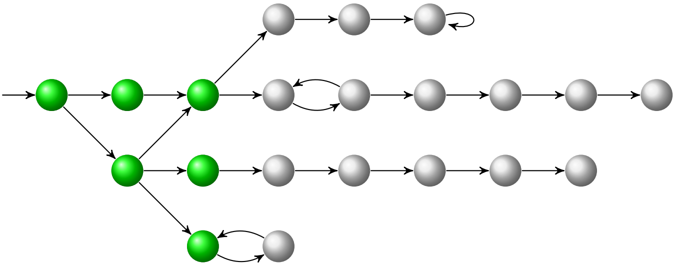
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

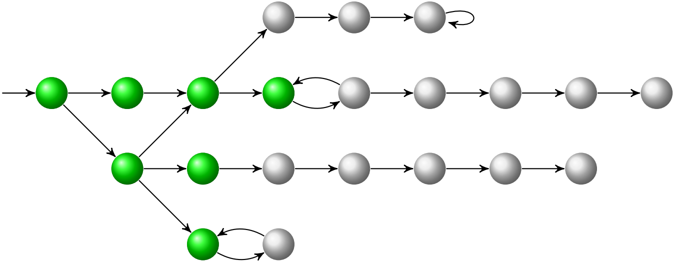
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

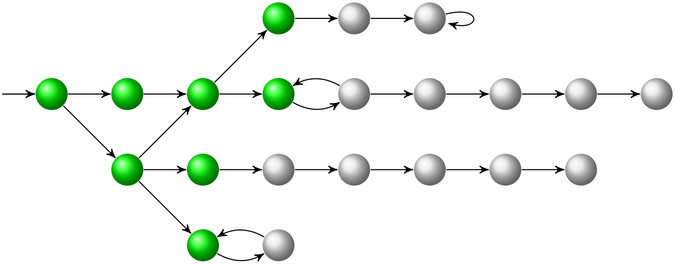
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

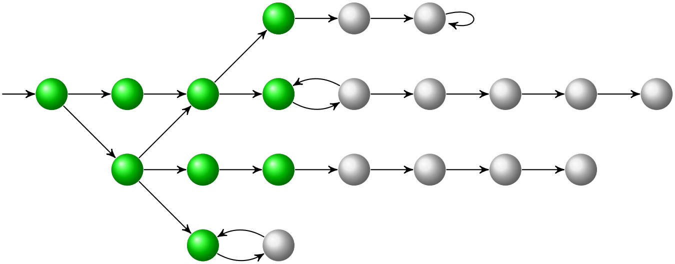
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

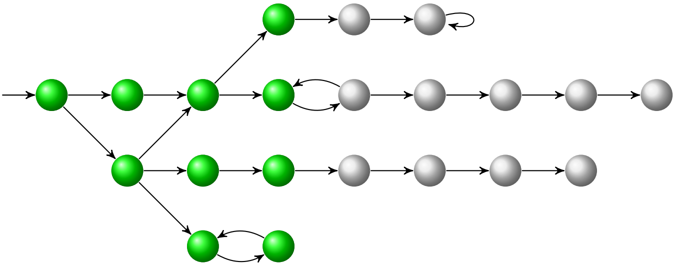
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

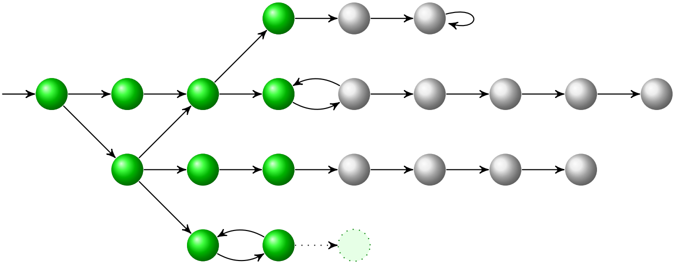
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

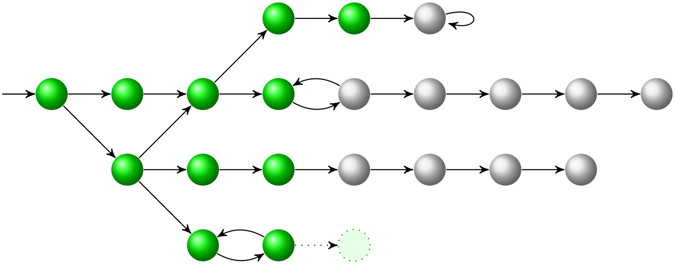
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

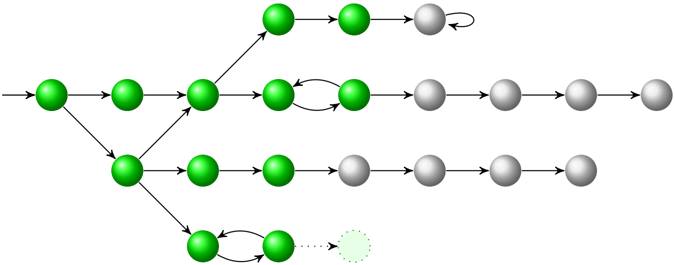
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

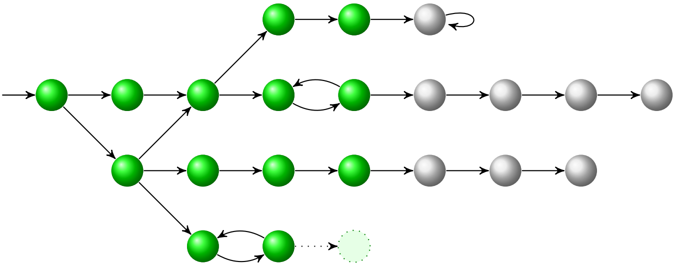
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

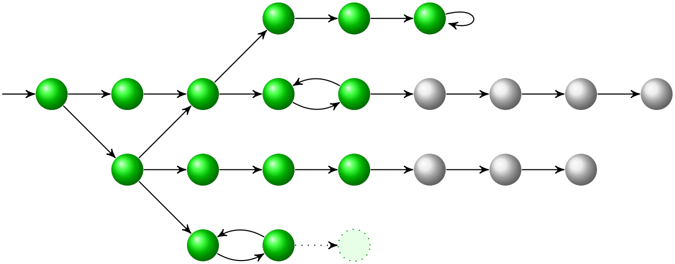
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

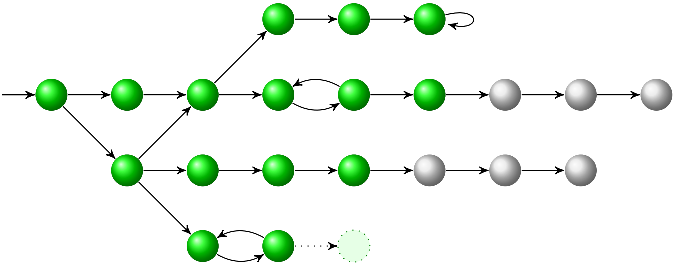
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

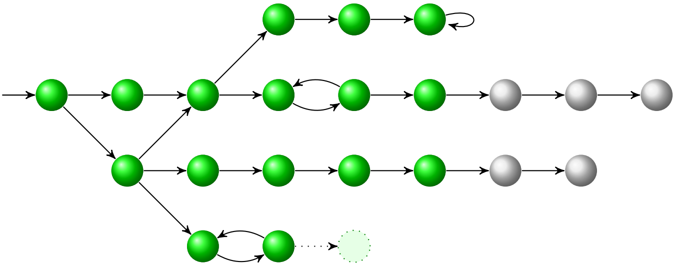
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

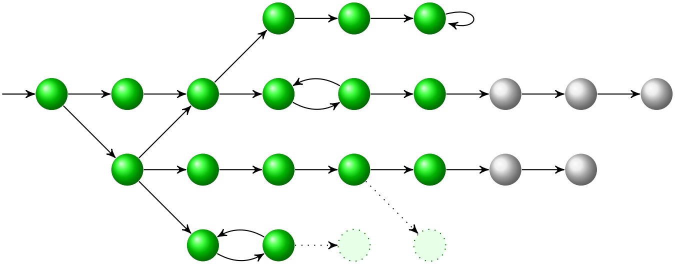
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

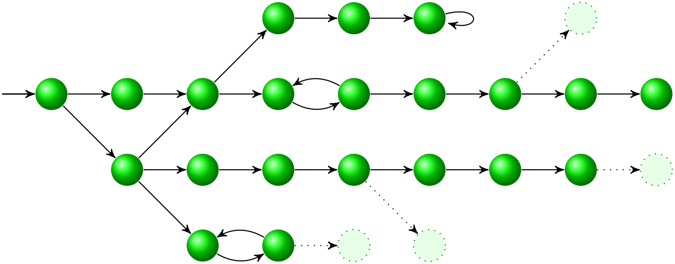
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

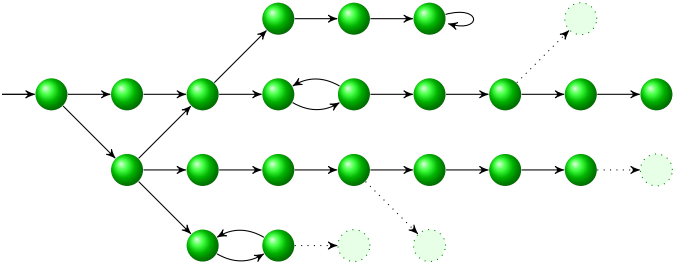
- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



PRP: Case 1

As long as l_{bad} is not met...

- Explore the symbolic state space
- But do not explore the behaviors not present in $\mathcal{A}[\pi]$!



When no successors, and if l_{bad} was never met:

- return $\neg \bigcirc \land \cdots \land \neg \bigcirc$
- Ensures a subset of the behaviors of $\mathcal{A}[\pi]$, and hence guarantees the unreachability of l_{bad}

PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $\mathcal{A}[\pi]$?

How do we know that a symbolic state of $\mathcal{A}$ corresponds to a behavior of $\mathcal{A}[\pi]$?

PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $\mathcal{A}[\pi]$?
How do we know that a symbolic state of $\mathcal{A}$ corresponds to a behavior of $\mathcal{A}[\pi]$?

We could compute the zone graph of $\mathcal{A}[\pi]$.
But this is not necessary.
In fact, we do not even need to know whether $\mathcal{A}[\pi]$ reaches l_{bad} or not.

PRP: Case 1 (Remark)

Questions

How do we know the possible behaviors of $\mathcal{A}[\pi]$?
How do we know that a symbolic state of $\mathcal{A}$ corresponds to a behavior of $\mathcal{A}[\pi]$?

We could compute the zone graph of $\mathcal{A}[\pi]$.
But this is not necessary.
In fact, we do not even need to know whether $\mathcal{A}[\pi]$ reaches l_{bad} or not.

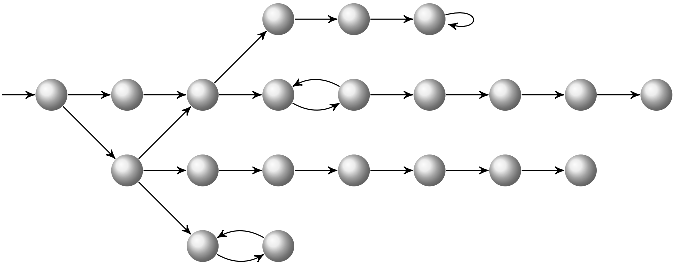
Trick

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

PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm. . .

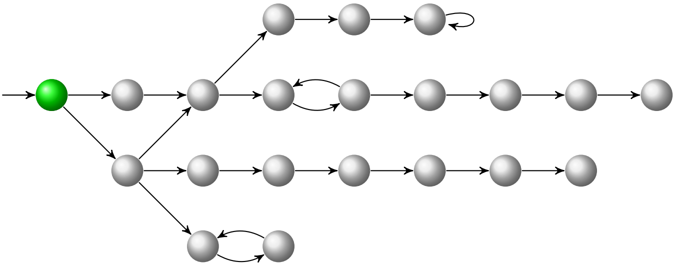
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

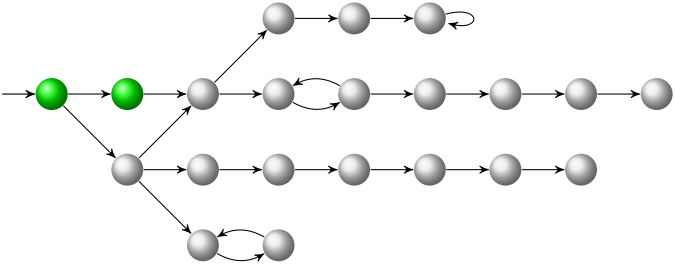
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

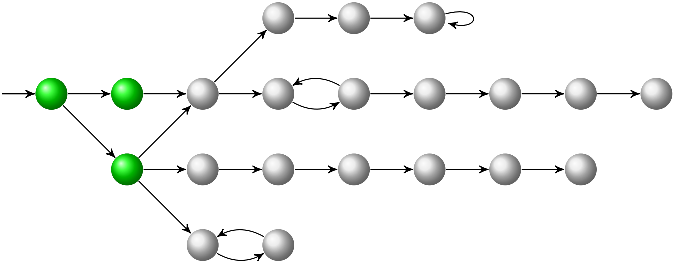
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

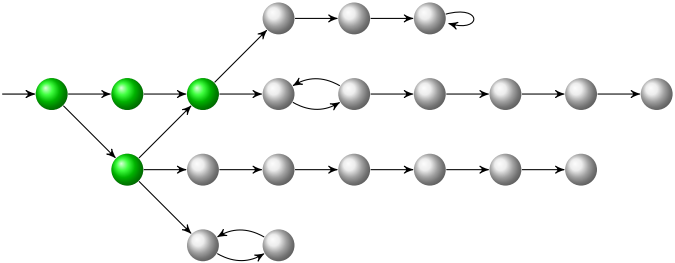
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

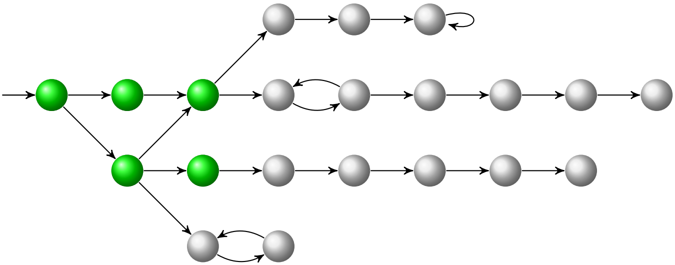
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

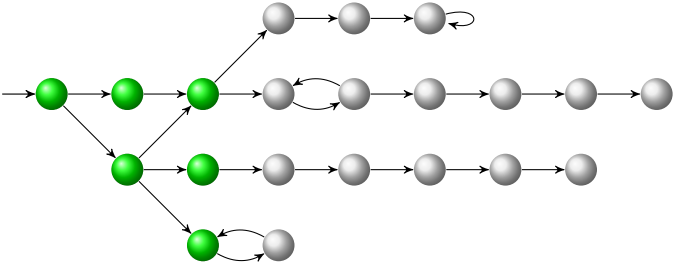
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

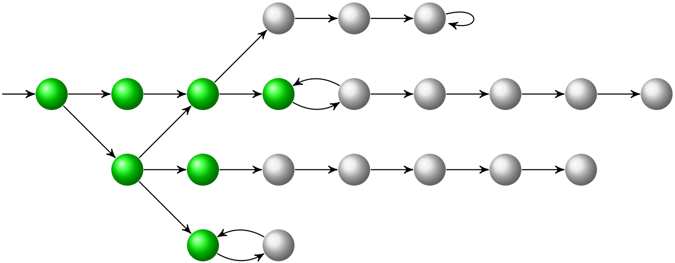
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

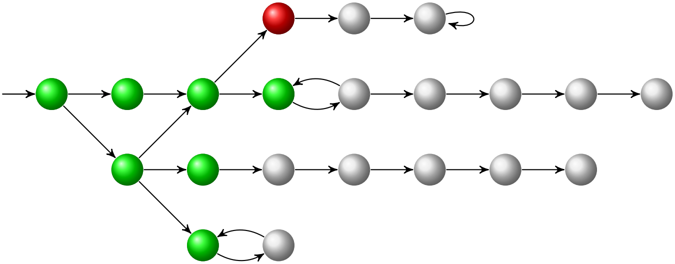
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

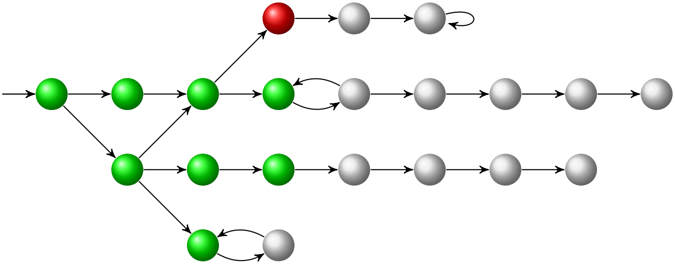
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

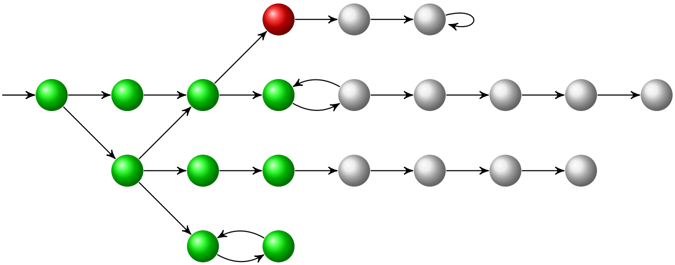
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

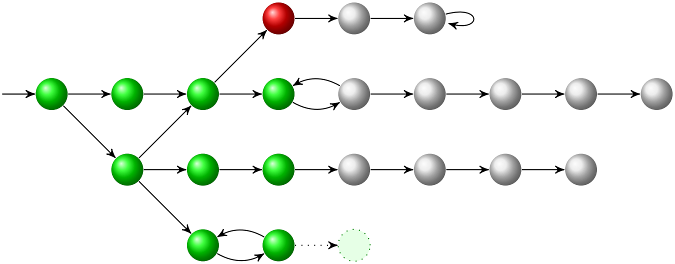
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

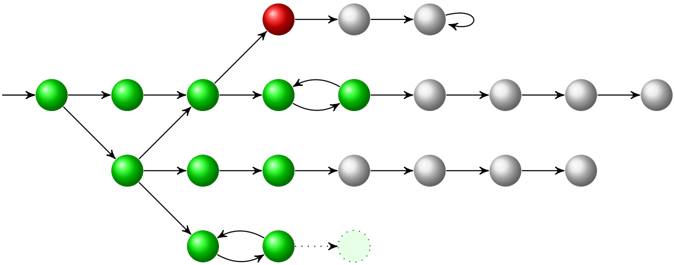
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

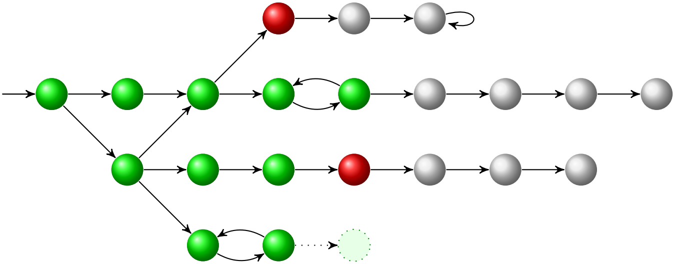
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

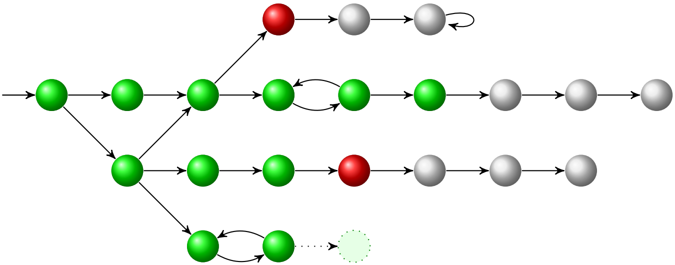
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

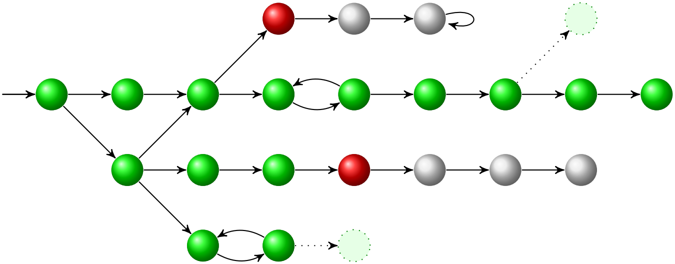
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm...

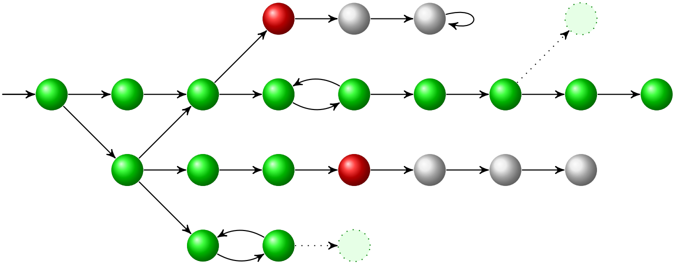
- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



PRP: Case 2

When l_{bad} is met, switch to an EFsynth-like algorithm…

- But still without exploring the behaviors not present in $\mathcal{A}[\pi]$



When no successors, and if l_{bad} was met:

- return  $\vee \cdots \vee$ 
- Guarantees the reachability of l_{bad}

PRP: Early termination

Recall that PREACH-emptiness is undecidable

Hence PRP may not terminate.

PRP: Early termination

Recall that PREACH-emptiness is undecidable
Hence PRP may not terminate.

Proposition (Early termination)

If $\mathsf{PRP}(\mathcal{A}, \pi)$ *does not terminate and is interrupted (e.g., after a timeout), the result is still a valid under-approximation provided* l_{bad} *has been reached.*

This is also true for EFsynth (in any case)

Outline

Perform EF-synthesis using PRP

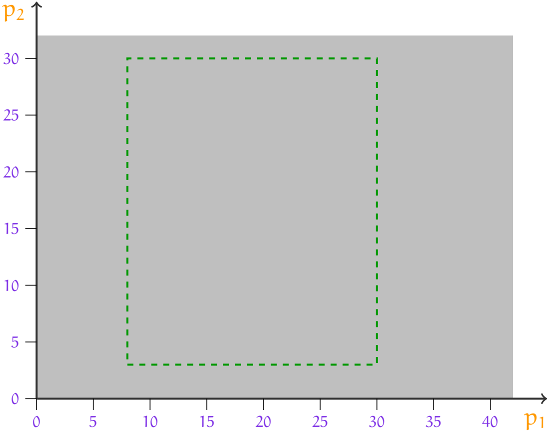
Input: parameter bounded domain V
Output: constraints on the parameter such that l_{bad} is / is not reachable in $\mathcal{A}$

- The idea: reuse the "behavioral cartography" of parametric timed automata [André and Fribourg, 2010]
- Iterate on integer points, and call PRP on each point not covered by a constraint
  - If no termination: break, and keep result if possible (i.e., if l_{bad} is reachable in this analysis)

PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

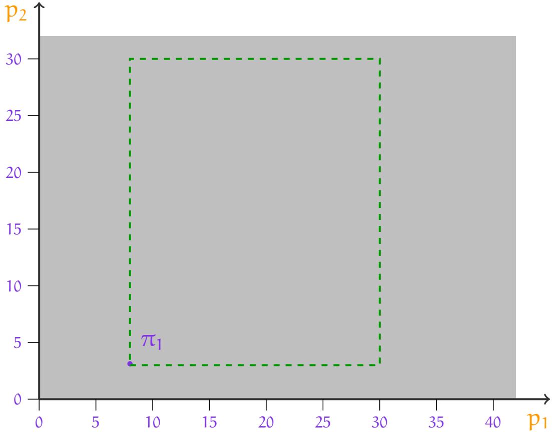
Method: done by calling PRP on integer points (parameter valuations) sequentially



PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

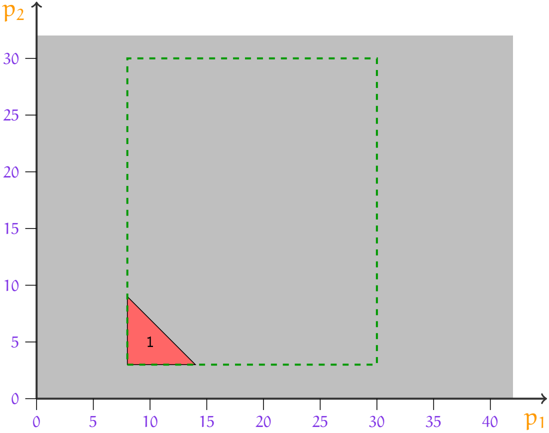
Method: done by calling PRP on integer points (parameter valuations) sequentially



PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

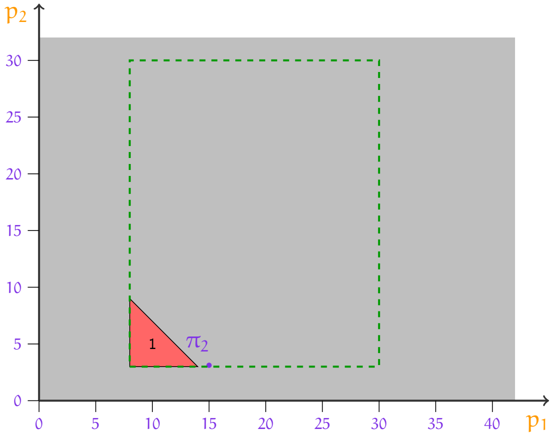
Method: done by calling PRP on integer points (parameter valuations) sequentially



PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

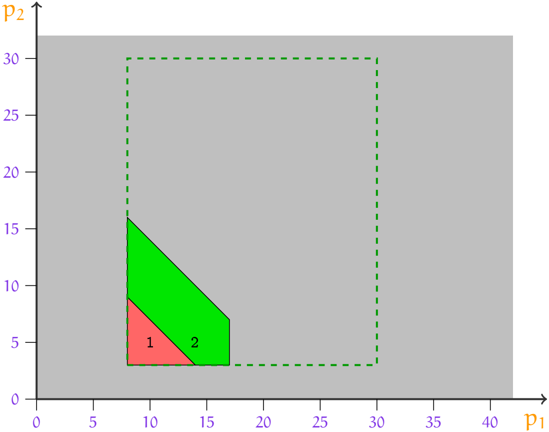
Method: done by calling PRP on integer points (parameter valuations) sequentially



PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

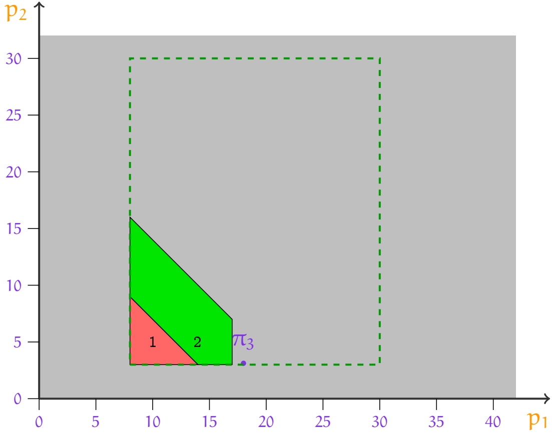
Method: done by calling PRP on integer points (parameter valuations) sequentially



PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

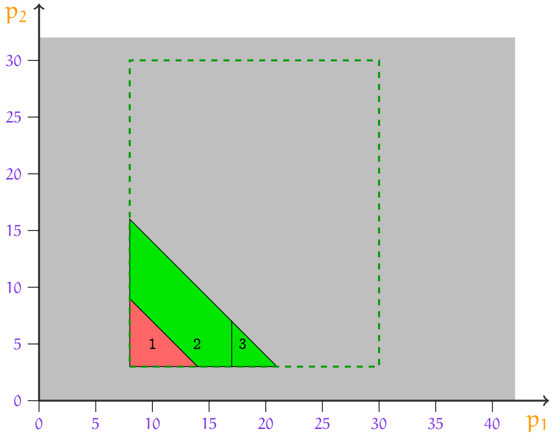
Method: done by calling PRP on integer points (parameter valuations) sequentially



PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

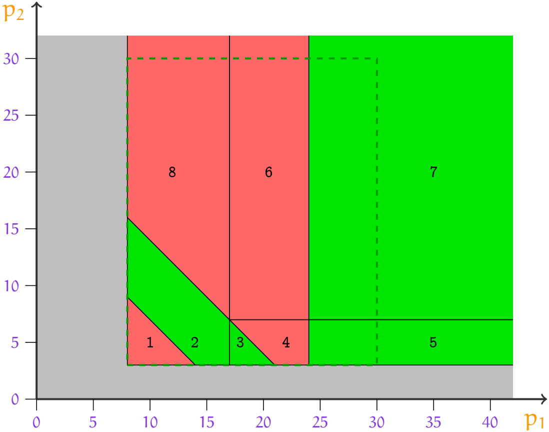
Method: done by calling PRP on integer points (parameter valuations) sequentially



PRPC: Reusing the Behavioral Cartography

Partition the domain V into constraints where the reachability of l_{bad} is uniform

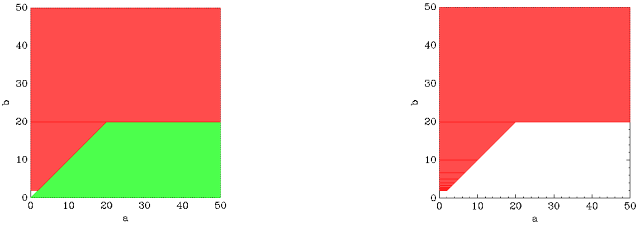
Method: done by calling PRP on integer points (parameter valuations) sequentially



Result: "interval" under-approximation

- PRPC synthesizes:
  - An under-approximation of the bad constraints (reaching l_{bad})
  - An under-approximation of the good constraints (avoiding l_{bad})
- EFsynth synthesizes:
  - An under-approximation of the bad constraints

⇒ The result of PRPC is more valuable than EFsynth, at least when EFsynth does not terminate and is interrupted



PRPC EFsynth

Towards Distributed Parameter Synthesis

Idea

Calling sequentially PRP on various integer points in a bounded parameter domain looks like something that can be easily distributed.

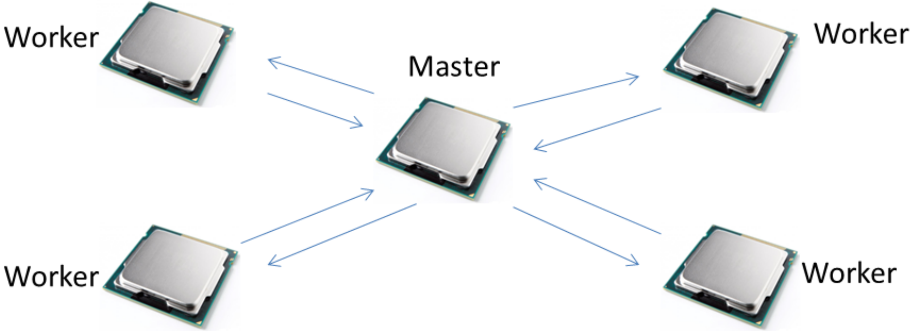
Towards Distributed Parameter Synthesis

Idea

Calling sequentially PRP on various integer points in a bounded parameter domain looks like something that can be easily distributed.

Reuse the distributed algorithms to compute the behavioral cartography of parametric timed automata [A., Coti, Evangelista, 2014]

Master Worker Scheme



Master-Worker distribution scheme:

- Workers: ask the master for a point, calls PRP on that point, and send the result (constraint) to the master
- Master: is responsible for smart repartition of data between the workers
  - (Note: not trivial at all)

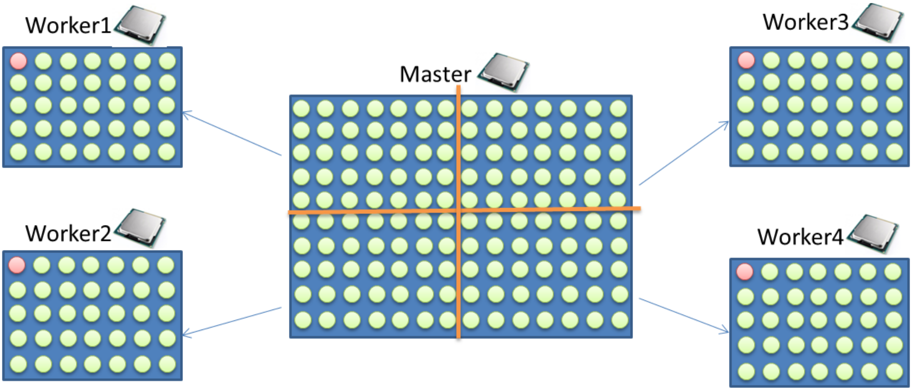
Dynamic Decomposition of BC

Most efficient distributed algorithm for BC (so far!):
"Domain decomposition" scheme [work in progress]

- Master
  1. initially splits the parameter domain into subdomains and send them to the workers
  2. when a worker has completed its subdomain, the master splits another subdomain, and sends it to the idle worker

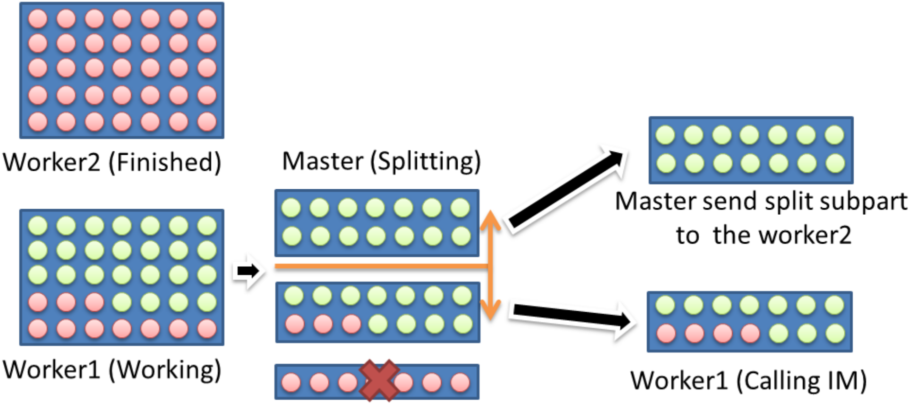
- Workers
  1. receives the subdomain from the master
  2. calls PRP on the points of this subdomain
  3. sends the results (list of constraints) back to the master
  4. asks for more work

Domain Decomposition: Initial Splitting



- Prevent to choose close points
- Prevent bottleneck phenomenon at the master side
  - Master only responsible for gathering constraints and splitting subdomains

Domain Decomposition: Dynamic Splitting



- Master can balance workload between workers

Outline

Implementation in IMITATOR

- IMITATOR [A., Fribourg, Kühne, Soulat, 2012]
  - 26,000 lines of OCaml code
    - Development started in 2009... in Hilton Pasadena!
  - Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
  - Available under the GNU-GPL license
  - Latest version (2.7) implements distributed algorithms

- Distributed version of IMITATOR relying on MPI
  - Using the OcamlMPI library

Implementation in IMITATOR

- IMITATOR [A., Fribourg, Kühne, Soulat, 2012]
  - 26,000 lines of OCaml code
    - Development started in 2009... in Hilton Pasadena!
  - Relies on the PPL library for operations on polyhedra [Bagnara et al., 2008]
  - Available under the GNU-GPL license
  - Latest version (2.7) implements distributed algorithms

- Distributed version of IMITATOR relying on MPI
  - Using the OcamlMPI library

http://www.imitator.fr/

PRPC: experiments

Case study	$\|X\|$	$\|V\|$	EFsynth	BC	PRPC	PRPC distr(12)
$\mathcal{A}_1$	2	2,601	0.401*	TO	0.078*	0.050*
Sched1	13	6,561	TO	TO	1,595	219
Sched2.50.0	6	3,321	9.25	990	14.55	4.77
Sched2.50.2	6	3,321	662	TO	213	84
Sched2.100.0	6	972,971	21.4	2,093	116	10.1
Sched2.100.2	6	972,971	3,757	TO	4,557	1,543
Sched5	21	1,681	352	TO	TO	917
SPSMALL	11	3,082	7.49	587	118	11.2

IMITATOR version: 2.6.2, build 845
* experiment run using -depth-limit 10 (does not terminate in general)
Experiments available at http://www.imitator.fr/static/NFM15/

Outline

Summary

PRP

- Given a parameter valuation π and a location l_{bad}, outputs a dense set of parameter valuations around π that preserve the (un)reachability of l_{bad}

PRPC

- Computes an under-approximated set of parameter valuations reaching / not reaching l_{bad}
- Can be distributed
- Often outperforms EFsynth, especially when distributed

Perspectives

- Improvement: always return both good and bad constraints (for both PRP and EFsynth)
- Combine with integer hull to ensure termination [Jovanović et al., 2014]
  - At least for integer valuations
- Combine with multi-core techniques [Laarman et al., 2013]
- Verify the communication scheme in the distributed IMITATOR for an arbitrary number of nodes
  - Using parametric verification techniques?

Perspectives

- Improvement: always return both good and bad constraints (for both PRP and EFsynth)
- Combine with integer hull to ensure termination [Jovanović et al., 2014]
  - At least for integer valuations
- Combine with multi-core techniques [Laarman et al., 2013]
- Verify the communication scheme in the distributed IMITATOR for an arbitrary number of nodes
  - Using parametric verification techniques?
- Extend to compositional verification

Bibliography

References I

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

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

André, É., Coti, C., and Evangelista, S. (2014).
Distributed behavioral cartography of timed automata.
In Dongarra, J., Ishikawa, Y., and Atsushi, H., editors, *21st European MPI Users' Group Meeting (EuroMPI/ASIA'14)*, pages 109–114. ACM.

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

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

References II

André, É. and Soulat, R. (2011).
Synthesis of timing parameters satisfying safety properties.
In Delzanno, G. and Potapov, I., editors, *Proceedings of the 5th Workshop on Reachability Problems in Computational Models (RP'11)*, volume 6945 of *Lecture Notes in Computer Science*, pages 31–44. Springer.

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

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.

Jovanović, A., Lime, D., and Roux, O. H. (2014).
Integer parameter synthesis for timed automata.
IEEE Transactions on Software Engineering (TSE).
To appear.

References III

Laarman, A., Olesen, M. C., Dalsgaard, A. E., Larsen, K. G., and Van De Pol, J. (2013).
Multi-core emptiness checking of timed Büchi automata using inclusion abstraction.
In *CAV*, volume 8044 of *Lecture Notes in Computer Science*.

Markey, N. (2011).
Robustness in real-time systems.
In *SIES*, pages 28–34. IEEE Computer Society Press.

Additional explanation

PRP: The Algorithm

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

input : PTA $\mathcal{A}$ of initial state s_0, parameter valuation π

output : Constraint over the parameters

$S \leftarrow \emptyset$; $S_{new} \leftarrow \{s_0\}$; $Bad \leftarrow \texttt{false}$; $K_{good} \leftarrow \top$; $K_{bad} \leftarrow \bot$; $i \leftarrow 0$

while $\texttt{true}$ **do**

  foreach π-*incompatible state* (l, C) *in* S_{new} **do**

    $S_{new} \leftarrow S_{new} \setminus \{(l, C)\}$

    if $Bad = \texttt{false}$ **then**

      Select a π-incompatible inequality J in $C\downarrow_P$ (i.e., s.t. $\pi \not\models J$)

      $K_{good} \leftarrow K_{good} \wedge \neg J$

  foreach *bad state* (l_{bad}, C) *in* S_{new} **do**

    $Bad \leftarrow \texttt{true}$; $K_{bad} \leftarrow K_{bad} \vee C\downarrow_P$; $S_{new} \leftarrow S_{new} \setminus \{(l_{bad}, C)\}$

  if $S_{new} \subseteq S$ **then**

    if $Bad = \texttt{true}$ **then return** K_{bad} **else return** K_{good} ;

  $S \leftarrow S \cup S_{new}$; $S_{new} \leftarrow Succ(S_{new})$; $i \leftarrow i + 1$

Licensing

Source of the graphics used I

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

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

License of this document