ICTAC 2016

Wednesday 26th of October, 2016
臺北，臺灣

Parametric Deadlock-Freeness Checking Timed Automata

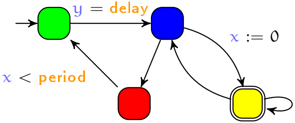
Étienne André[1,2]

[1] LIPN, Université Paris 13, Sorbonne Paris Cité, CNRS, France
[2] École Centrale de Nantes, IRCCyN, CNRS, UMR 6597, France

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

Context: timed model checking

- Timed model checking



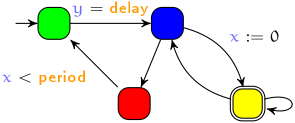
 is unreachable

A model of the system

A property to be satisfied

Context: timed model checking

- Timed model checking



$\stackrel{?}{\models}$

 is unreachable

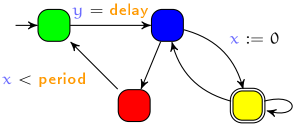
A model of the system

A property to be satisfied

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

Context: timed model checking

- Timed model checking



$\overset{?}{\models}$

 is unreachable

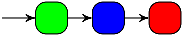
A model of the system

A property to be satisfied

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

Yes

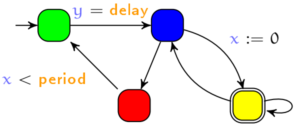
No



Counterexample

Context: parametric timed model checking

- Timed model checking



$\overset{?}{\models}$

 is unreachable

A model of the system

A property to be satisfied

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

Yes if. . .

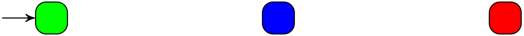
$$2\text{delay} > \text{period} \wedge \text{period} < 20.46$$

Outline

Outline

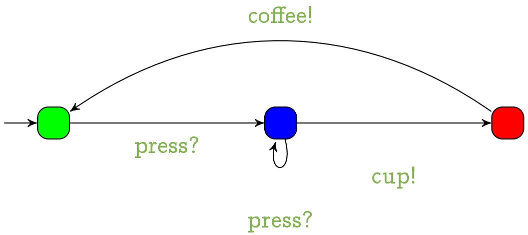
Timed automaton (TA)

- Finite state automaton (sets of locations)



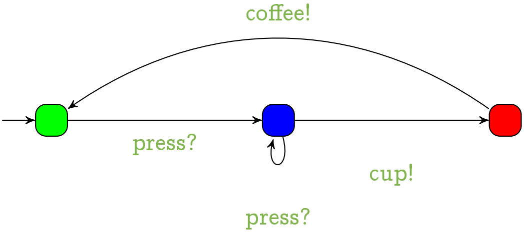
Timed automaton (TA)

- Finite state automaton (sets of locations and actions)



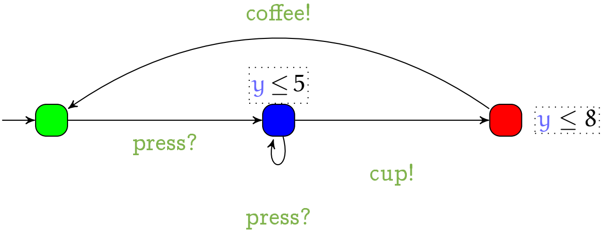
Timed automaton (TA)

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



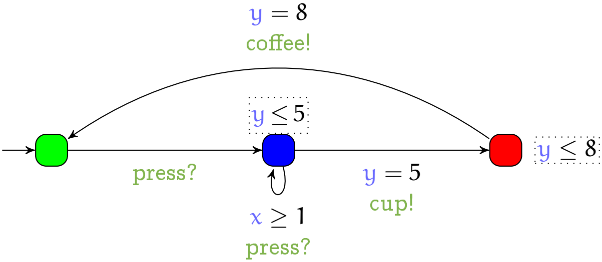
Timed automaton (TA)

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



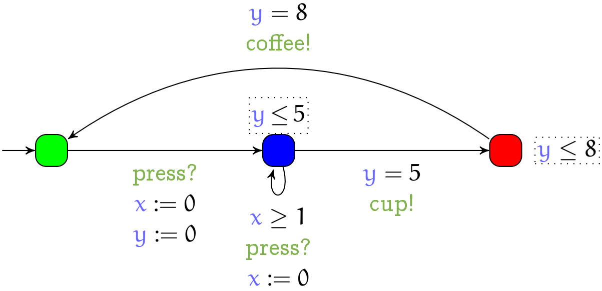
Timed automaton (TA)

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



Timed automaton (TA)

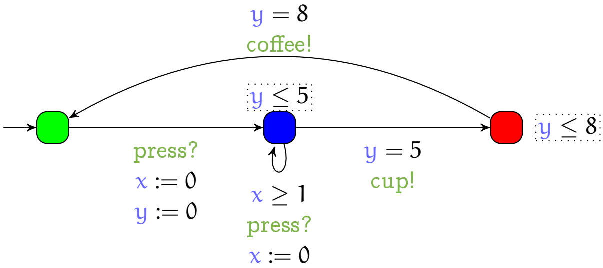
- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994, Henzinger et al., 1994]
  - Real-valued variables evolving linearly at the same rate
- Features
  - Location invariant: constraint to be verified to stay at a location
  - Transition guard: constraint to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition



Concrete semantics of timed automata

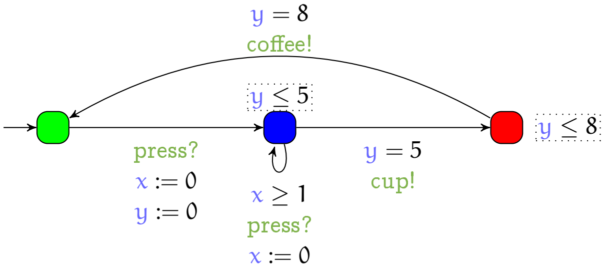
- Concrete state of a TA: pair (l, w), where
  - l is a location,
  - w is a valuation of each clock
- Concrete run: alternating sequence of concrete states and actions or time elapse

Examples of concrete runs

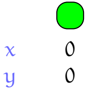


- Possible concrete runs for the coffee machine

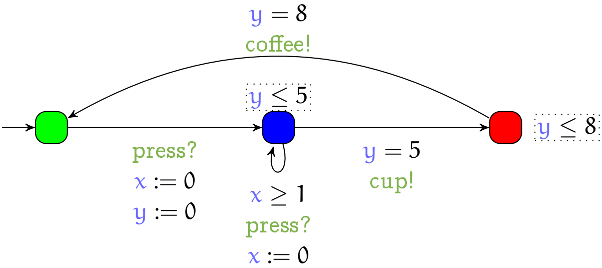
Examples of concrete runs



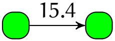
- Possible concrete runs for the coffee machine
  - Coffee with no sugar



Examples of concrete runs

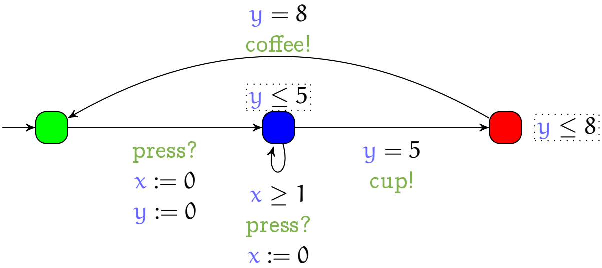


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

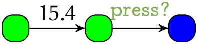


x	0	15.4
y	0	15.4

Examples of concrete runs

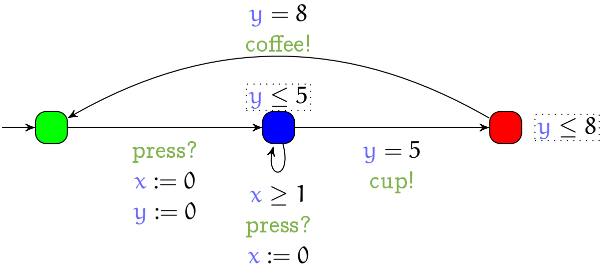


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



x	0	15.4	0
y	0	15.4	0

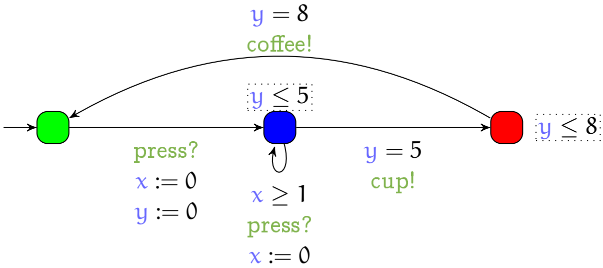
Examples of concrete runs



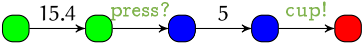
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

x	0	15.4	0	5
y	0	15.4	0	5

Examples of concrete runs

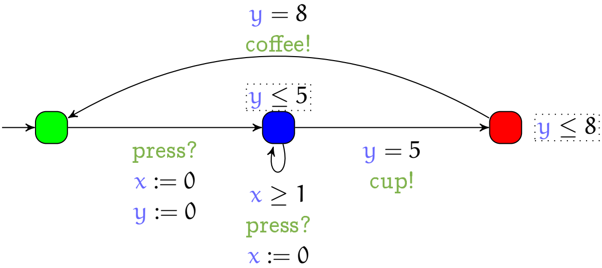


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

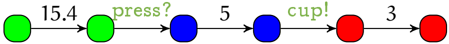


x	0	15.4	0	5	5
y	0	15.4	0	5	5

Examples of concrete runs

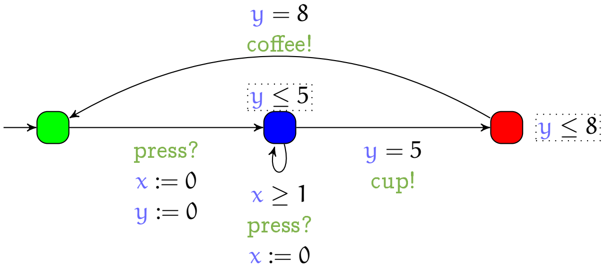


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

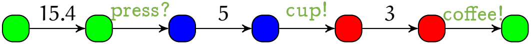


x	0	15.4	0	5	5	8
y	0	15.4	0	5	5	8

Examples of concrete runs

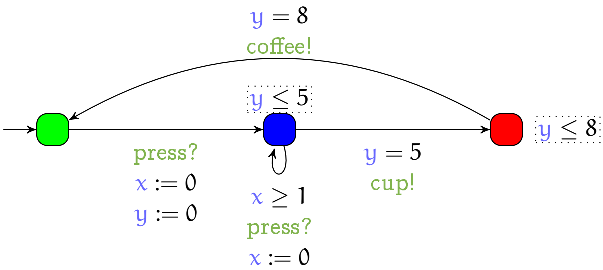


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

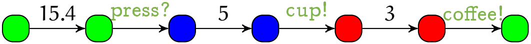


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar

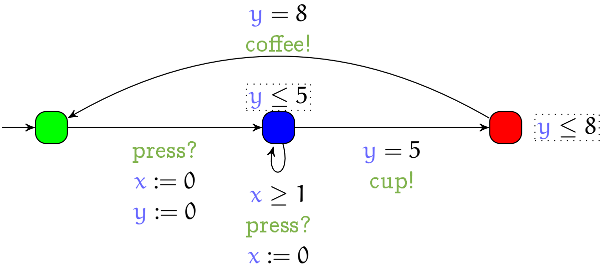


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

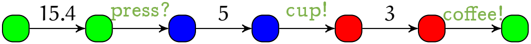
  - Coffee with 2 doses of sugar

x	0
y	0

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar

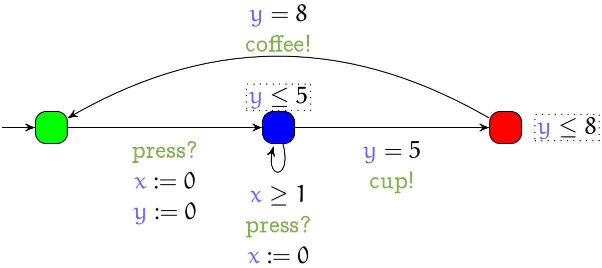


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

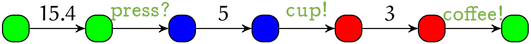
  - Coffee with 2 doses of sugar

x	0	0
y	0	0

Examples of concrete runs

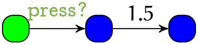


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



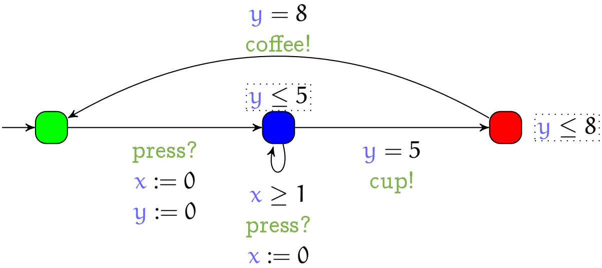
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

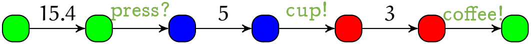


x	0	0	1.5
y	0	0	1.5

Examples of concrete runs

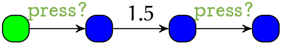


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



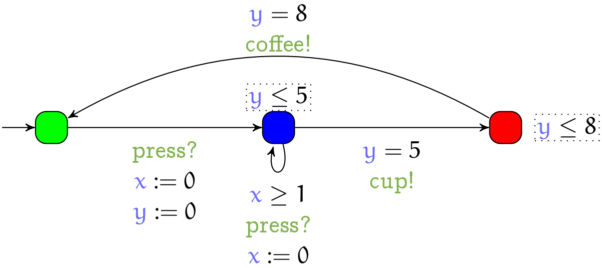
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

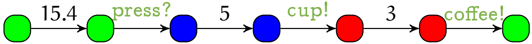


x	0	0	1.5	0
y	0	0	1.5	1.5

Examples of concrete runs

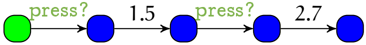


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



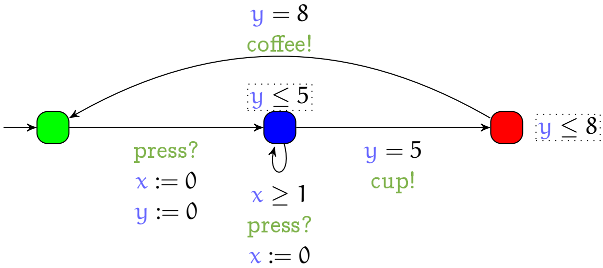
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

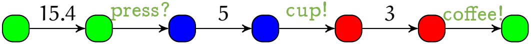


x	0	0	1.5	0	2.7
y	0	0	1.5	1.5	4.2

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar

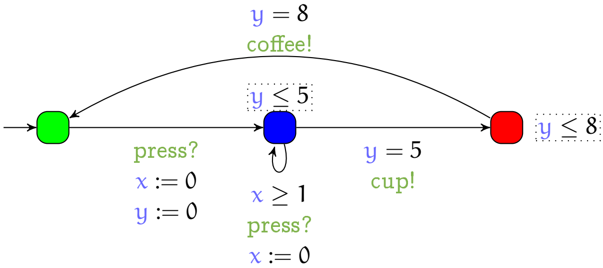


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

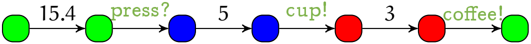
  - Coffee with 2 doses of sugar

x	0	0	1.5	0	2.7	0
y	0	0	1.5	1.5	4.2	4.2

Examples of concrete runs

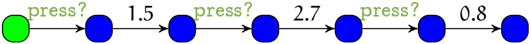


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



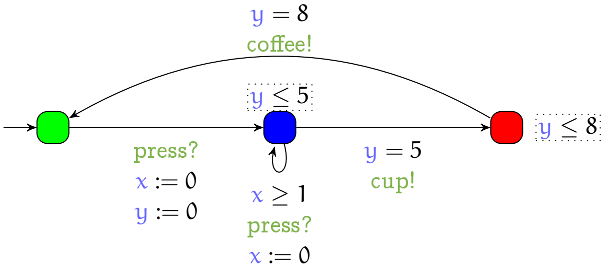
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

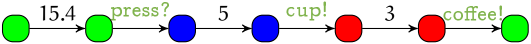


x	0	0	1.5	0	2.7	0	0.8
y	0	0	1.5	1.5	4.2	4.2	5

Examples of concrete runs

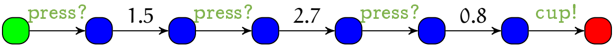


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



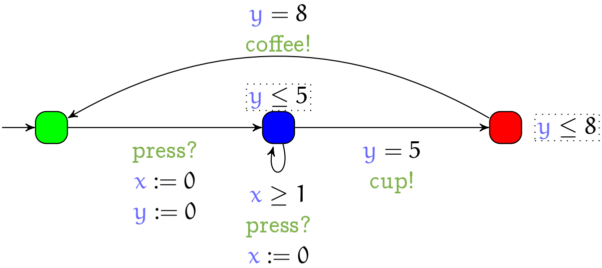
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

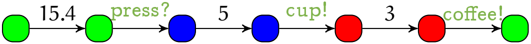


x	0	0	1.5	0	2.7	0	0.8	0.8
y	0	0	1.5	1.5	4.2	4.2	5	5

Examples of concrete runs

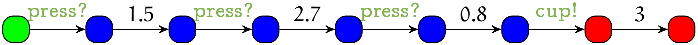


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



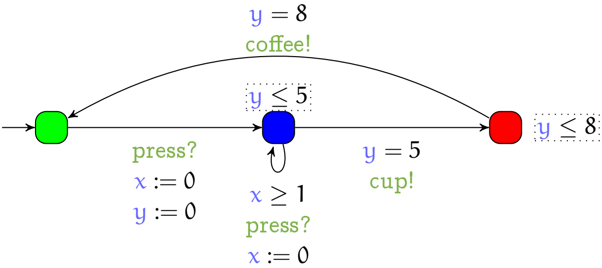
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

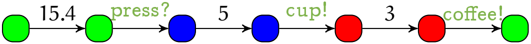


x	0	0	1.5	0	2.7	0	0.8	0.8	3.8
y	0	0	1.5	1.5	4.2	4.2	5	5	8

Examples of concrete runs



- Possible concrete runs for the coffee machine
  - Coffee with no sugar



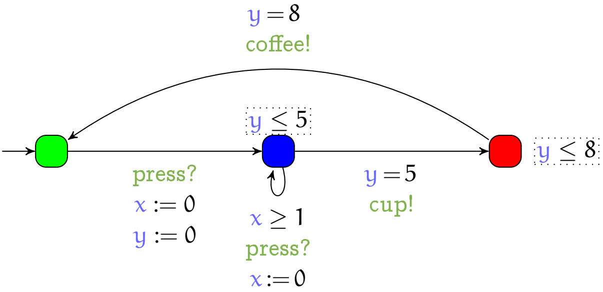
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

x	0	0	1.5	0	2.7	0	0.8	0.8	3.8	3.8
y	0	0	1.5	1.5	4.2	4.2	5	5	8	8

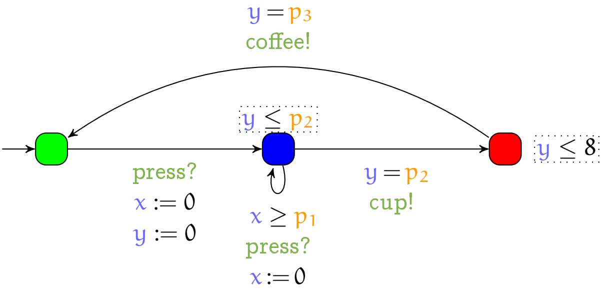
Parametric timed automaton (PTA)

- 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

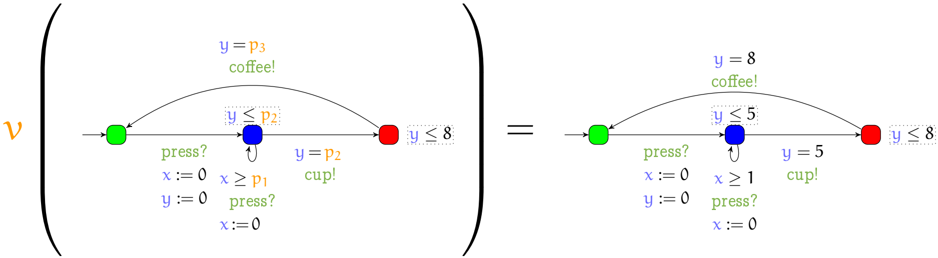


Valuation of a PTA

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

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 all parameters are valuated by v



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

Outline

Objective

Problem

Given a PTA $\mathcal{A}$, synthesize valuations v for which $v(\mathcal{A})$ is deadlock-free.

Objective

Problem

Given a PTA $\mathcal{A}$, synthesize valuations v for which $v(\mathcal{A})$ is deadlock-free.

Deadlock: state in which no discrete action may be taken in some state, even after elapsing some time.

Objective

Problem

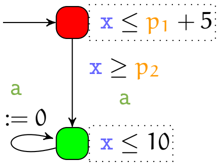
Given a PTA $\mathcal{A}$, synthesize valuations v for which $v(\mathcal{A})$ is deadlock-free.

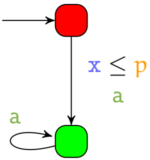
Deadlock: state in which no discrete action may be taken in some state, even after elapsing some time.

The mere emptiness of this valuation set is undecidable for PTAs, and even for subclasses known for other decidability results

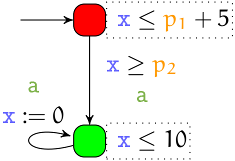
[André and Lime, 2016]

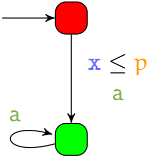
Examples





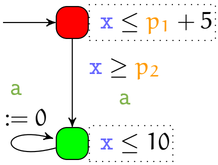
Examples



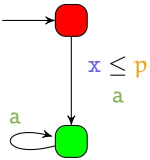


Deadlock-free if
$p_1 + 5 \geq p_2 \leq 10$

Examples



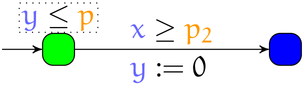
Deadlock-free if $p_1 + 5 \geq p_2 \leq 10$



Deadlock-free for no valuation

Symbolic semantics of PTAs

- Symbolic state $s = (l, Z)$: location + convex polyhedron constraining both clocks and parameters
  [André et al., 2009, Jovanović et al., 2015]
- Convex polyhedra obtained have a special form called parametric zone
  [Hune et al., 2002]



$$Z_0 = \left\{ \begin{array}{ll} & x = y \\ \wedge & 0 \leq y \leq p_1 \\ \wedge & p_1, p_2 \geq 0 \end{array} \right. \qquad Z_1 = \left\{ \begin{array}{ll} & p_2 \leq x - y \leq p_1 \\ \wedge & (p_2 \leq p_1) \\ \wedge & x, y, p_1, p_2 \geq 0 \end{array} \right.$$

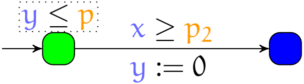
Symbolic semantics of PTAs

- Symbolic state $s = (l, Z)$: location + convex polyhedron constraining both clocks and parameters

  [André et al., 2009, Jovanović et al., 2015]

- Convex polyhedra obtained have a special form called parametric zone

  [Hune et al., 2002]

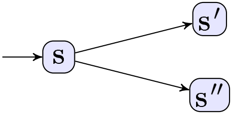


$$Z_0 = \begin{cases} & x = y \\ \wedge & 0 \leq y \leq p_1 \\ \wedge & p_1, p_2 \geq 0 \end{cases} \qquad Z_1 = \begin{cases} & p_2 \leq x - y \leq p_1 \\ \wedge & (p_2 \leq p_1) \\ \wedge & x, y, p_1, p_2 \geq 0 \end{cases}$$

Note: there is a potentially infinite number of symbolic states

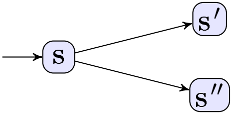
A semi-algorithm for deadlock-existence

$$\mathsf{DSynth}(\mathbf{s}, \mathsf{Passed}) = \Bigg\{$$



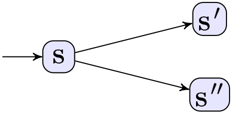
A semi-algorithm for deadlock-existence

$$\mathsf{DSynth}(\mathrm{s}, \mathsf{Passed}) = \begin{cases} \text{return } \underbrace{\text{nothing if state met}}_{} \text{ before} \end{cases}$$



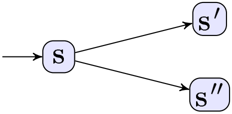
A semi-algorithm for deadlock-existence

$$\mathsf{DSynth}(\mathbf{s}, \mathsf{Passed}) = \begin{cases} \overbrace{\bot \text{ if } \mathbf{s} \in \mathsf{Passed}}^{\text{return nothing if state met before}} \end{cases}$$



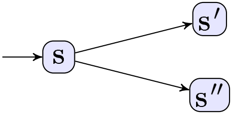
A semi-algorithm for deadlock-existence

$$\mathsf{DSynth}(\mathrm{s}, \mathsf{Passed}) = \begin{cases} \overbrace{\bot \text{ if } \mathrm{s} \in \mathsf{Passed}}^{\text{return nothing if state met before}} \\ \text{otherwise: } \overbrace{\qquad\qquad\qquad\qquad}^{\text{recursive call on the successors}} \\ \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{valuations for which one may get stuck in s}} \end{cases}$$



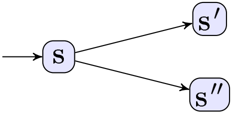
A semi-algorithm for deadlock-existence

$$\mathsf{DSynth}(s, \mathsf{Passed}) = \begin{cases} \overbrace{\bot \text{ if } s \in \mathsf{Passed}}^{\text{return nothing if state met before}} \\[2em] \text{otherwise: } \overbrace{\Big(\bigcup_{s' \in \mathsf{Succ}(s)} \mathsf{DSynth}(s', \mathsf{Passed} \cup \{s\})\Big)}^{\text{recursive call on the successors}} \\[4em] \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{valuations for which one may get stuck in s}} \end{cases}$$



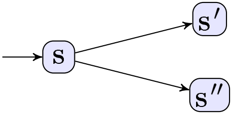
A semi-algorithm for deadlock-existence

$$
\mathsf{DSynth}(s, \mathsf{Passed}) = \begin{cases} \overbrace{\bot \text{ if } s \in \mathsf{Passed}}^{\text{return nothing if state met before}} \\[2em] \text{otherwise: } \overbrace{\left(\bigcup_{s' \in \mathsf{Succ}(s)} \mathsf{DSynth}(s', \mathsf{Passed} \cup \{s\}) \right)}^{\text{recursive call on the successors}} \\[2em] \quad \cup \\[1em] \qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{valuations for which one may get stuck in } s} \end{cases}
$$



A semi-algorithm for deadlock-existence

$$
\mathsf{DSynth}(s, \mathsf{Passed}) = \begin{cases} \overbrace{\bot \text{ if } s \in \mathsf{Passed}}^{\text{return nothing if state met before}} \\[2em] \text{otherwise: } \overbrace{\left(\bigcup_{s' \in \mathsf{Succ}(s)} \mathsf{DSynth}(s', \mathsf{Passed} \cup \{s\})\right)}^{\text{recursive call on the successors}} \\ \underbrace{\cup \left(s_C \setminus \left(\bigcup_{s' \in \mathsf{Succ}(s)} \overbrace{\left(s_C \wedge g(s, s')\swarrow\right)}^{\text{can pass the guard}} \wedge s'_C \downarrow_P) \right) \right) \downarrow_P}_{\text{valuations for which one may get stuck in } s} \end{cases}
$$



A semi-algorithm for deadlock-freeness

$\mathsf{PDFC}(\mathcal{A}) = \neg\mathsf{DSynth}(s_0^{\mathcal{A}}, \emptyset)$

A semi-algorithm for deadlock-freeness

$\mathsf{PDFC}(\mathcal{A}) = \neg\mathsf{DSynth}(s_0^{\mathcal{A}}, \emptyset)$

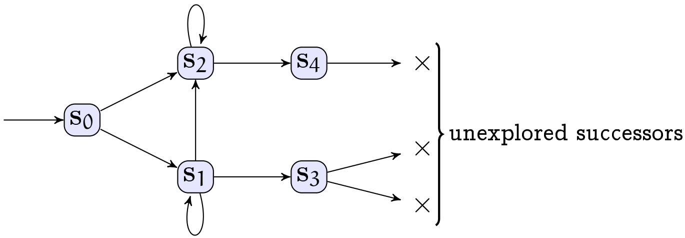
Semi-algorithm: result correct whenever it terminates (which is not guaranteed – undecidable problem)

Outline

A conservative under-approximation BwUS

Idea:

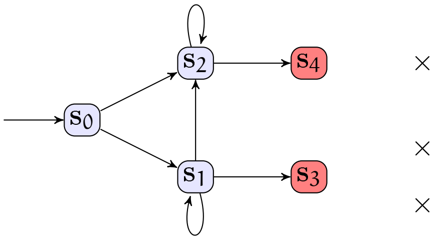
1. Generate a partial state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a backward manner until fixpoint



A conservative under-approximation BwUS

Idea:

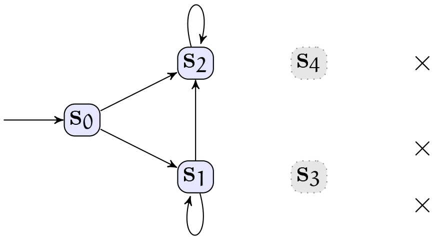
1. Generate a partial state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a backward manner until fixpoint



A conservative under-approximation BwUS

Idea:

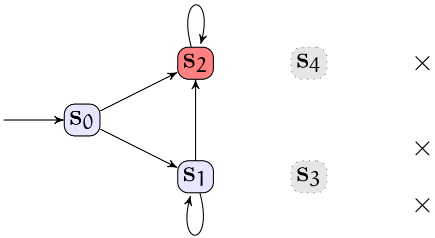
1. Generate a partial state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a backward manner until fixpoint



A conservative under-approximation BwUS

Idea:

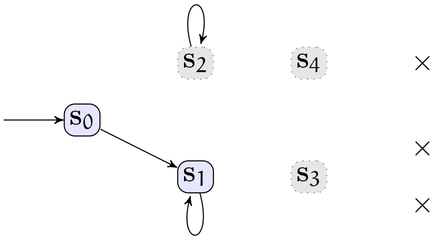
1. Generate a partial state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a backward manner until fixpoint



A conservative under-approximation BwUS

Idea:

1. Generate a partial state space
2. Consider unknown states as unsafe (deadlocked)
3. Recompute the deadlock-freeness constraint in a backward manner until fixpoint



Outline

Implementation in IMITATOR

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

Under continuous development since 2008 [André et al., 2012]

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

GPL Free Software

Free as in Freedom

www.imitator.fr

Summary of the experiments

Case study	$\lvert\mathcal{A}\rvert$	$\lvert X\rvert$	$\lvert P\rvert$	States	PDFC	BwUS	K	Soundness
Fig. 1a	1	1	2	3	0.012	-	nncc	exact
Fig. 1b	1	1	1	2	0.005	-	$\perp$	exact
and–or circuit	4	4	4	5,265	TO	171	$[\text{nncc}^-, \text{nncc}^+]$	under/over-app
coffee machine 1	1	2	3	9,042	TO	8.4	$[\text{nncc}^-, \text{nncc}^+]$	under/over-app
coffee machine 2	2	3	3	51	0.198	-	nncc	exact
CSMA/CD protocol	3	3	3	38	0.105	-	$\perp$	exact
flip-flop circuit	6	5	2	20	0.093	-	$\perp$	exact
nuclear plant	1	2	4	13	0.014	-	nncc	exact
RCP protocol	5	6	5	2,091	10.63	-	$\perp$	exact
SIMOP	5	8	2	22,894	TO	121	nncc	over-app
Train controller	1	2	3	11	0.025	-	nncc	exact
WFAS	3	4	2	14,614	TO	69.1	$[\text{nncc}^-, \text{nncc}^+]$	under/over-app

Outline

Conclusion and perspectives

Conclusion

- Semi-algorithm to guarantee the absence of deadlocks in timed automata with parametric timing constants
- Under-approximation with guaranteed termination

Perspectives

- Theory: decidable subclasses for deadlock-checking?
- Practice: efficient computation (multi-core)
- Reusability: Apply the idea behind BwUS to other algorithms

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

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

André, É. and Lime, D. (2016).
Liveness in L/U-parametric timed automata.
Submitted.

Henzinger, T. A., Nicollin, X., Sifakis, J., and Yovine, S. (1994).
Symbolic model checking for real-time systems.
Information and Computation, 111(2):193–244.

References II

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.
Transactions on Software Engineering, 41(5):445–461.

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