ATVA 2019

2019年10月29日
台灣，台北

# Parametric Timed Model Checking for Guaranteeing Timed Opacity

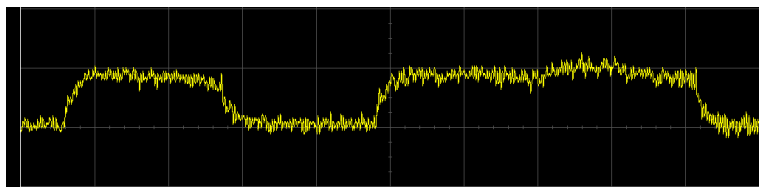
Étienne André[1,2,3] and Sun Jun[4]

[1] Université de Lorraine, CNRS, Inria, LORIA, Nancy, France
[2] National Institute of Informatics, Japan
[3] JFLI, UMI CNRS, Tokyo, Japan
[4] Singapore Management University

Supported by the ANR national research program PACS (ANR-14-CE28-0002), and by JST ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603).

# Context: side-channel attacks

- Threats to a system using non-algorithmic weaknesses
  - Cache attack
  - Electromagnetic attacks
  - Power attacks



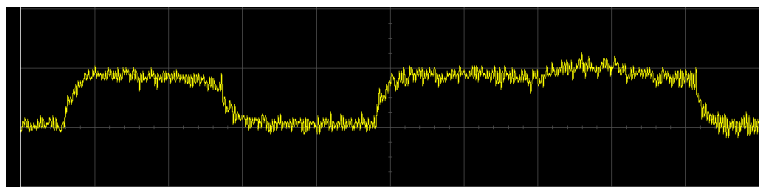
  - Acoustic attacks
  - Timing attacks
  - etc.

- Example
  - Number of pizzas (and order time) ordered by the white house prior to major war announcements [1]

[1] `http://home.xnet.com/~warinner/pizzacites.html`

# Context: side-channel attacks

- Threats to a system using non-algorithmic weaknesses
  - Cache attack
  - Electromagnetic attacks
  - Power attacks



  - Acoustic attacks
  - Timing attacks
  - etc.

- Example
  - Number of pizzas (and order time) ordered by the white house prior to major war announcements [1]

[1] `http://home.xnet.com/~warinner/pizzacites.html`

# Context: timing attacks

- Principle: deduce private information from timing data (execution time)

Issues:

- May depend on the implementation (or, even worse, be introduced by the compiler)
- A relatively trivial solution: make the program last always its maximum execution time
  Drawback: loss of efficiency

$\rightsquigarrow$ Non-trivial problem

# A simple example of timing attack

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd, len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5     return false
6 done
7 return true
```

# A simple example of timing attack

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd, len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5     return false
6 done
7 return true
```

| pwd | c | h | o | u | d | o | u | f | u |
|---|---|---|---|---|---|---|---|---|---|
| attempt | c | h | e | e | s | e | | | |

Execution time:

# A simple example of timing attack

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd, len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5     return false
6 done
7 return true
```

| pwd | c | h | o | u | d | o | u | f | u |
|---|---|---|---|---|---|---|---|---|---|
| attempt | c | h | e | e | s | e | | | |

Execution time: $\epsilon$

# A simple example of timing attack

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd, len(attempt)) − 1 do
4     if pwd[i] =/= attempt[i] then
5     return false
6 done
7 return true
```

| pwd | c | h | o | u | d | o | u | f | u |
|---|---|---|---|---|---|---|---|---|---|
| attempt | c | h | e | e | s | e | | | |

Execution time: $\epsilon + \epsilon$

# A simple example of timing attack

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd, len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5     return false
6 done
7 return true
```

| pwd | c | h | o | u | d | o | u | f | u |
|---|---|---|---|---|---|---|---|---|---|
| attempt | c | h | e | e | s | e | | | |

Execution time: $\epsilon + \epsilon + \epsilon$

# A simple example of timing attack

```
1 # input pwd    : Real password
2 # input attempt: Tentative password
3 for i = 0 to min(len(pwd, len(attempt)) - 1 do
4     if pwd[i] =/= attempt[i] then
5     return false
6 done
7 return true
```

| pwd | c | h | o | u | d | o | u | f | u |
|---|---|---|---|---|---|---|---|---|---|
| attempt | c | h | e | e | s | e | | | |

Execution time: $\epsilon + \epsilon + \epsilon$

- Problem: The execution time is proportional to the number of consecutive correct characters from the beginning of `attempt`

# Outline

# Informal problems

Question: can we exhibit secure execution times?

**Time-opacity computation**

Exhibit execution times for which it is not only possible to infer information on the internal behavior

# Informal problems

Question: can we exhibit secure execution times?

## Time-opacity computation

Exhibit execution times for which it is not only possible to infer information on the internal behavior

Further question: can we also tune internal timing constants to make the system resisting to timing attacks?

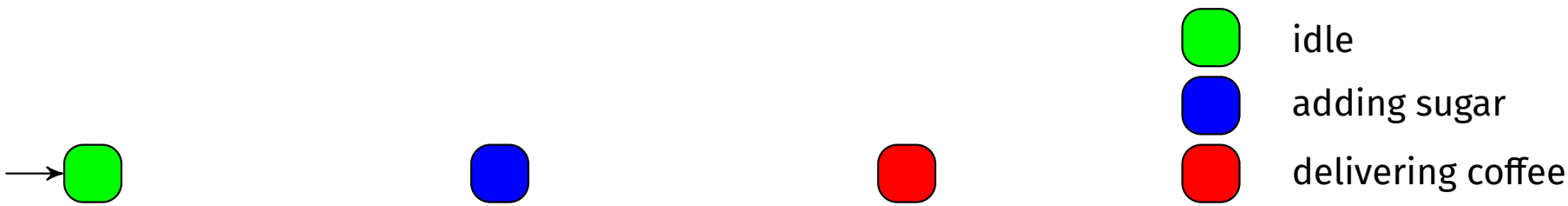
## Time-opacity synthesis

Exhibit execution times and internal timing constants for which it is not only possible to infer information on the internal behavior

# Outline

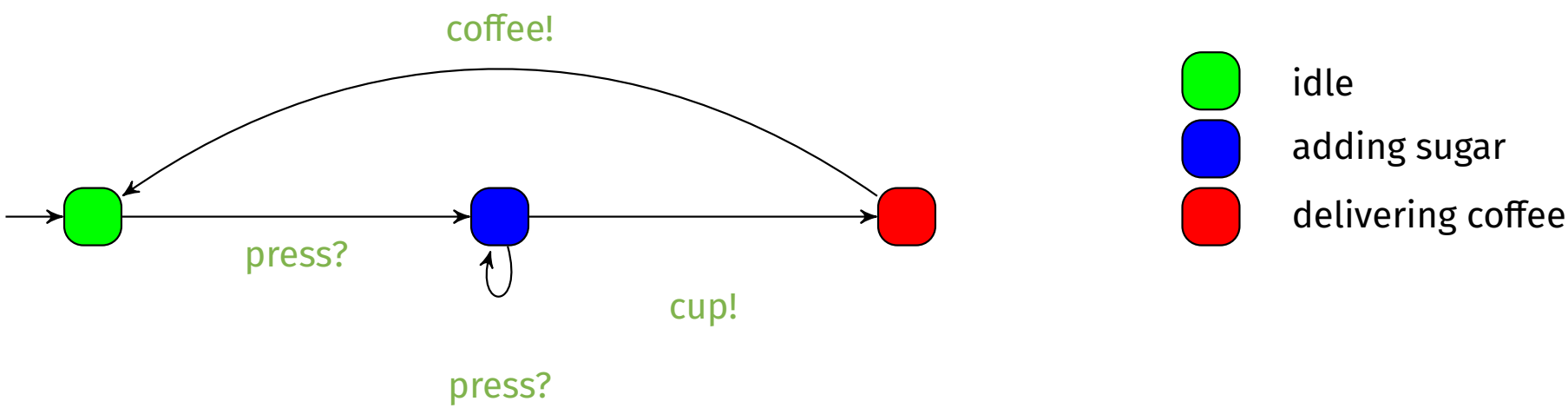
# Timed automaton (TA)

- Finite state automaton (sets of locations)



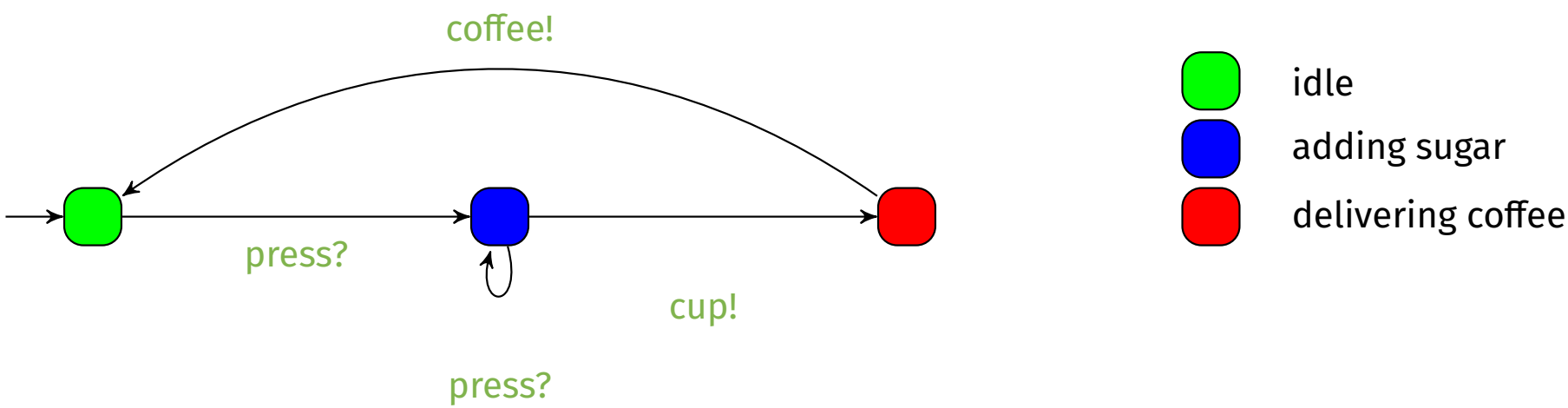
# Timed automaton (TA)

- Finite state automaton (sets of locations and actions)



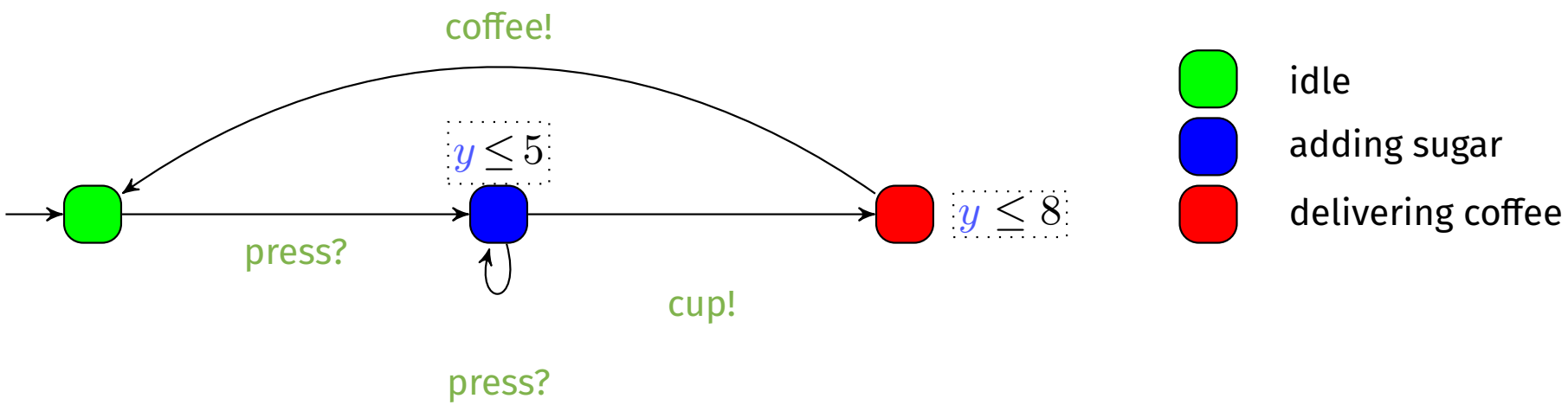
# Timed automaton (TA)

- 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)

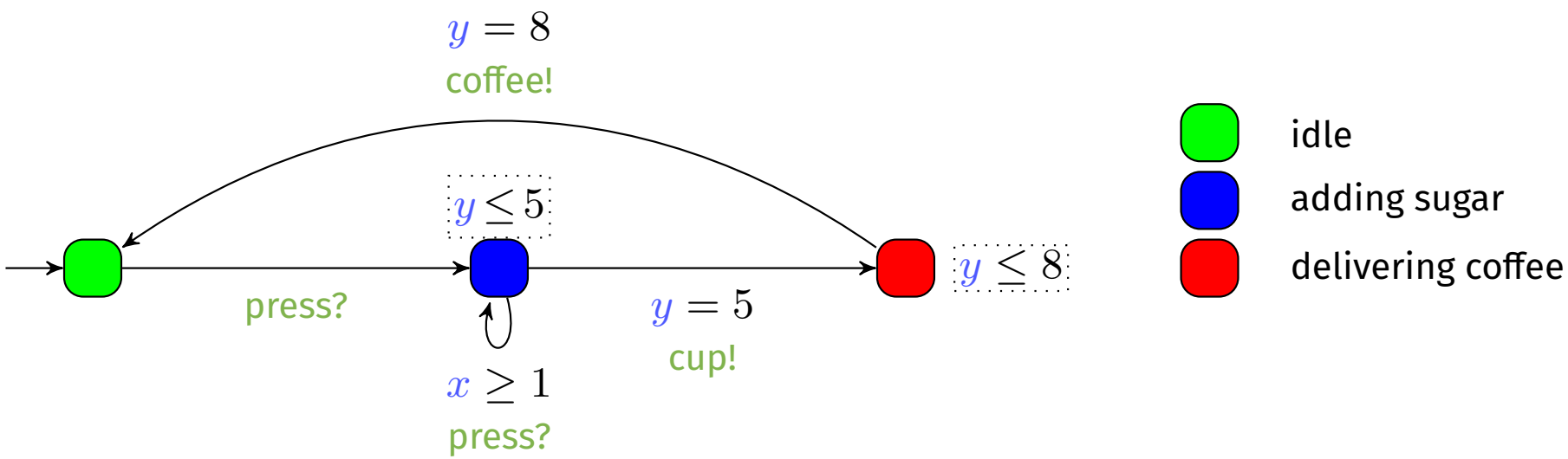
- Finite state automaton (sets of locations and actions) augmented with a set $X$ of clocks [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate
  - Can be compared to integer constants in invariants
- Features
  - Location invariant: property to be verified to stay at a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

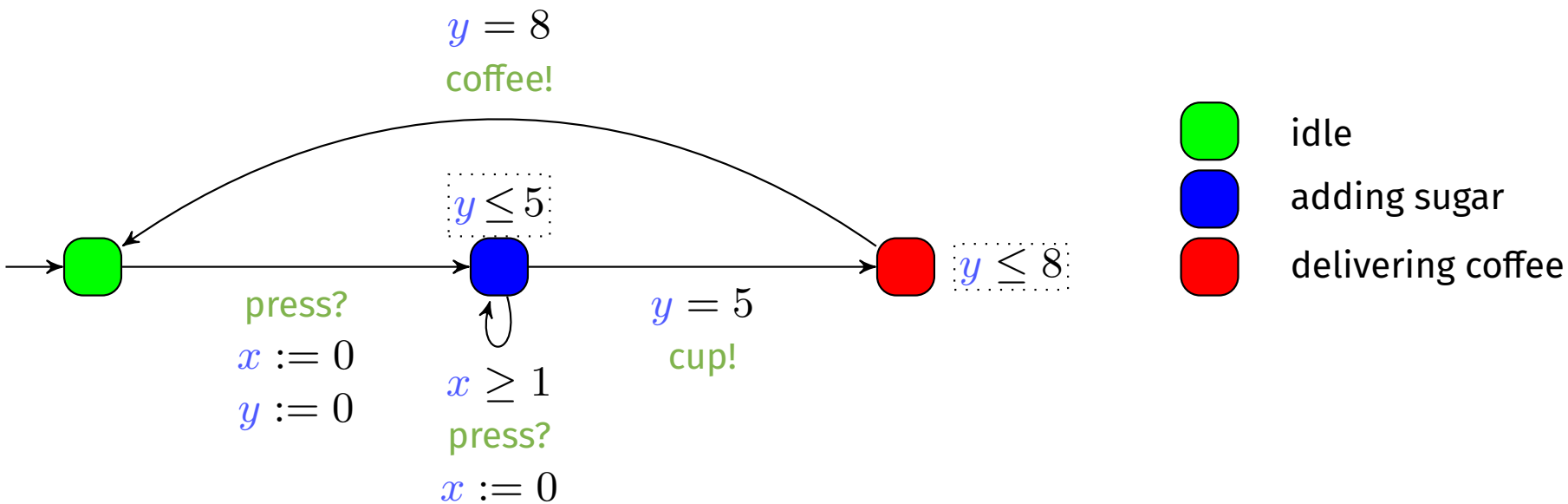
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition



# 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



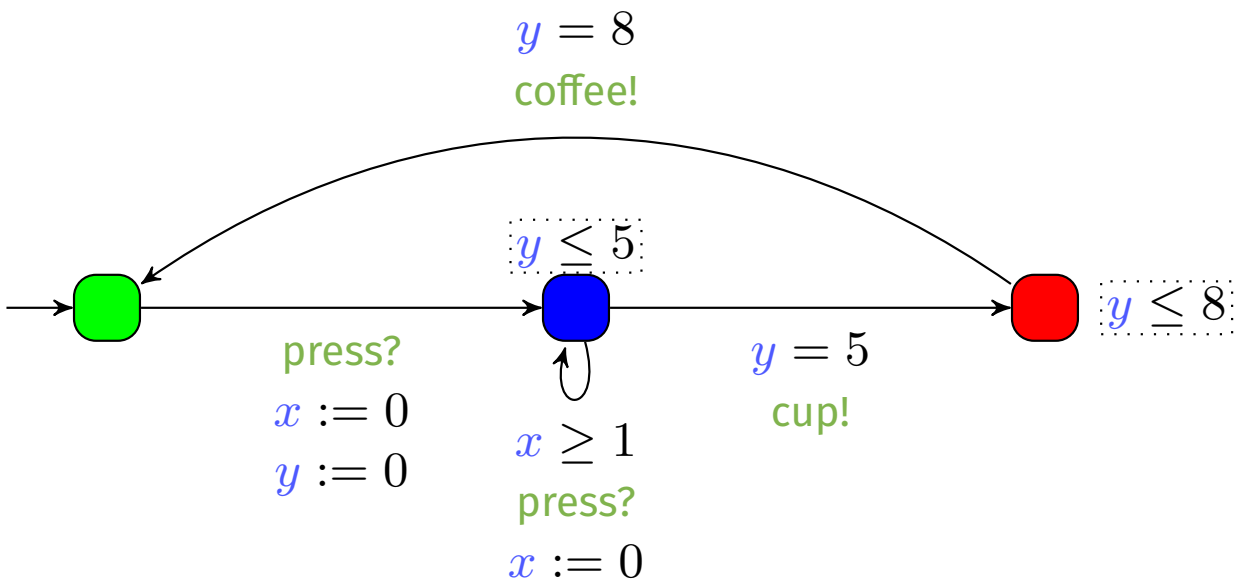
# Concrete semantics of timed automata

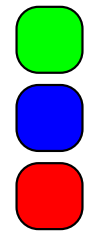
- Concrete state of a TA: pair $(\ell, w)$, where
  - $\ell$ is a location,
  - $w$ is a valuation of each clock

  Example: $\left(\text{  }, \begin{pmatrix} x=1.2 \\ y=3.7 \end{pmatrix}\right)$

- Concrete run: alternating sequence of concrete states and actions or time elapse

# The most critical system: The coffee machine



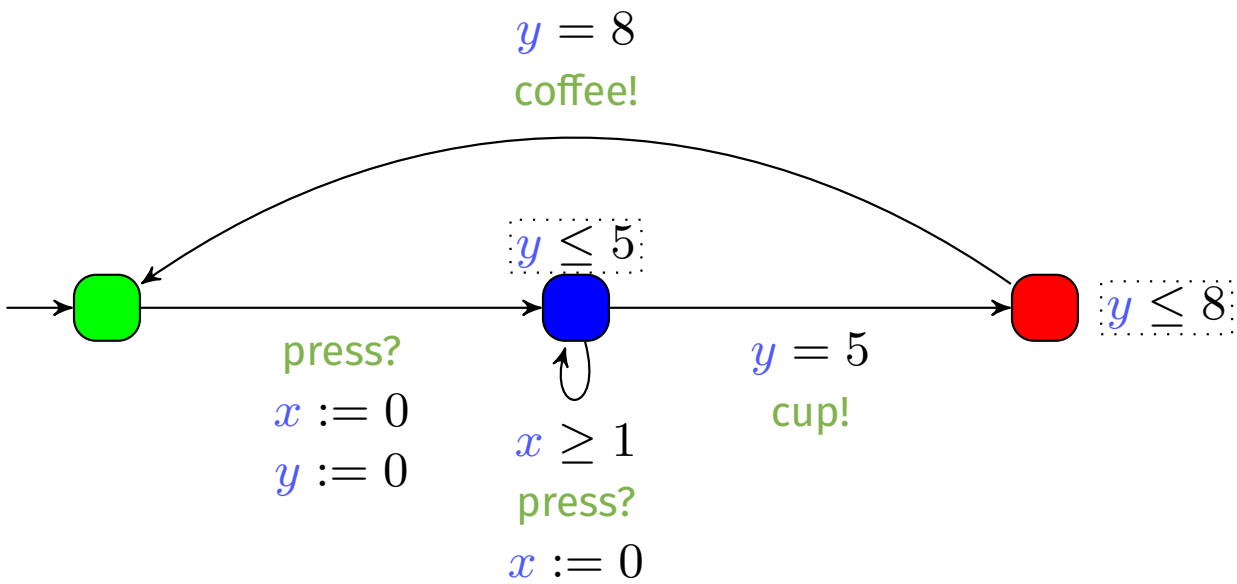



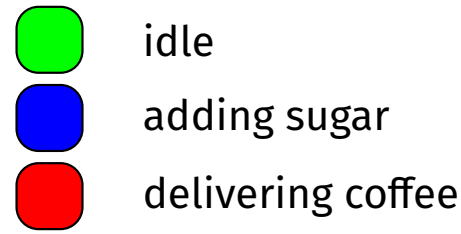
idle

adding sugar

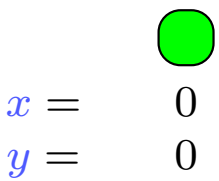
delivering coffee

# The most critical system: The coffee machine

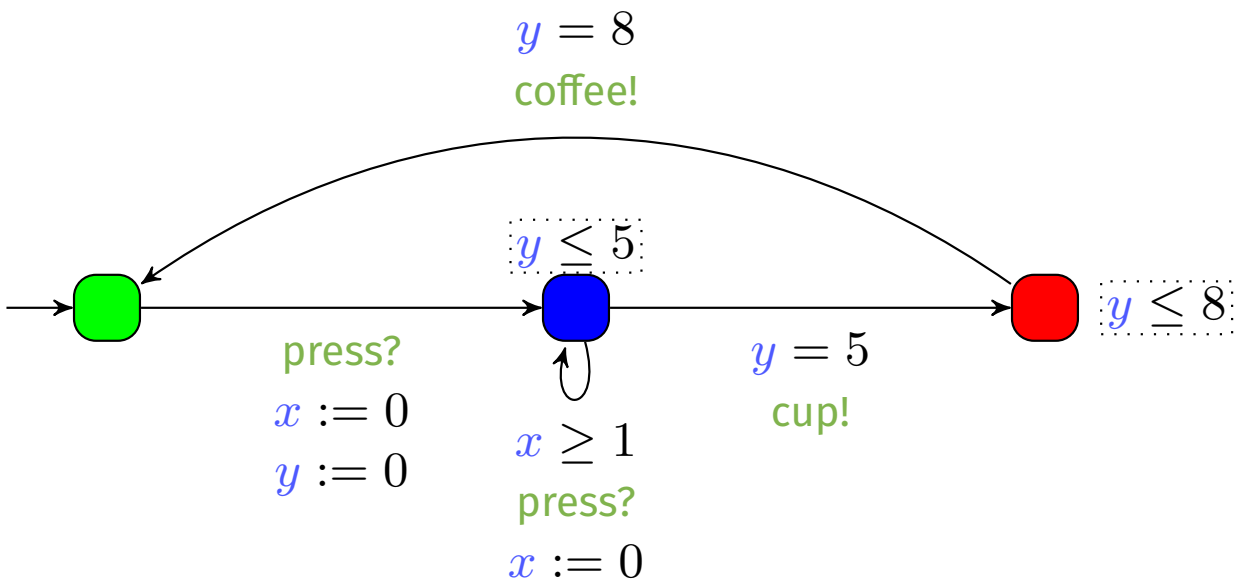


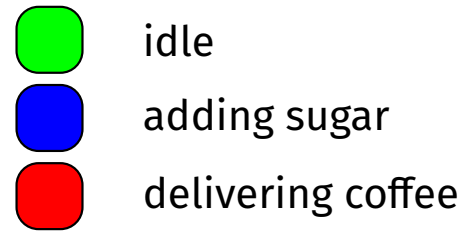




- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

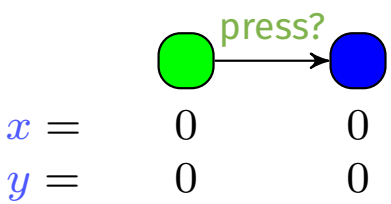


# The most critical system: The coffee machine

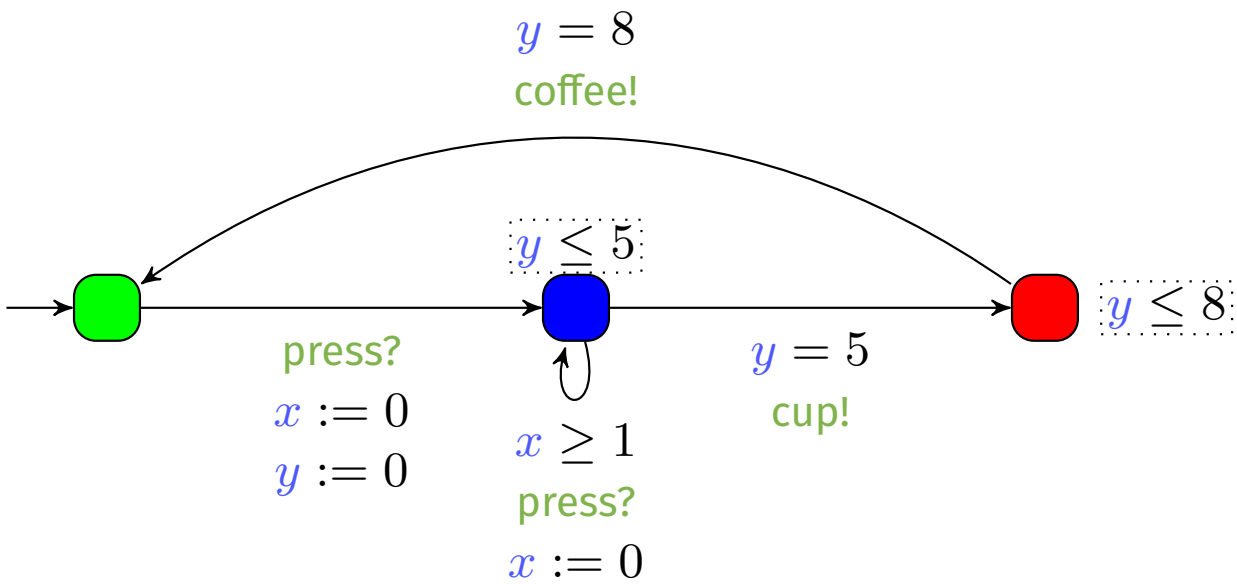


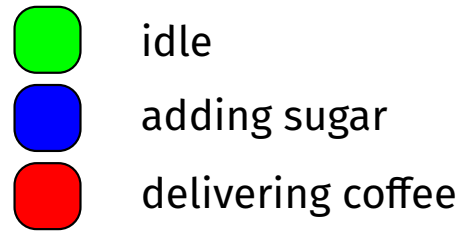




- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

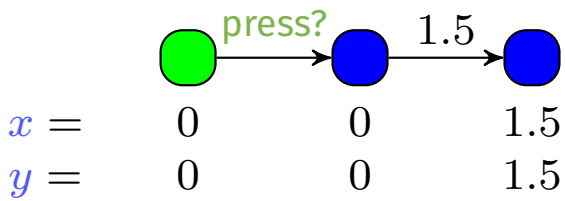


# The most critical system: The coffee machine

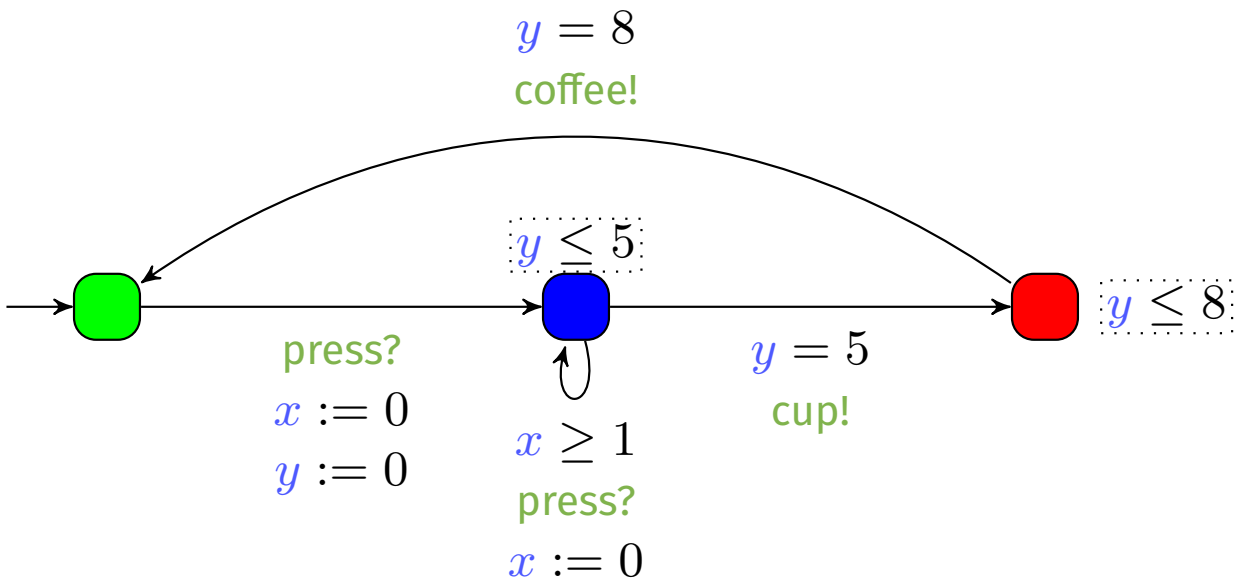


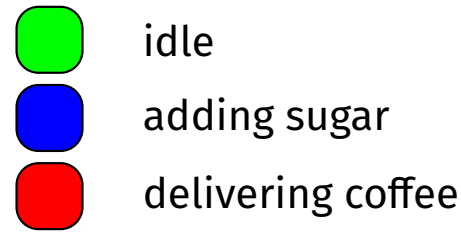




- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

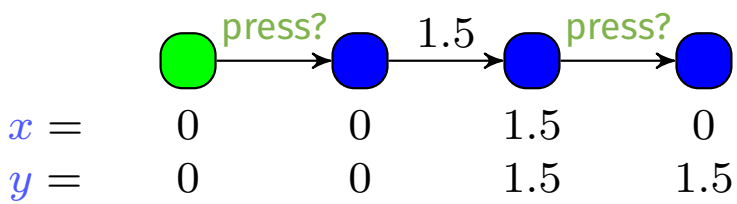


# The most critical system: The coffee machine

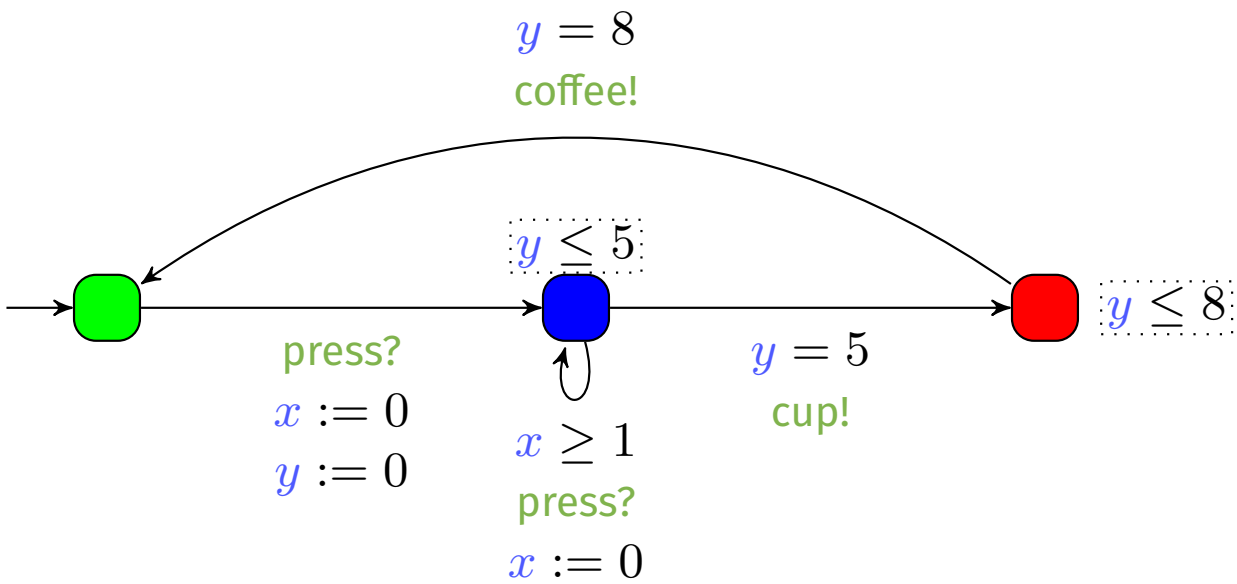


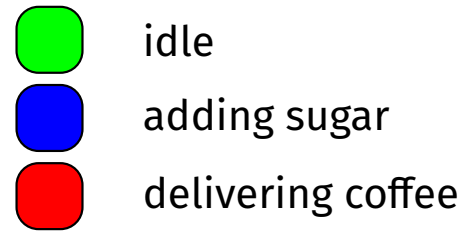




- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

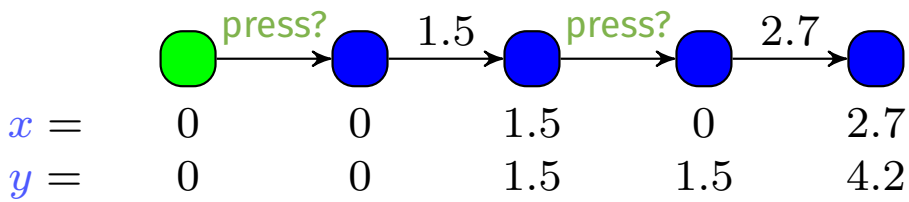


# The most critical system: The coffee machine

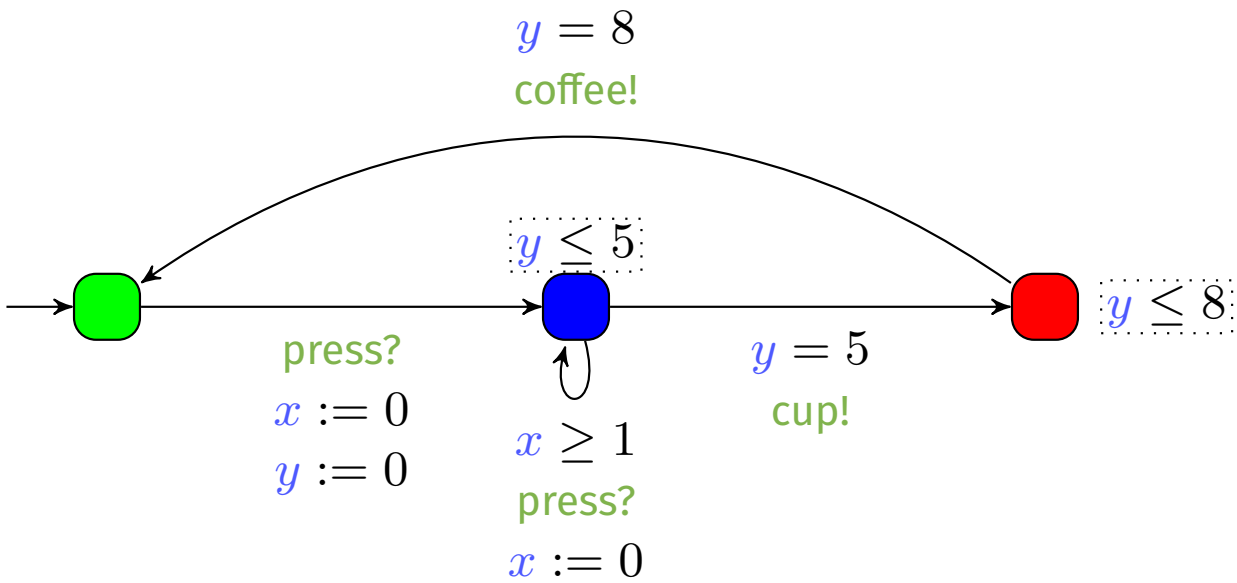


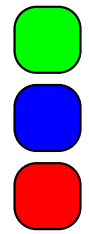




- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



# The most critical system: The coffee machine



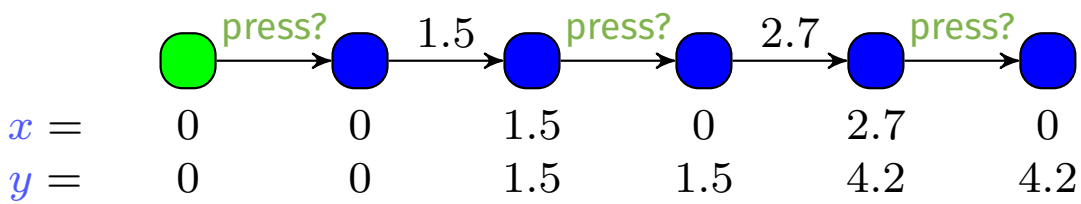



idle

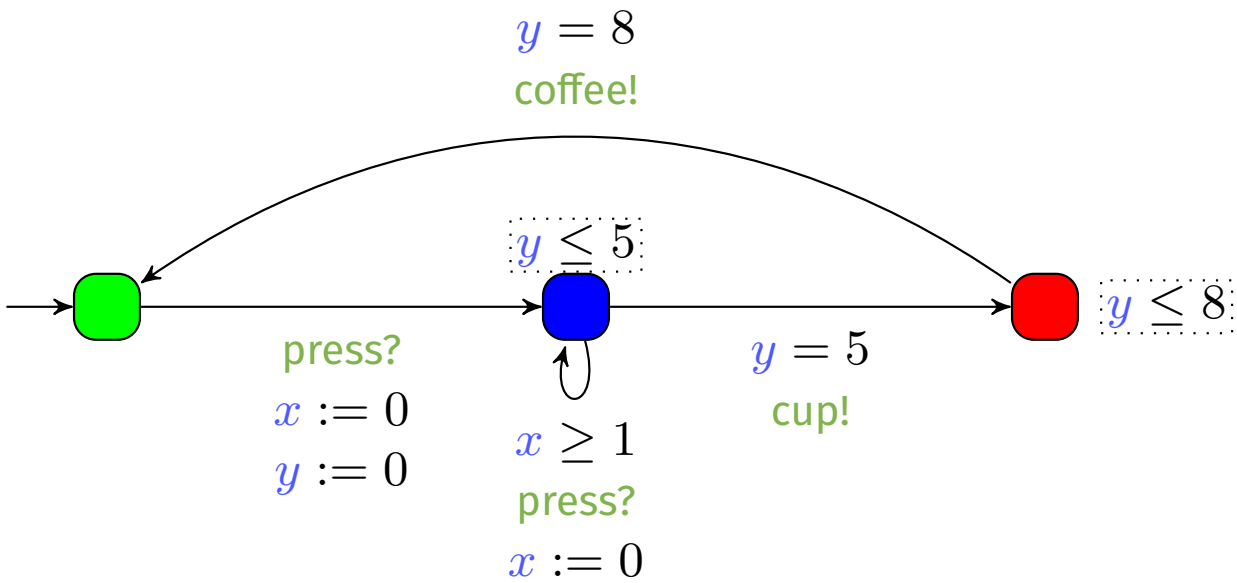
adding sugar

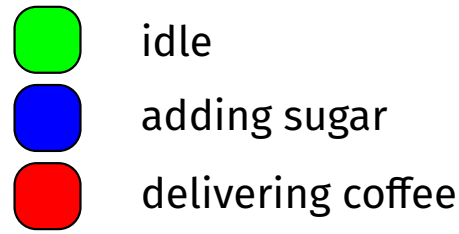
delivering coffee

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

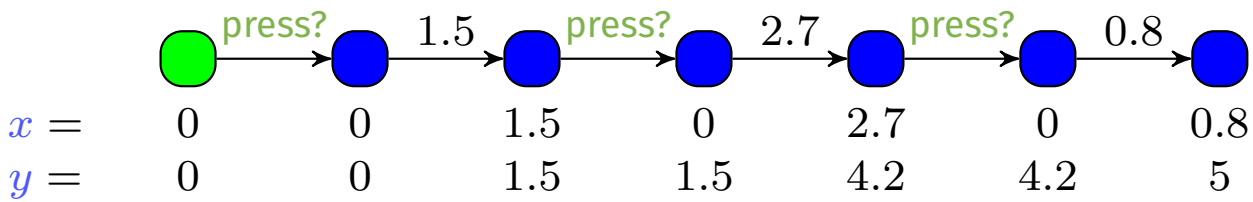


# The most critical system: The coffee machine

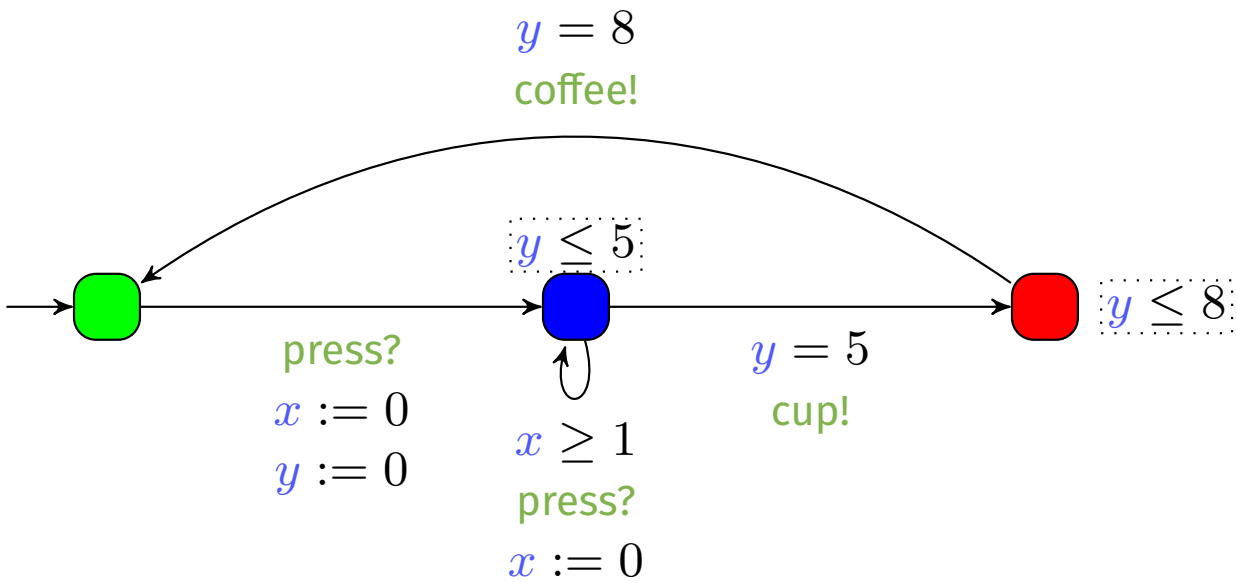


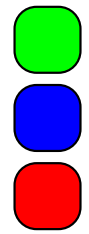




- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



# The most critical system: The coffee machine



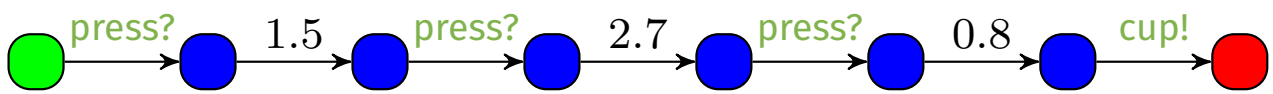



idle

adding sugar

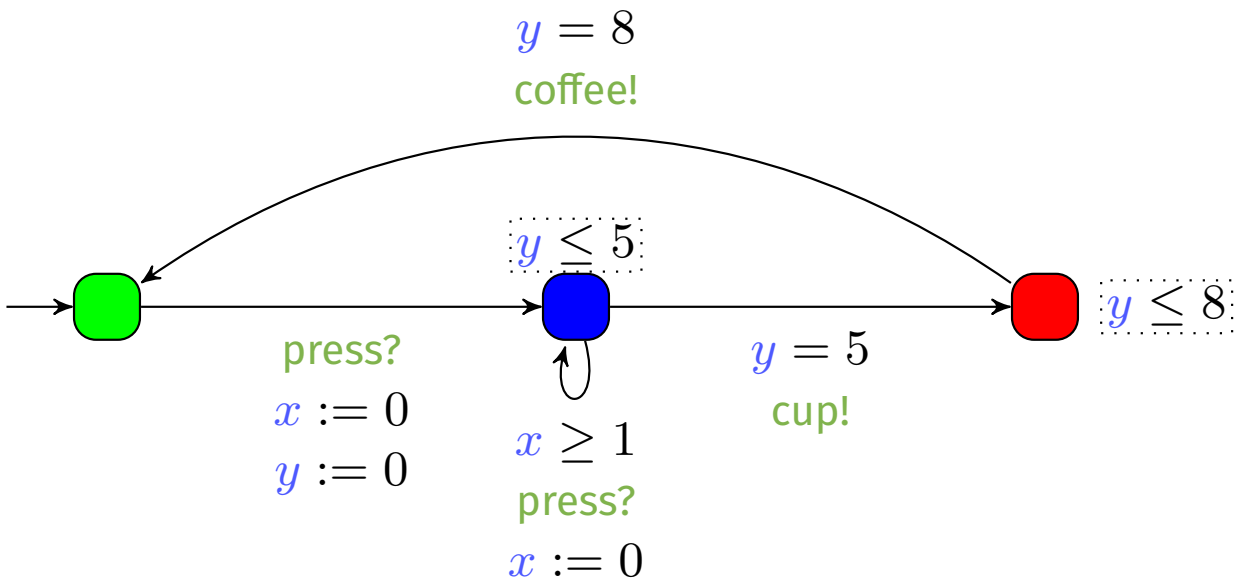
delivering coffee

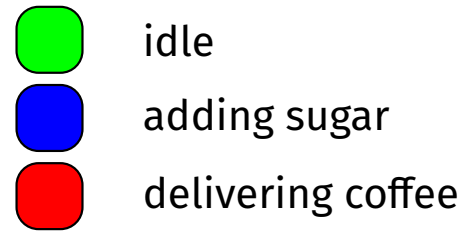
- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar



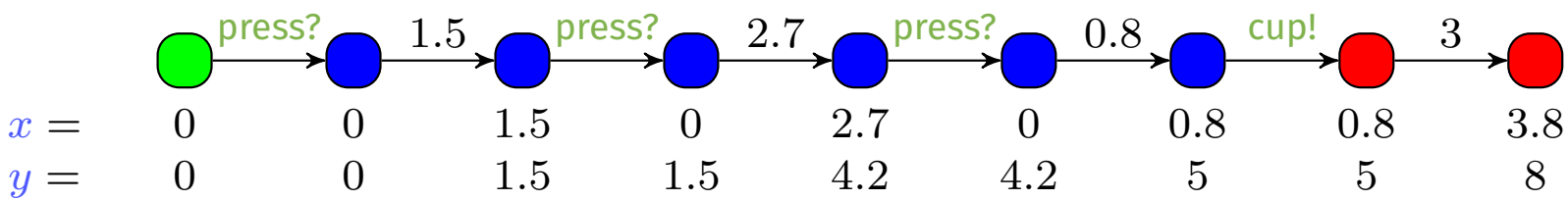

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $x =$ | 0 | 0 | 1.5 | 0 | 2.7 | 0 | 0.8 | 0.8 |
| $y =$ | 0 | 0 | 1.5 | 1.5 | 4.2 | 4.2 | 5 | 5 |

# The most critical system: The coffee machine

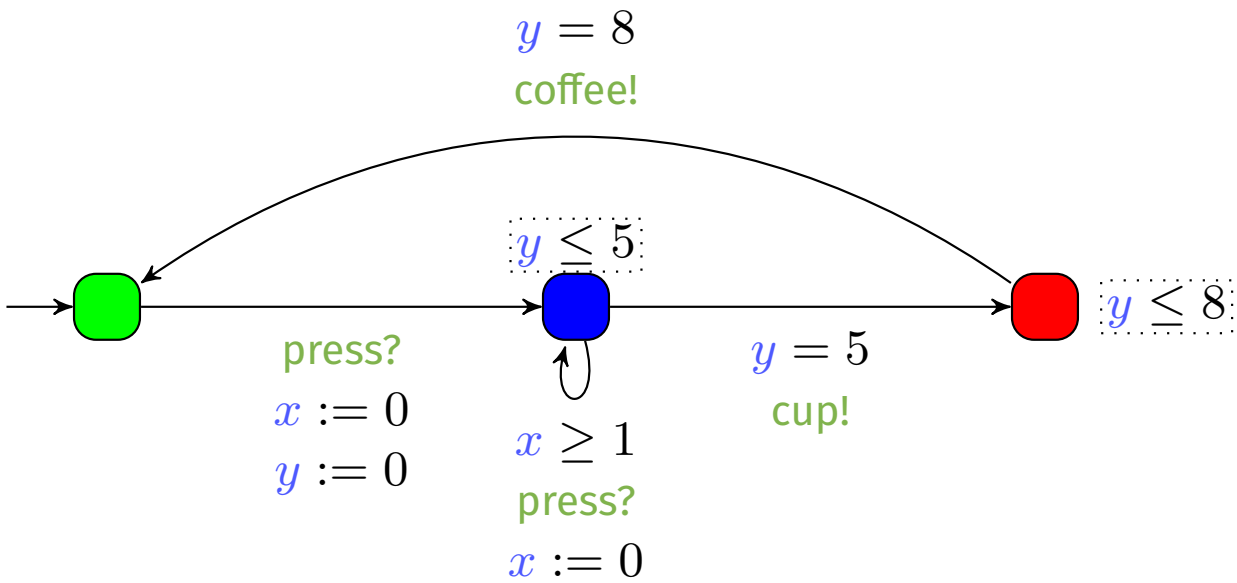


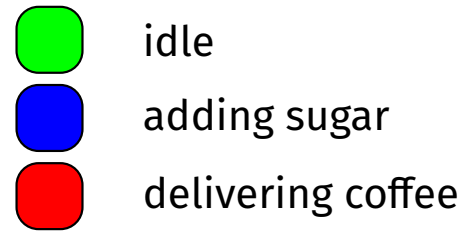




- Example of concrete run for the coffee machine
  - Coffee with 2 doses of sugar

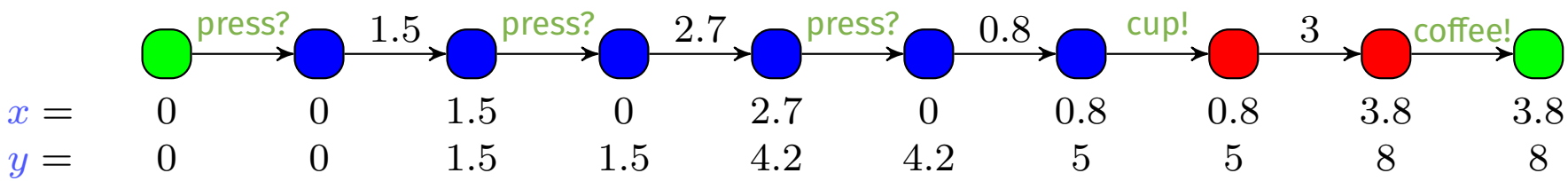


# The most critical system: The coffee machine




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

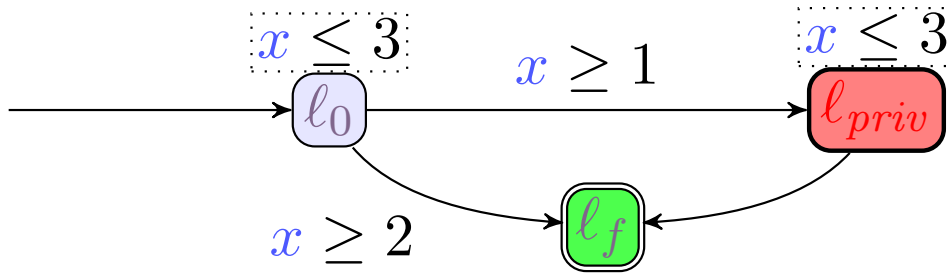


# Outline

# Formalization

Hypotheses:

- A start location $\ell_0$ and an end location $\ell_f$
- A special private location $\ell_{priv}$




## Definition (timed opacity)

The system is opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$ if

1. for any run to $\ell_f$ of duration $d$ passing by $\ell_{priv}$, there exists another run to $\ell_f$ of duration $d$ *not* passing by $\ell_{priv}$, and
2. conversely

# Problem 1: timed-opacity computation

## Timed-opacity computation problem

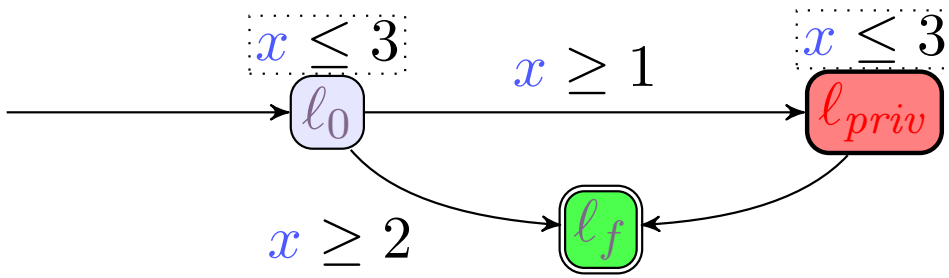
Find durations $d$ ("execution times") of runs from $\ell_0$ to $\ell_f$ such that the system is opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

# Problem 1: timed-opacity computation

## Timed-opacity computation problem

Find durations $d$ ("execution times") of runs from $\ell_0$ to $\ell_f$ such that the system is opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

Example:

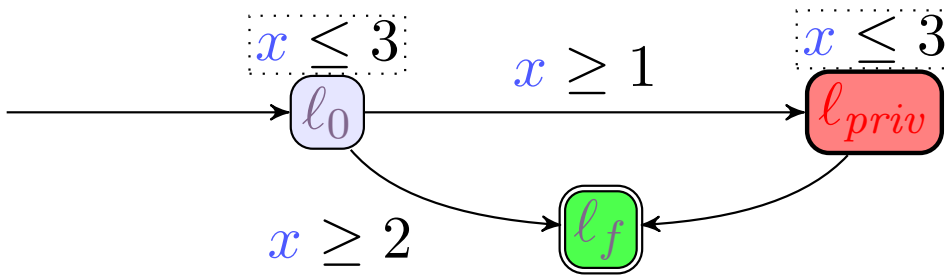


# Problem 1: timed-opacity computation

## Timed-opacity computation problem

Find durations $d$ ("execution times") of runs from $\ell_0$ to $\ell_f$ such that the system is opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

Example:




Here, only computations with durations $d \in [2, 3]$ are opaque

# Problem 1b: timed-opacity

**Definition (Timed-opacity)**

A system is timed-opaque if the answer to the timed-opacity computation problem is the set of all possible execution times for this system.

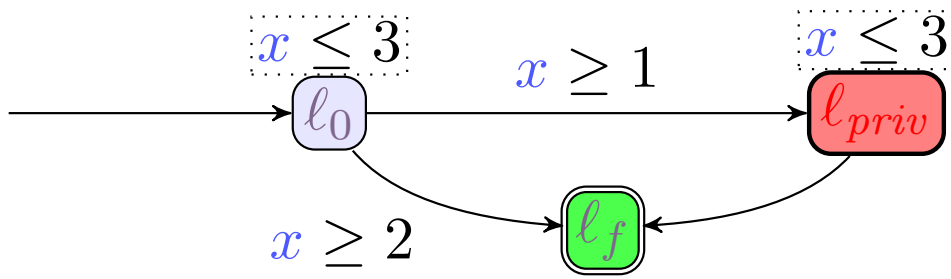
Intuition: is a system timed-opaque for all its execution times?

# Problem 1b: timed-opacity

## Definition (Timed-opacity)

A system is timed-opaque if the answer to the timed-opacity computation problem is the set of all possible execution times for this system.

Intuition: is a system timed-opaque for all its execution times?

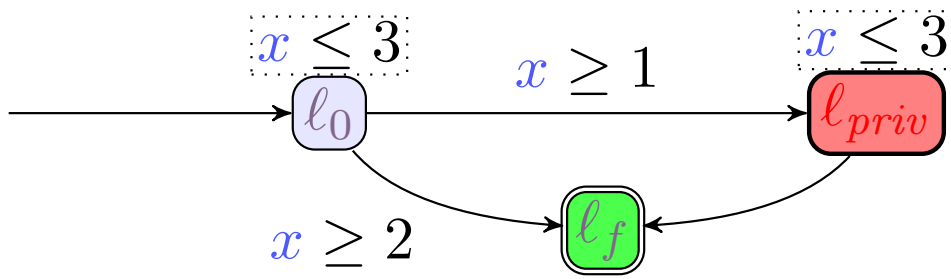


# Problem 1b: timed-opacity

## Definition (Timed-opacity)

A system is timed-opaque if the answer to the timed-opacity computation problem is the set of all possible execution times for this system.

Intuition: is a system timed-opaque for all its execution times?




Here, only computations with durations $d \in [2, 3]$ are opaque

All execution times: $[1, 3]$

$\rightsquigarrow$ not timed-opaque

# Timed-opacity computation can be achieved

**Theorem (Computability of timed-opacity)**

*The answer to the timed-opacity computation problem can be effectively be computed in the form of a finite union of intervals*

Proof: based on the region graph (see paper)

Exact complexity: unproved (EXPSACE upper bound proved, but exponential hardness seems likely)

Remark: to be put in perspective with [Cassez, 2009]

- undecidability for a less expressive class, for a stronger notion of opacity

# Outline

# Towards a more abstract framework…

Problems

- Can we tune some timing constants to guarantee opacity?
- Verification for one set of constants does not usually guarantee the correctness for other values
- Robustness [Bouyer et al., 2013]: What happens if $50$ is implemented with $49.99$?

# Towards a more abstract framework…

Problems

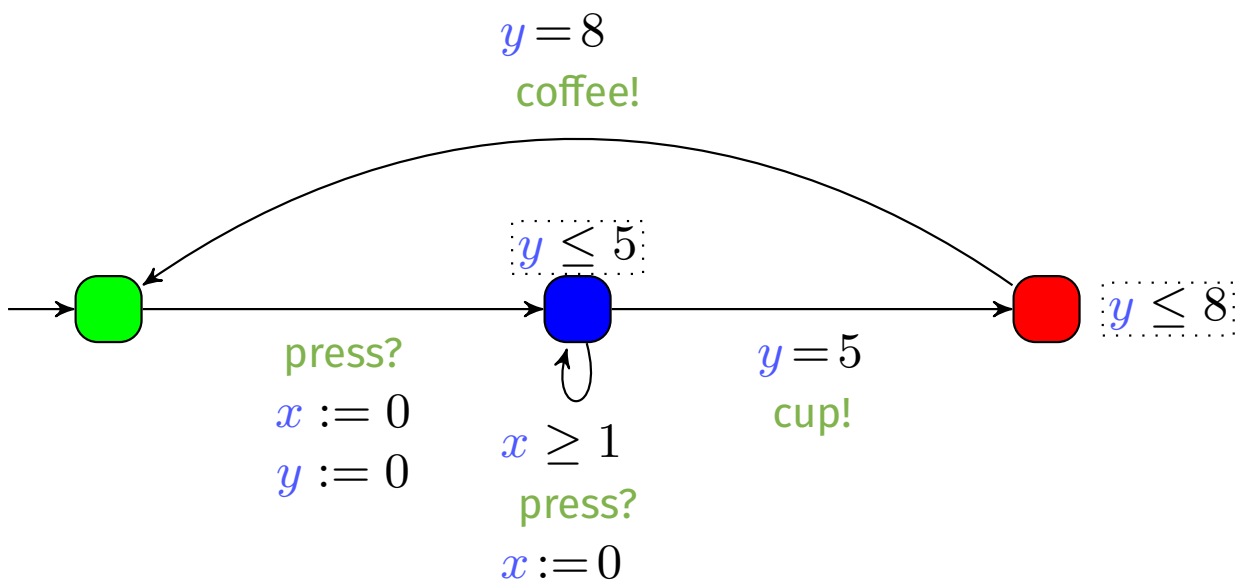
- Can we tune some timing constants to guarantee opacity?
- Verification for one set of constants does not usually guarantee the correctness for other values
- Robustness [Bouyer et al., 2013]: What happens if $50$ is implemented with $49.99$?

A solution:

- Parameter synthesis
  - Consider that timing constants are unknown constants (parameters)

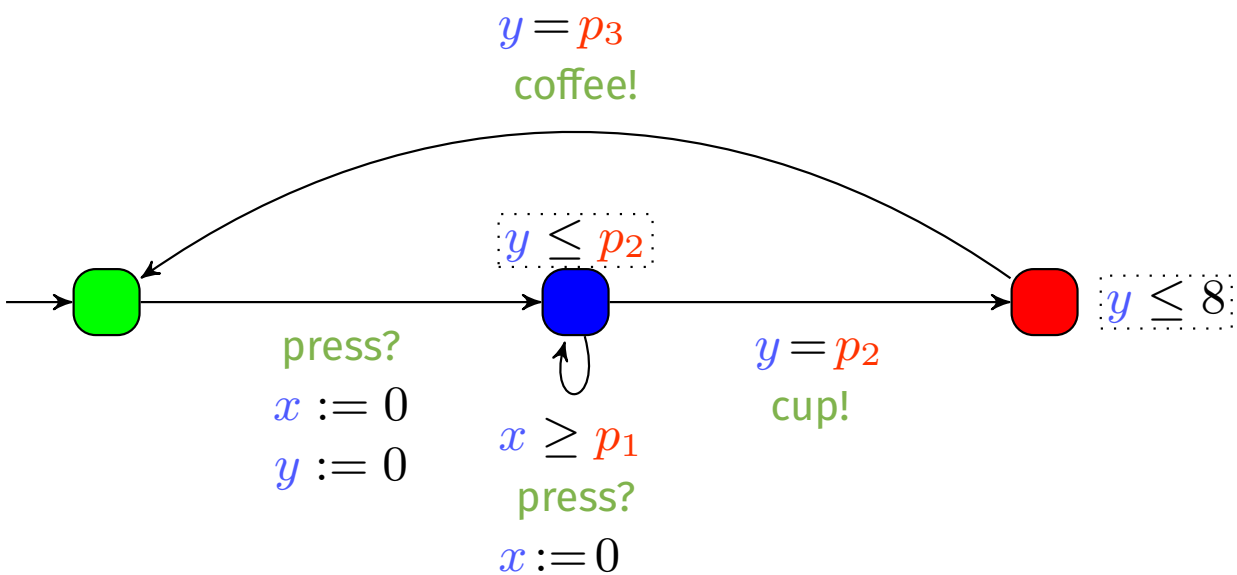
# Parametric Timed Automaton (PTA)

- 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

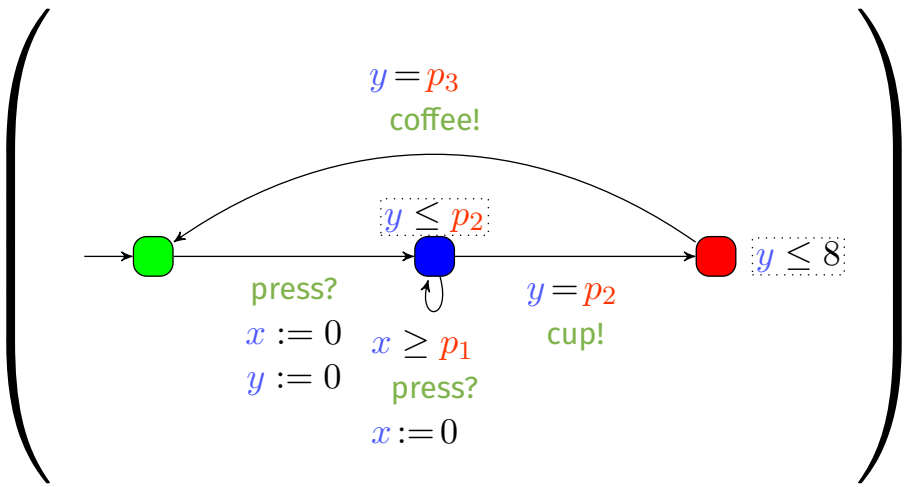


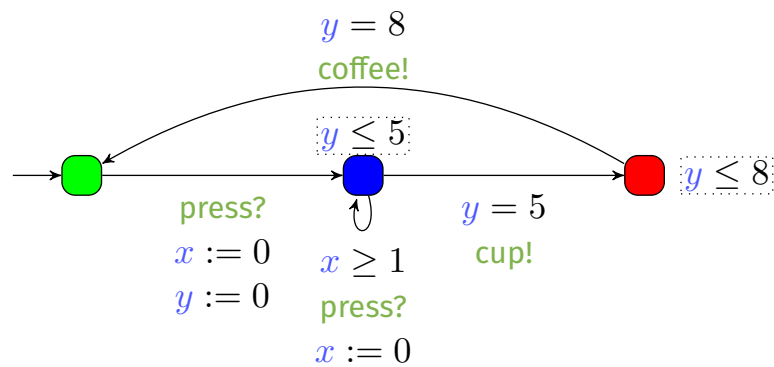
# Notation: Valuation of a PTA

- 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 $v(p)$

$v$ 

$=$ 


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

# Problem 2: timed-opacity synthesis

## Timed-opacity synthesis problem

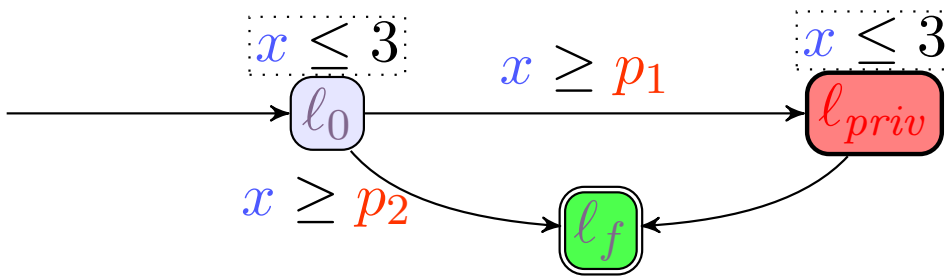
Find parameter valuations $v$ and durations $d$ ("execution times") of runs of $v(\mathcal{A})$ from $\ell_0$ to $\ell_f$ such that the system is opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

# Problem 2: timed-opacity synthesis

## Timed-opacity synthesis problem

Find parameter valuations $v$ and durations $d$ ("execution times") of runs of $v(\mathcal{A})$ from $\ell_0$ to $\ell_f$ such that the system is opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

Example:

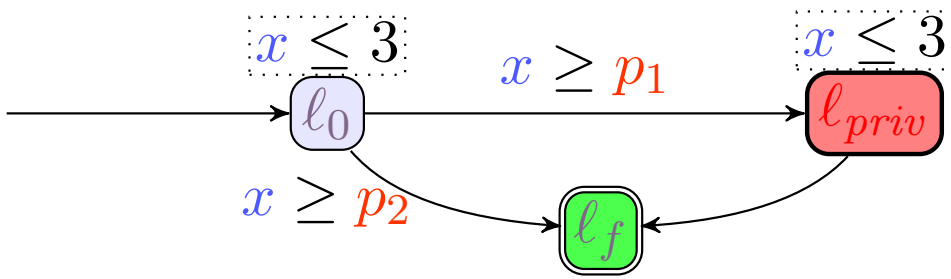


# Problem 2: timed-opacity synthesis

## Timed-opacity synthesis problem

Find parameter valuations $v$ and durations $d$ ("execution times") of runs of $v(\mathcal{A})$ from $\ell_0$ to $\ell_f$ such that the system is opaque w.r.t. $\ell_{priv}$ on the way to $\ell_f$

Example:




Expected result:

$p_1 \leq 3 \wedge p_2 \leq 3 \wedge d \in [p_2, 3]$

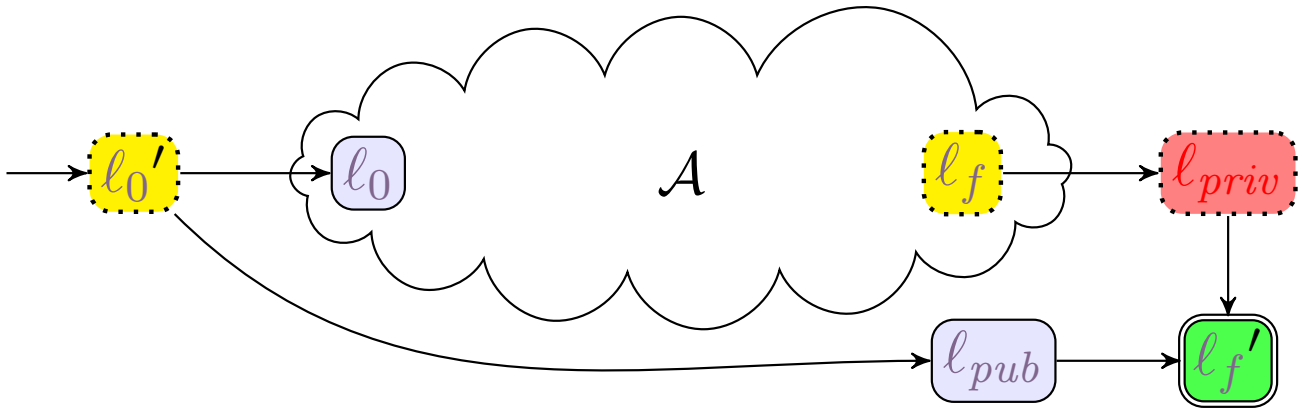
# Outline

# Timed-opacity synthesis is (very) difficult

**Theorem (Undecidability of timed-opacity-emptiness)**

*The mere existence of a parameter valuation for which there exists a duration for which timed-opacity is achieved is undecidable.*

Proof idea: reduction from reachability-emptiness for PTAs [Alur et al., 1993]




Remark: decidable subclass

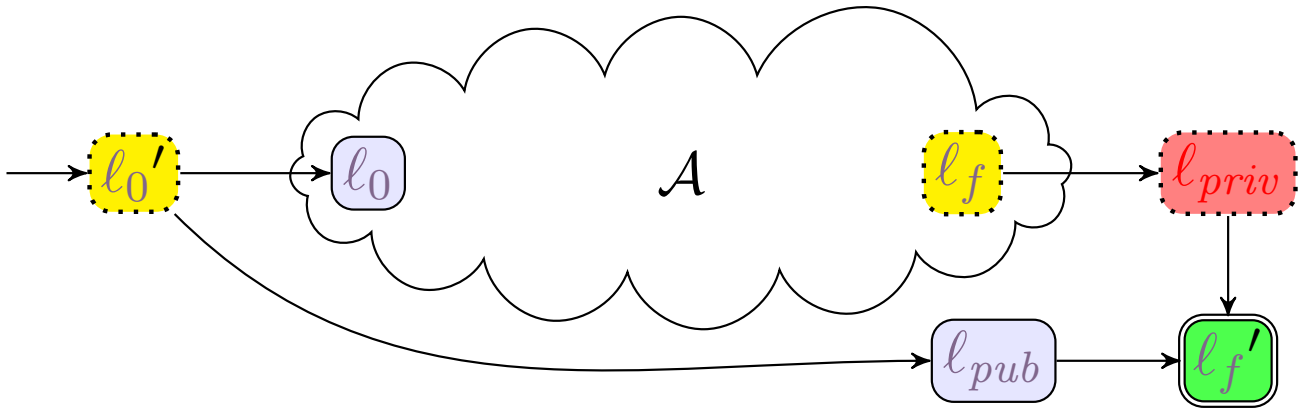
(see Theorem 1 in the paper)

# Timed-opacity synthesis is (very) difficult

**Theorem (Undecidability of timed-opacity-emptiness)**

*The mere existence of a parameter valuation for which there exists a duration for which timed-opacity is achieved is undecidable.*

Proof idea: reduction from reachability-emptiness for PTAs [Alur et al., 1993]




Remark: decidable subclass

(see Theorem 1 in the paper)

In the following, we adopt a "best-effort" approach

- Approach not guaranteed to terminate in theory

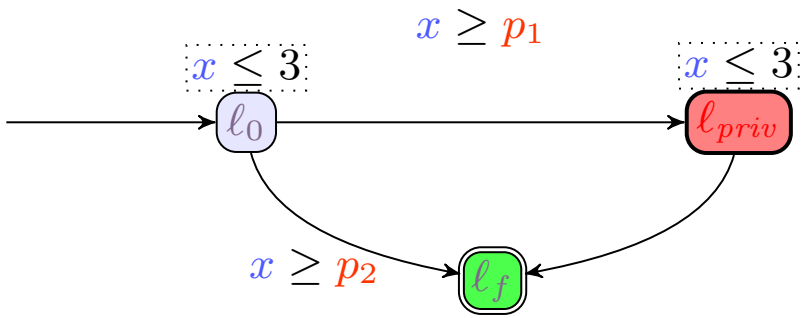
# Outline

# Reducing timed-opacity synthesis to reachability synthesis

Big picture:

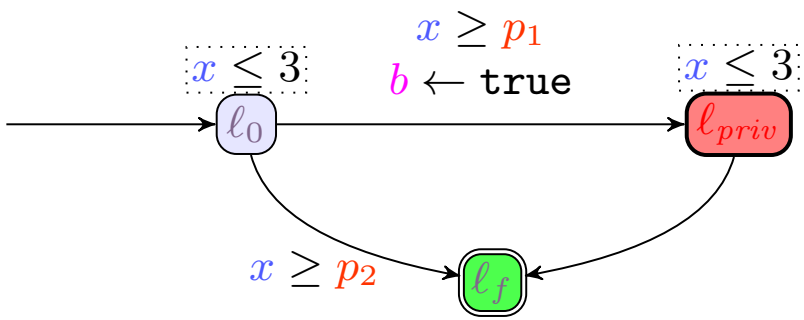
- Formalism: parametric timed automata
- Our approach:
  1. Perform a (mild) transformation of the PTA
  2. Perform self-composition
  3. Apply parametric timed model checking (reachability-synthesis)
- Tool support: IMITATOR [André et al., 2012]

# Our transformation of the PTA in 4 overlays



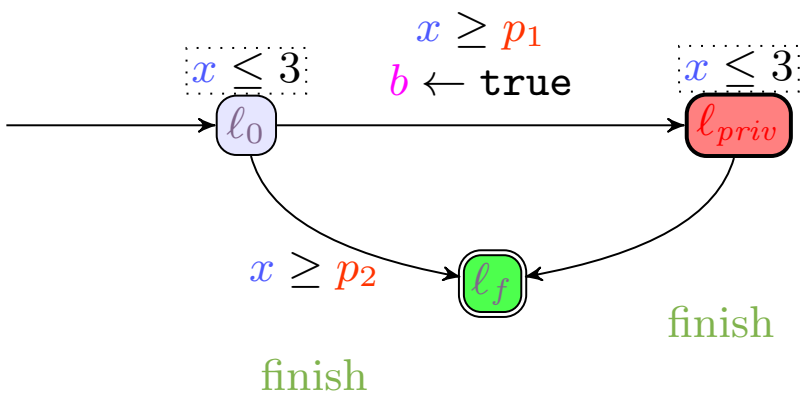
# Our transformation of the PTA in 4 overlays

1. Add a Boolean flag $b$ to remember whether $\ell_{priv}$ was visited



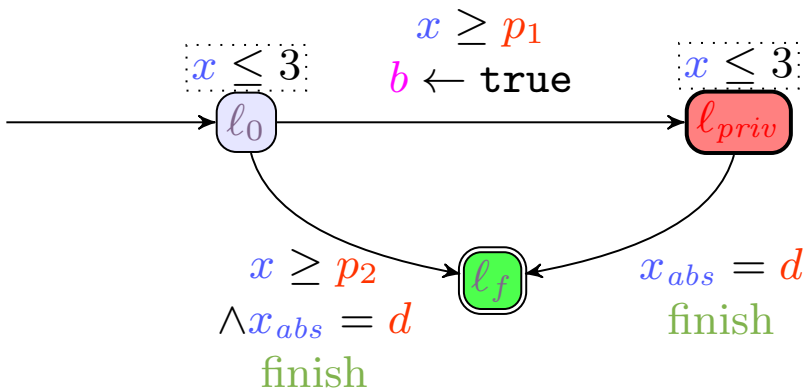
# Our transformation of the PTA in 4 overlays

1. Add a Boolean flag $b$ to remember whether $\ell_{priv}$ was visited
2. Add a synchronization action $\mathrm{finish}$ on any transition to $\ell_f$



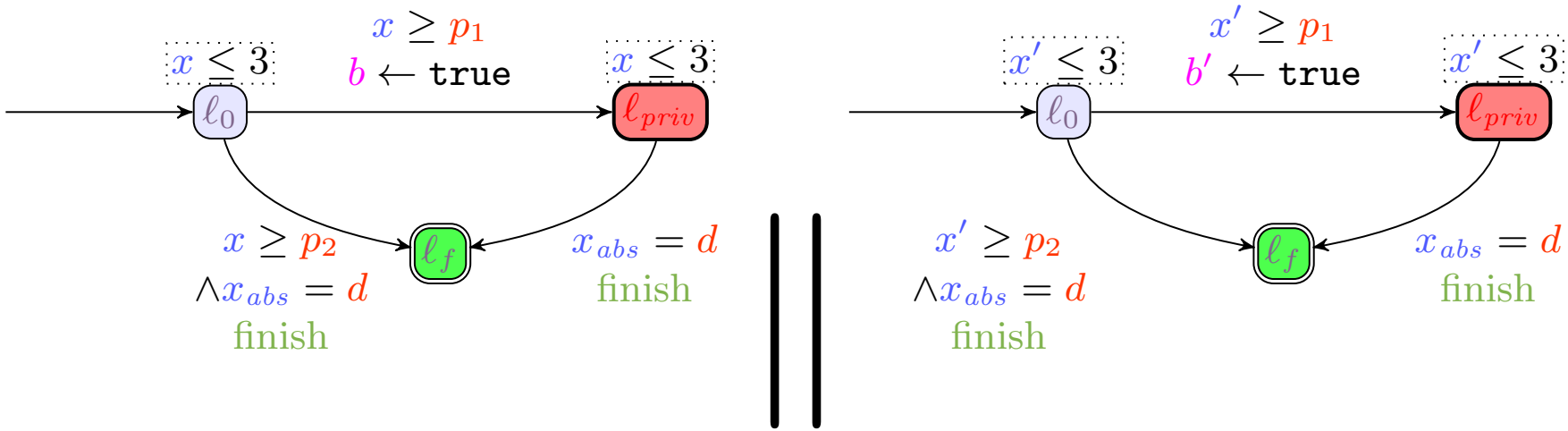
# Our transformation of the PTA in 4 overlays

1. Add a Boolean flag $b$ to remember whether $\ell_{priv}$ was visited
2. Add a synchronization action $\mathrm{finish}$ on any transition to $\ell_f$
3. Measure the (parametric) duration to $\ell_f$ thanks to a new clock $x_{abs}$ and a new parameter $d$

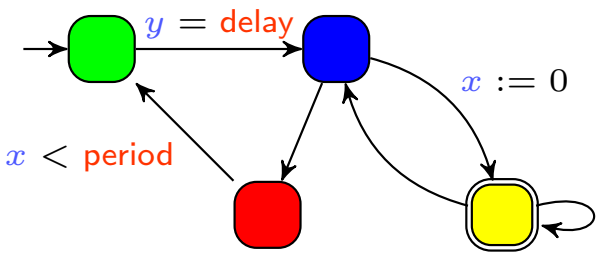


# Our transformation of the PTA in 4 overlays

1. Add a Boolean flag $b$ to remember whether $\ell_{priv}$ was visited
2. Add a synchronization action finish on any transition to $\ell_f$
3. Measure the (parametric) duration to $\ell_f$ thanks to a new clock $x_{abs}$ and a new parameter $d$
4. Perform self-composition (i. e., a synchronization on shared actions of the PTA with a copy of itself)



# timed model checking




A model of the system

$?$

$\models$

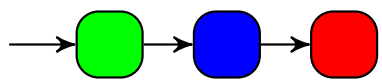
is unreachable

A property to be satisfied

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

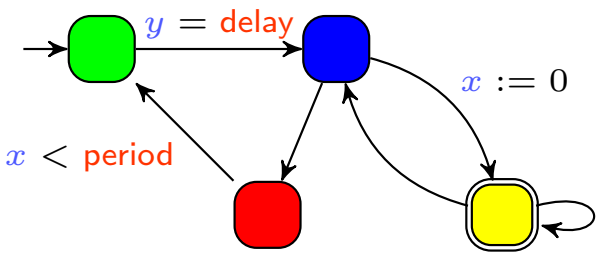
**Yes**

**No**



Counterexample

# Parametric timed model checking




A model of the system

$?$

$\models$

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...**

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

# Applying reachability-synthesis

We then synthesize all parameter valuations (including $d$) for which the following discrete state is reachable:

- the original automaton is in $\ell_f$ with $b = \texttt{true}$
- the copy automaton is in $\ell_f$ with $b' = \texttt{false}$

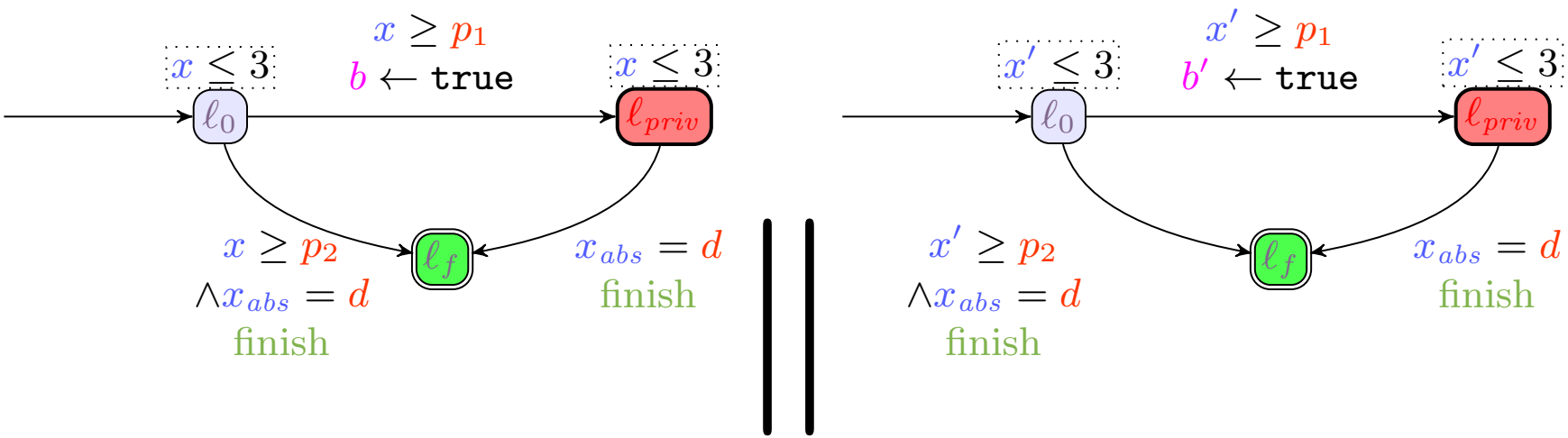
# Applying reachability-synthesis

We then synthesize all parameter valuations (including $d$) for which the following discrete state is reachable:

- the original automaton is in $\ell_f$ with $b = \texttt{true}$
- the copy automaton is in $\ell_f$ with $b' = \texttt{false}$

Intuition:

- for the same duration (thanks to the synchroniation on finish), we can reach $\ell_f$ "both" with passing by $\ell_{priv}$ (i. e., $b = \texttt{true}$) or without (i. e., $b = \texttt{false}$)




Formal proof of correctness: see paper

# Outline

# Experimental environment

## Algorithms

1. Timed-opacity: “for a non-parametric TA, is the TA opaque for all execution times?”
2. Timed-opacity synthesis: “for a PTA, synthesize parameter valuations and execution times ensuring timed opacity”

## Benchmarks

- Common PTA benchmarks [André, 2019]
- Library of Java programs https://github.com/Apogee-Research/STAC/
  - Manually translated to PTAs
  - User-input variables translated to (non-timing) parameters (supported by IMITATOR)

See experiments at `doi.org/10.5281/zenodo.3251141`

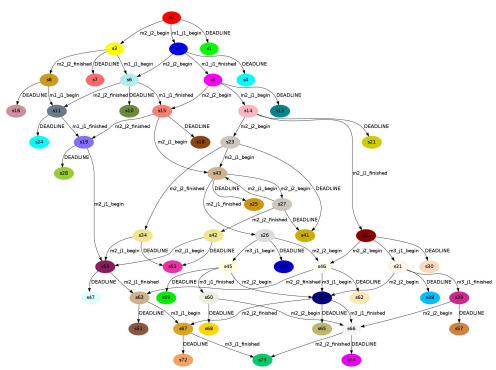
and `imitator.fr/static/ATVA19/`

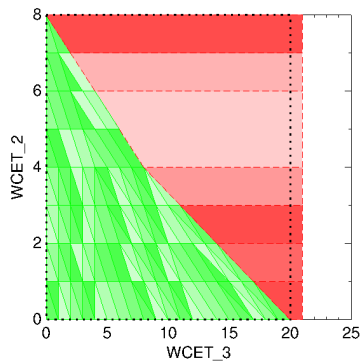
# Outline

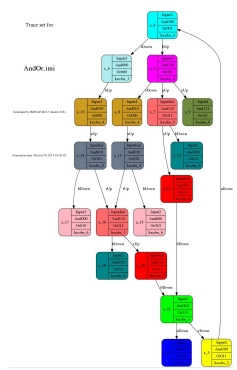
# IMITATOR

- 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





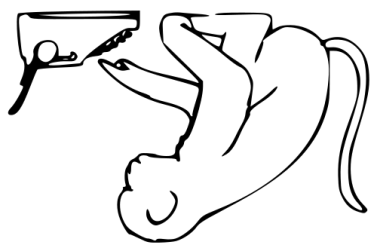
# IMITATOR

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



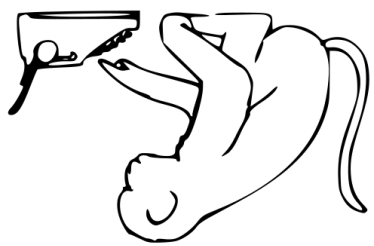
# IMITATOR

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

# Outline

# Experiments: (non-parametric) timed opacity

| Model | | | Transf. PTA | | | Result | |
|---|---|---|---|---|---|---|---|
| Name | $\lvert\mathcal{A}\rvert$ | $\lvert X\rvert$ | $\lvert\mathcal{A}\rvert$ | $\lvert X\rvert$ | $\lvert P\rvert$ | Time (s) | Vulnerable? |
| Fig. 5, [Vasilikos et al., 2018] | 1 | 1 | 2 | 3 | 3 | 0.02 | $(\sqrt{})$ |
| Fig. 1b, [Gardey et al., 2007] | 1 | 1 | 2 | 3 | 1 | 0.04 | $(\sqrt{})$ |
| Fig. 2a, [Gardey et al., 2007] | 1 | 1 | 2 | 3 | 1 | 0.05 | $(\sqrt{})$ |
| Fig. 2b, [Gardey et al., 2007] | 1 | 1 | 2 | 3 | 1 | 0.02 | $(\sqrt{})$ |
| Web privacy problem [Benattar et al., 2015] | 1 | 2 | 2 | 4 | 1 | 0.07 | $(\sqrt{})$ |
| Coffee | 1 | 2 | 2 | 5 | 1 | 0.05 | $\times$ |
| Fischer-HSRV02 | 3 | 2 | 6 | 5 | 1 | 5.83 | $(\sqrt{})$ |
| STAC:1:n | | | 2 | 3 | 6 | 0.12 | $(\sqrt{})$ |
| STAC:1:v | | | 2 | 3 | 6 | 0.11 | $\sqrt{}$ |
| STAC:3:n | | | 2 | 3 | 8 | 0.72 | $\times$ |
| STAC:3:v | | | 2 | 3 | 8 | 0.74 | $(\sqrt{})$ |
| STAC:4:n | | | 2 | 3 | 8 | 6.40 | $\sqrt{}$ |
| STAC:4:v | | | 2 | 3 | 8 | 265.52 | $\sqrt{}$ |
| STAC:5:n | | | 2 | 3 | 6 | 0.24 | $\times$ |
| STAC:11A:v | | | 2 | 3 | 8 | 47.77 | $(\sqrt{})$ |
| STAC:11B:v | | | 2 | 3 | 8 | 59.35 | $(\sqrt{})$ |
| STAC:12c:v | | | 2 | 3 | 8 | 18.44 | $\sqrt{}$ |
| STAC:12e:n | | | 2 | 3 | 8 | 0.58 | $\sqrt{}$ |
| STAC:12e:v | | | 2 | 3 | 8 | 1.10 | $(\sqrt{})$ |
| STAC:14:n | | | 2 | 3 | 8 | 22.34 | $(\sqrt{})$ |

$\times$ = not vulnerable; $(\sqrt{})$ = vulnerable, can be repaired; $\sqrt{}$ = vulnerable, cannot be repaired

# Outline

# Experiments: (parametric) timed-opacity synthesis

| Model | | | | Transf. PTA | | | Result | |
|---|---|---|---|---|---|---|---|---|
| Name | $\lvert\mathcal{A}\rvert$ | $\lvert X\rvert$ | $\lvert P\rvert$ | $\lvert\mathcal{A}\rvert$ | $\lvert X\rvert$ | $\lvert P\rvert$ | Time (s) | Constraint |
| Fig. 5, [Vasilikos et al., 2018] | 1 | 1 | 0 | 2 | 3 | 4 | 0.02 | $K$ |
| Fig. 1b, [Gardey et al., 2007] | 1 | 1 | 0 | 2 | 3 | 3 | 0.03 | $K$ |
| Fig. 2, [Gardey et al., 2007] | 1 | 1 | 0 | 2 | 3 | 3 | 0.05 | $K$ |
| Web privacy problem [Benattar et al., 2015] | 1 | 2 | 2 | 2 | 4 | 3 | 0.07 | $K$ |
| Coffee | 1 | 2 | 3 | 2 | 5 | 4 | 0.10 | $\top$ |
| Fischer-HSRV02 | 3 | 2 | 2 | 6 | 5 | 3 | 7.53 | $K$ |
| STAC:3:v | | | 2 | 2 | 3 | 9 | 0.93 | $K$ |

$K$ = some valuations make the system non-vulnerable; $\top$ = all valuations make the system non-vulnerable

# Outline

# Conclusion

Context: vulnerability by timing-attacks

- Attacker model: observability of the global computation time
- Goal: avoid leaking information on whether some discrete state has been visited

Solution: parametric timed model checking

- Formalism: parametric timed automata
- Our approach:
  1. Perform a (mild) transformation of the PTA
  2. Perform self-composition
  3. Apply parametric timed model checking (reachability-synthesis)
- Toolkit: IMITATOR
- Benchmarks: concurrent systems and Java programs

# Perspectives

- Theoretical open problems
  - Decidability of timed-opacity emptiness remains open for 1 clock
  - Case of U-PTAs or L-PTAs [Bozzelli and La Torre, 2009]
- Automated translation of Java programs
  - Our translation required non-trivial creativity
  - How to automate it?
  - Finer grain needed for “untimed” instructions: probabilistic timings?
- Repairing a non-opaque system
  - “From PTA parameter tuning back to the original system”
  - In programs: using `Wait` or `Sleep`
  - Preliminary ideas in the paper

# We hire!

- What: Project on quantitative formal methods + security (2020-2023)
- Where: France (Nancy / Nantes), Singapore
- Who: Master students, PhD, post-docs
  …starting anytime!

# 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 Kosaraju, S. R., Johnson, D. S., and Aggarwal, A., editors, *STOC*, pages 592–601, New York, NY, USA. ACM.

André, É. (2019).
A benchmark library for parametric timed model checking.
In Artho, C. and Ölveczky, P. C., editors, *FTSCS*, volume 1008 of *Communications in Computer and Information Science*, pages 75–83. Springer.

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 Sun, J. (2019).
Parametric timed model checking for guaranteeing timed opacity.
In Chen, Y.-F., Cheng, C.-H., and Esparza, J., editors, *ATVA*, Lecture Notes in Computer Science. Springer.
To appear.

Benattar, G., Cassez, F., Lime, D., and Roux, O. H. (2015).
Control and synthesis of non-interferent timed systems.
*International Journal of Control*, 88(2):217–236.

# References II


Bouyer, P., Markey, N., and Sankur, O. (2013).
Robustness in timed automata.
In Abdulla, P. A. and Potapov, I., editors, *RP*, volume 8169 of *Lecture Notes in Computer Science*, pages 1–18. Springer.
Invited paper.

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.

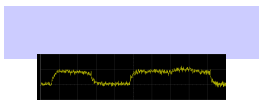
Cassez, F. (2009).
The dark side of timed opacity.
In Park, J. H., Chen, H., Atiquzzaman, M., Lee, C., Kim, T., and Yeo, S., editors, *ISA*, volume 5576 of *Lecture Notes in Computer Science*, pages 21–30. Springer.

Gardey, G., Mullins, J., and Roux, O. H. (2007).
Non-interference control synthesis for security timed automata.
*Electronic Notes in Theoretical Computer Science*, 180(1):35–53.

Vasilikos, P., Nielson, F., and Nielson, H. R. (2018).
Secure information release in timed automata.
In Bauer, L. and Küsters, R., editors, *POST*, volume 10804 of *Lecture Notes in Computer Science*, pages 28–52. Springer.

# Licensing

# Source of the graphics used I



Title: Power attack
Author: Audriusa
Source: `https://commons.wikimedia.org/wiki/File:Power_attack.png`
License: GNU GPL

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

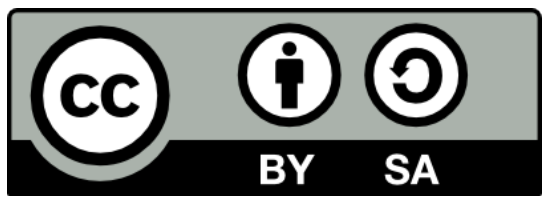
Title: Smiley green alien big eyes (cry)
Author: LadyofHats
Source: `https://commons.wikimedia.org/wiki/File:Smiley_green_alien_big_eyes.svg`
License: public domain

# License of this document

This presentation can be published, reused and modified under the terms of the license Creative Commons **Attribution-ShareAlike 4.0 Unported (CC BY-SA 4.0)**

(LaTeX source available on demand)

Author: **Étienne André**




https://creativecommons.org/licenses/by-sa/4.0/