Seminars on Formal Methods and Languages

Inria Grenoble – Rhône-Alpes

Mardi 6 septembre 2016

Grenoble, France

Computing best and worst case execution time of real-time systems under uncertainty

Étienne André

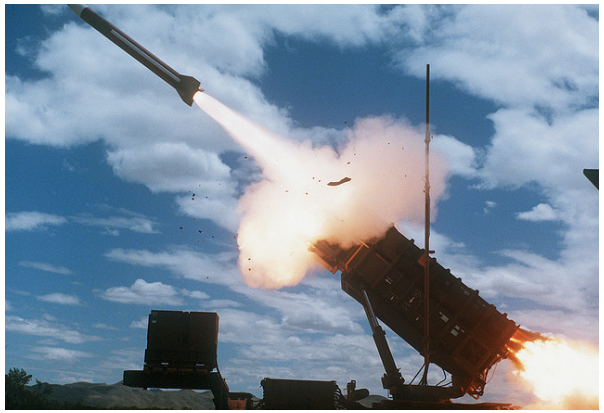
LIPN, Université Paris 13, Sorbonne Paris Cité, CNRS, France

Joint work with Giuseppe Lipari, Sun Youcheng, Camille Coti, Nguyễn Hoàng Gia

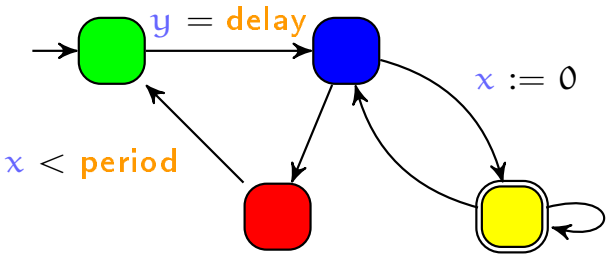
Context: Critical systems

- Need for early bug detection
  - Bugs discovered when final testing: expensive

  ⇝ Need for a thorough modeling and verification phase



Timed model checking (1/2)

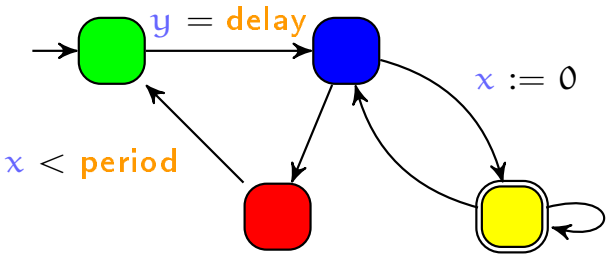


A model of the system

is unreachable

A property to be satisfied

Timed model checking (1/2)



$\overset{?}{\models}$

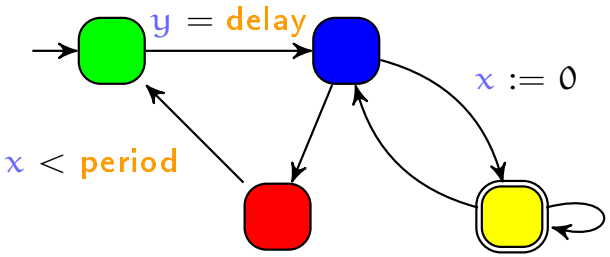
is unreachable

A model of the system

A property to be satisfied

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

Timed model checking (1/2)



$\overset{?}{\models}$

is unreachable

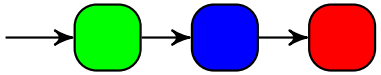
A model of the system

A property to be satisfied

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

Yes

No



Counterexample

Timed model checking (2/2)

- 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"
- Powerful model checking tools, e.g.:
  - UPPAAL [Larsen et al., 1997]
  - PAT [Sun et al., 2009]

Beyond timed model checking: parameter synthesis

- 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?
  - System incompletely specified: Can I verify my system even if I don't know the period value with full certainty?

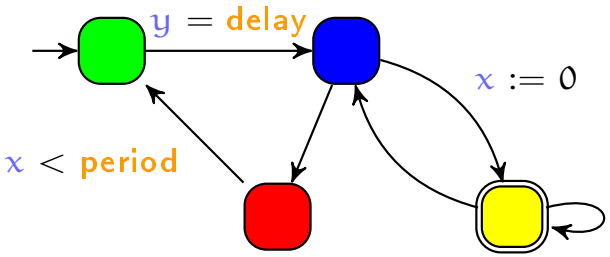
Beyond timed model checking: parameter synthesis

- 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?
  - System incompletely specified: Can I verify my system even if I don't know the period value with full certainty?

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

timed model checking



$\overset{?}{\models}$

is unreachable

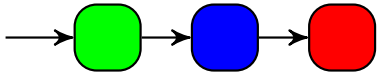
A model of the system

A property to be satisfied

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

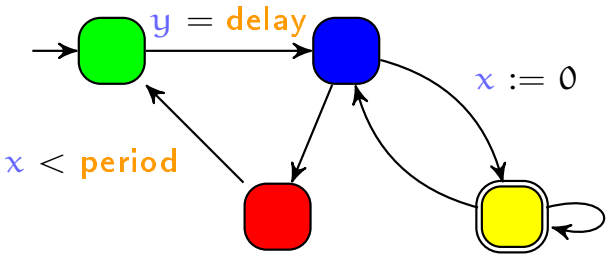
Yes

No



Counterexample

Parametric timed model checking



$\overset{?}{\models}$

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

1 Parametric Timed Automata

2 Modeling and Verifying Real-Time Systems

3 Verifying a Real-time System under Uncertainty

4 Distributed Verification of Distributed Systems

5 Conclusion and Perspectives

Outline

1 Parametric Timed Automata

2 Modeling and Verifying Real-Time Systems

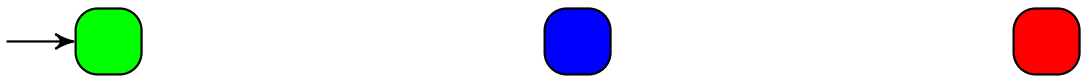
3 Verifying a Real-time System under Uncertainty

4 Distributed Verification of Distributed Systems

5 Conclusion and Perspectives

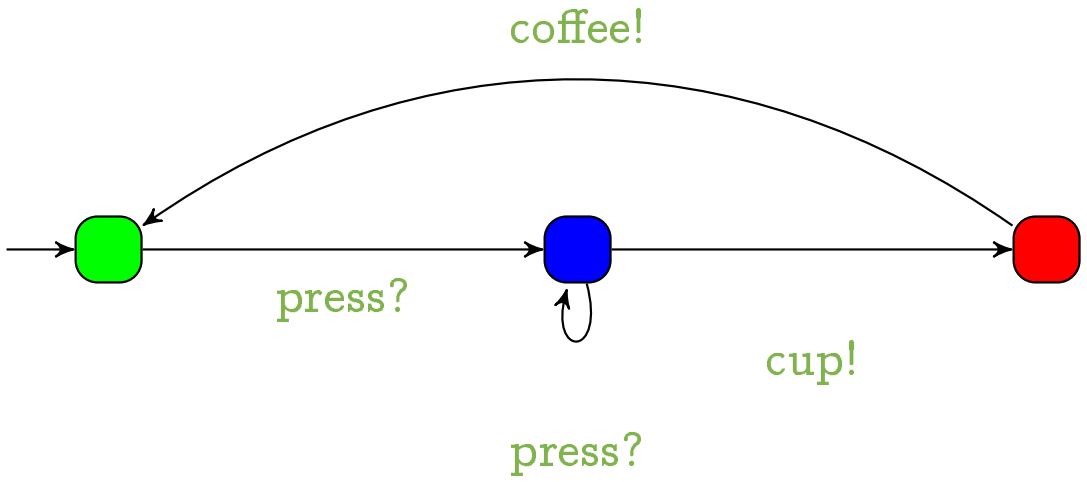
Timed automaton (TA)

- Finite state automaton (sets of locations)



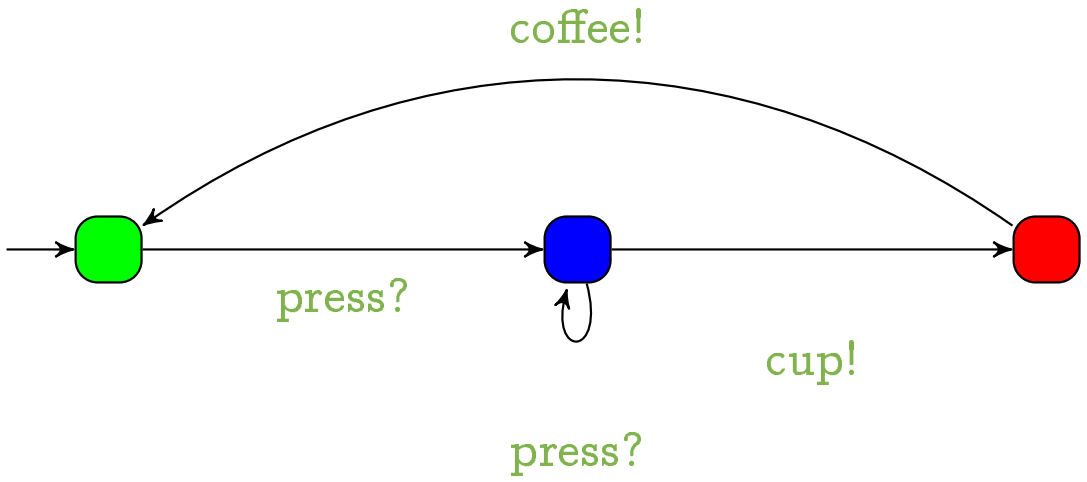
Timed automaton (TA)

- Finite state automaton (sets of locations and actions)



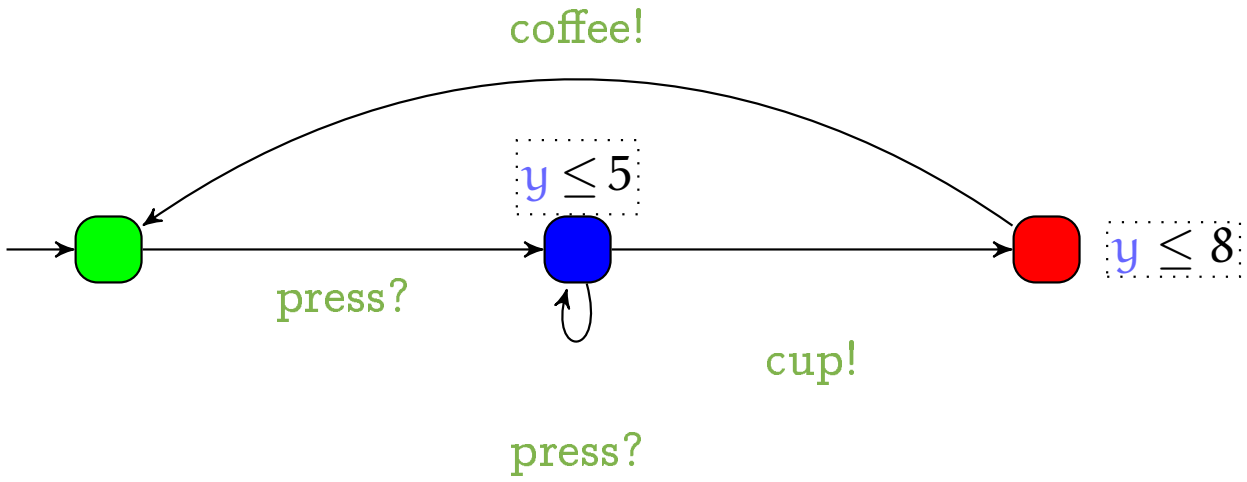
Timed automaton (TA)

- 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



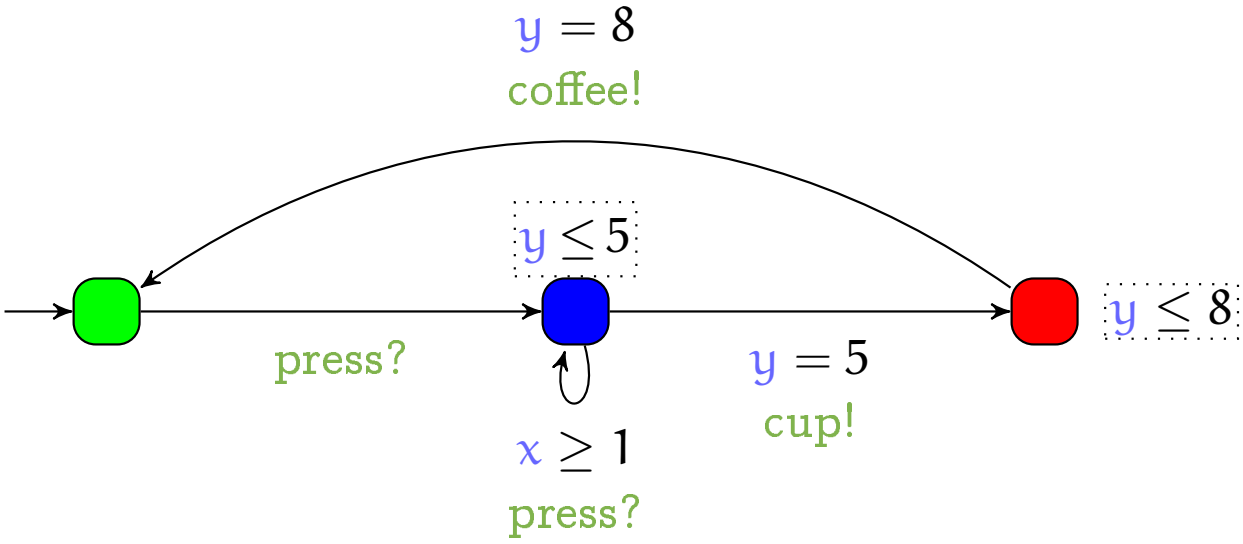
Timed automaton (TA)

- 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



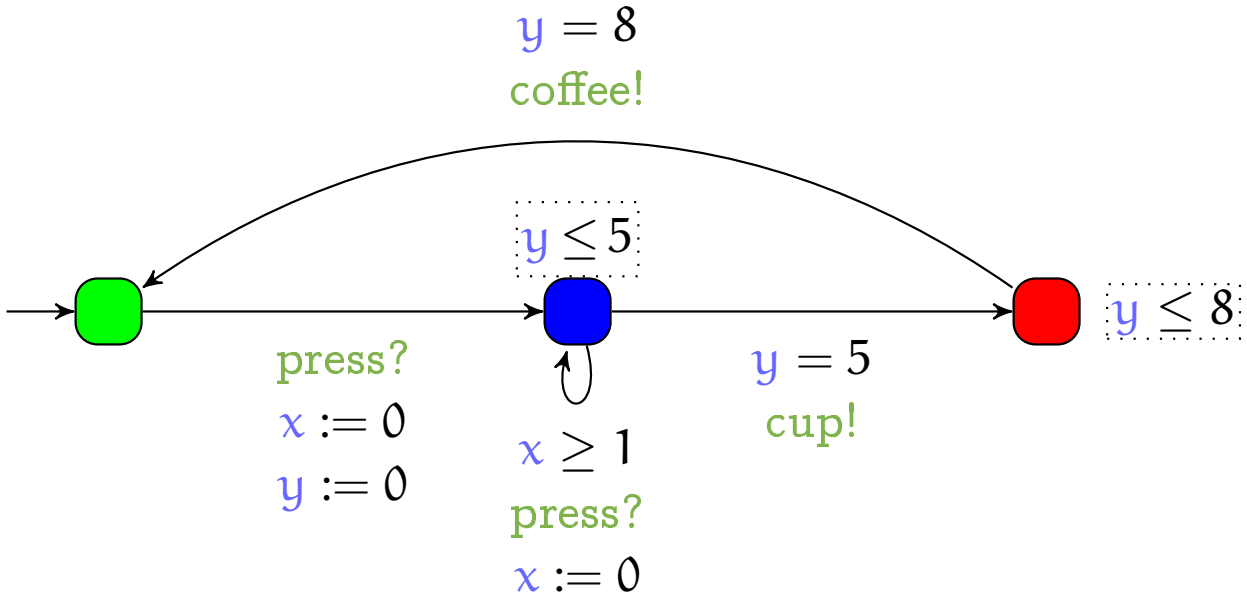
Timed automaton (TA)

- 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)

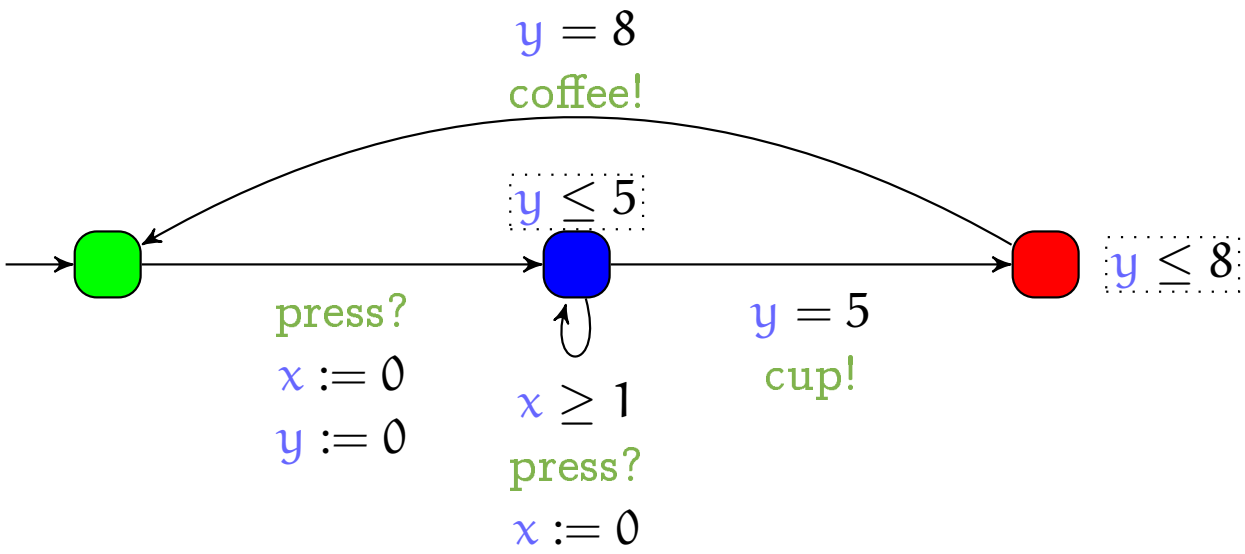
- Finite state automaton (sets of locations and actions) augmented with a set X of clocks [Alur and Dill, 1994, Henzinger et al., 1994]
  - Real-valued variables evolving linearly at the same rate
- Features
  - Location invariant: constraint to be verified to stay at a location
  - Transition guard: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

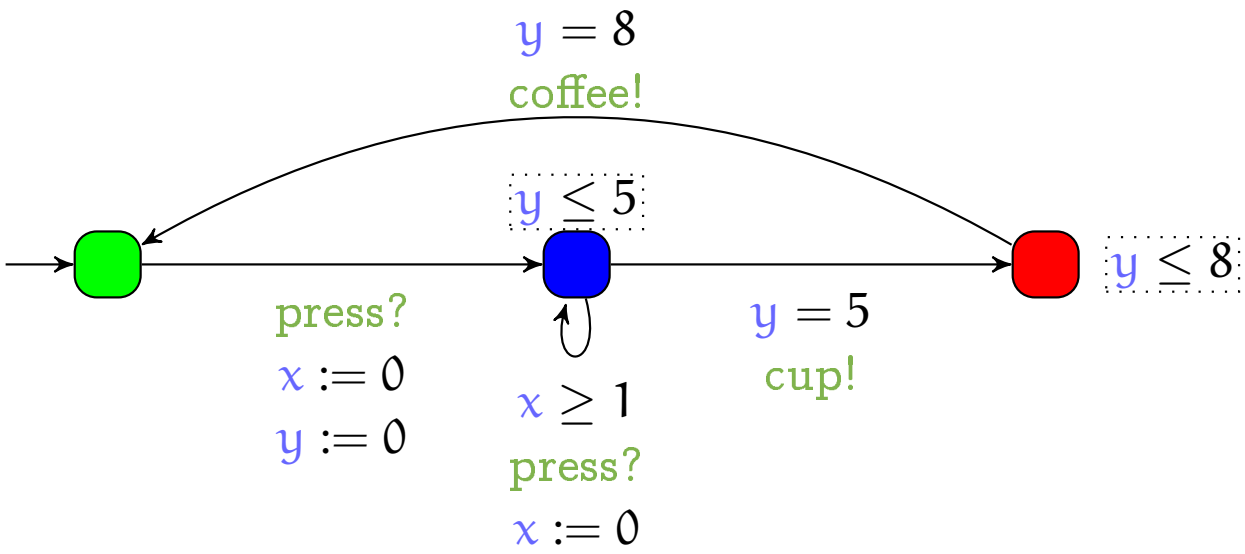
- Concrete state of a TA: pair (l, w), where
  - l is a location,
  - w is a valuation of each clock
- Concrete run: alternating sequence of concrete states and actions or time elapse

Examples of concrete runs



- Possible concrete runs for the coffee machine

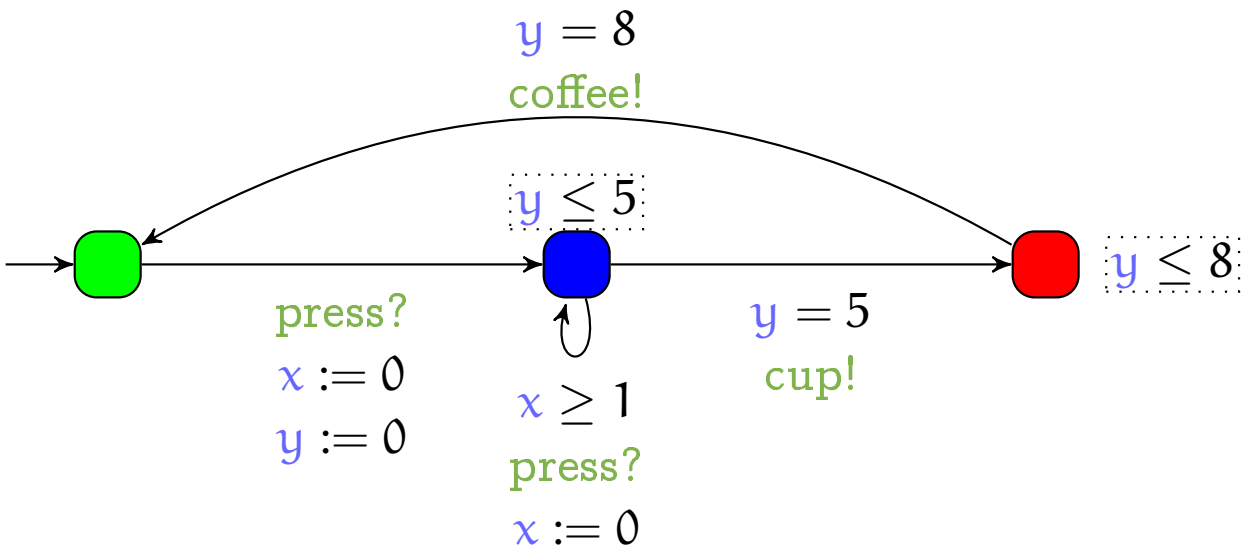
Examples of concrete runs



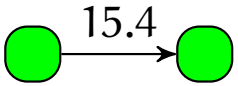
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

x	0
y	0

Examples of concrete runs

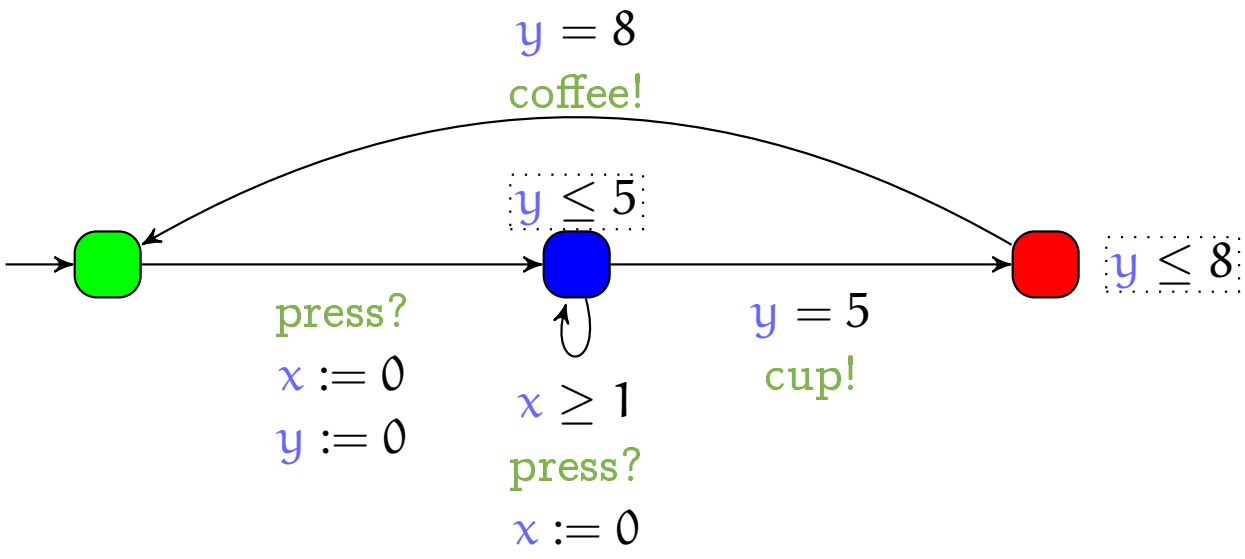


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

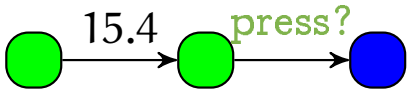


x	0	15.4
y	0	15.4

Examples of concrete runs

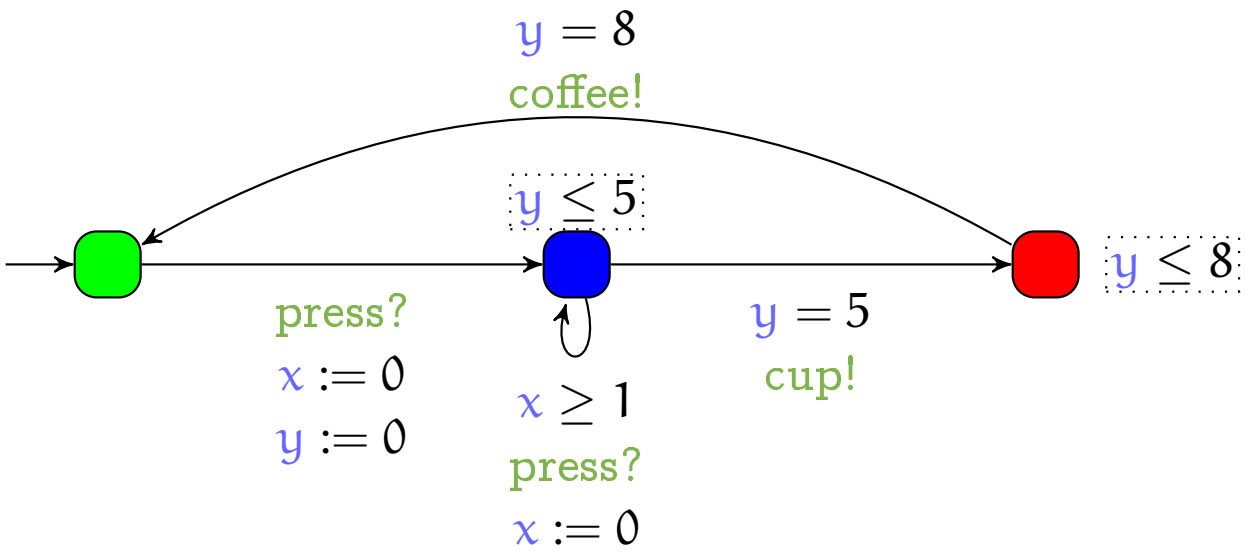


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



x	0	15.4	0
y	0	15.4	0

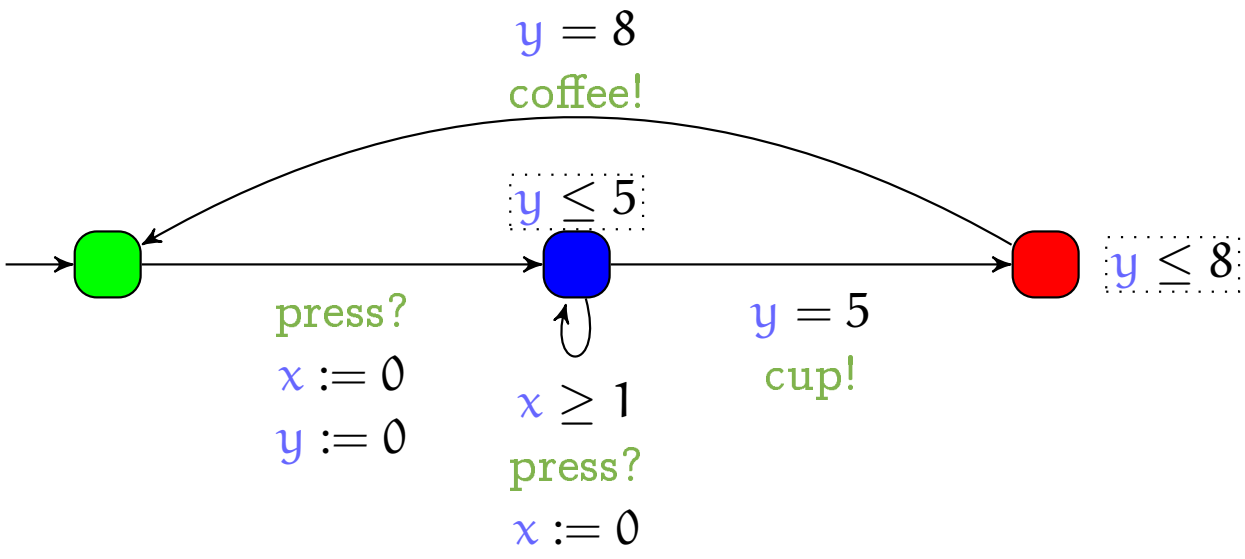
Examples of concrete runs



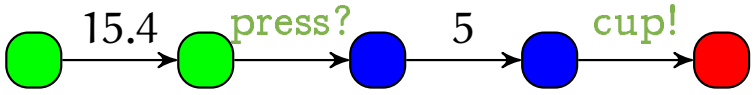
- Possible concrete runs for the coffee machine
  - Coffee with no sugar

x	0	15.4	0	5
y	0	15.4	0	5

Examples of concrete runs

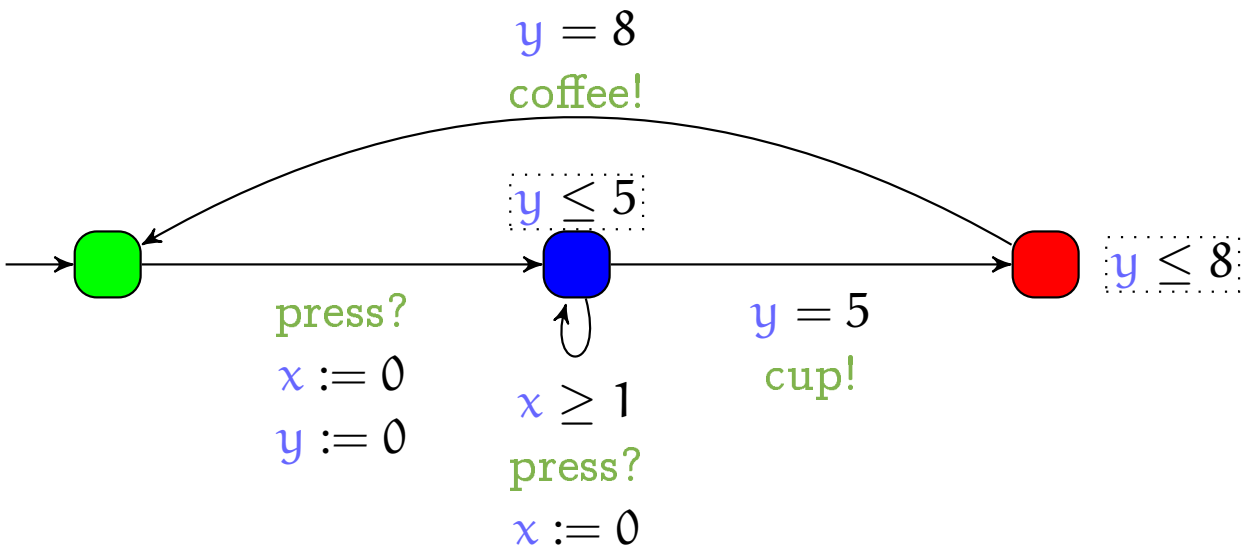


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

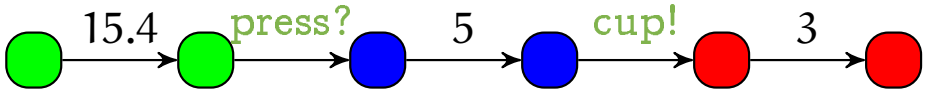


x	0	15.4	0	5	5
y	0	15.4	0	5	5

Examples of concrete runs

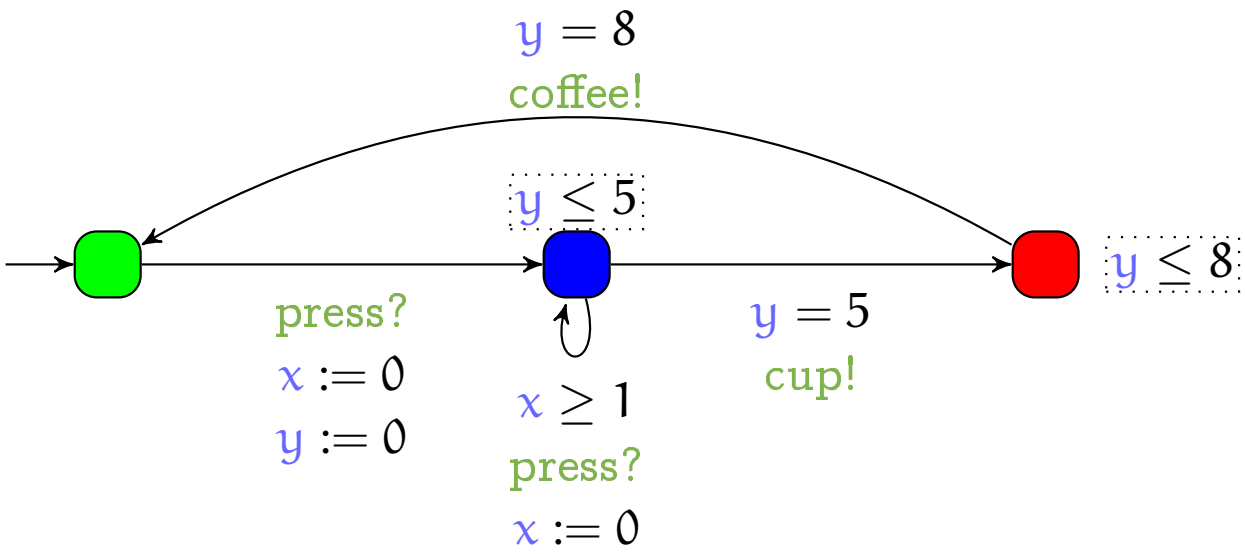


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

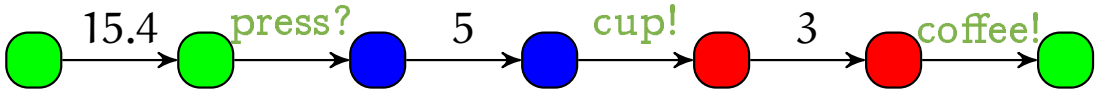


x	0	15.4	0	5	5	8
y	0	15.4	0	5	5	8

Examples of concrete runs

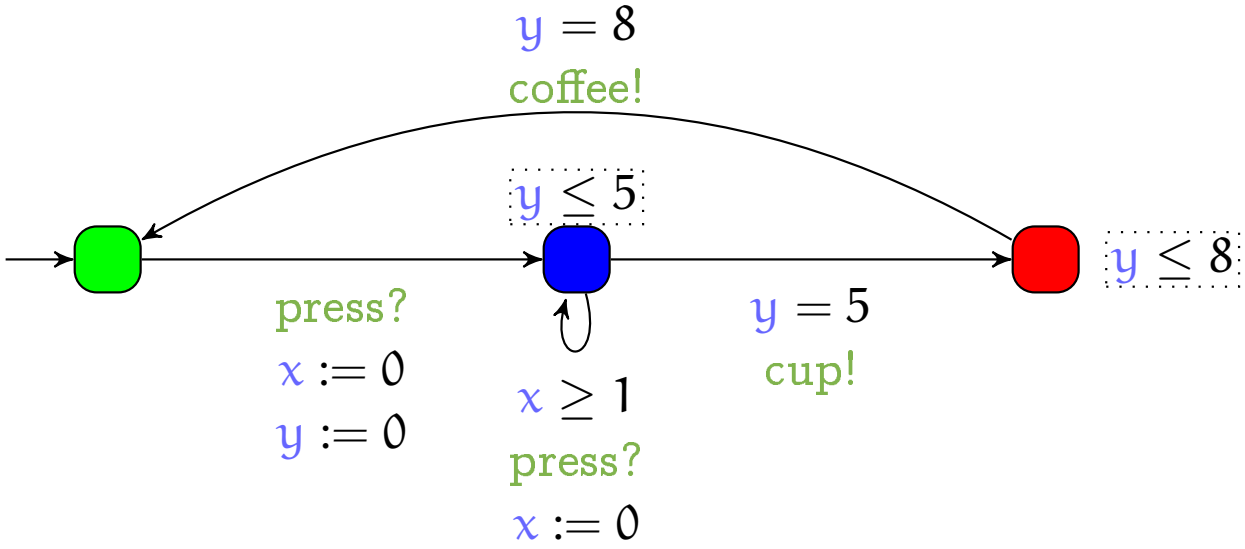


- Possible concrete runs for the coffee machine
  - Coffee with no sugar

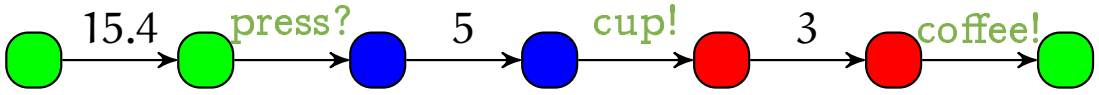


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

Examples of concrete runs



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

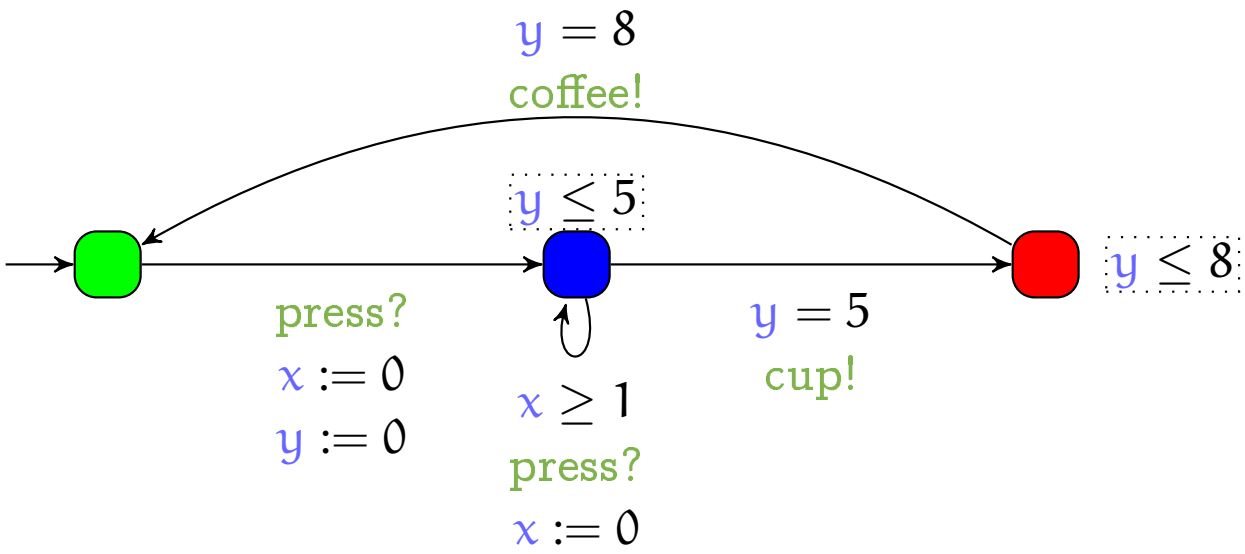


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

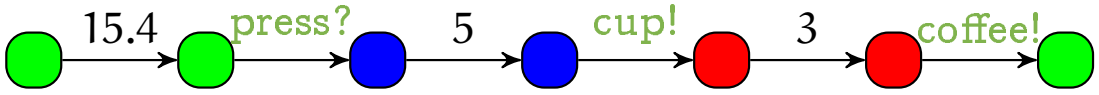
  - Coffee with 2 doses of sugar

x	0
y	0

Examples of concrete runs



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

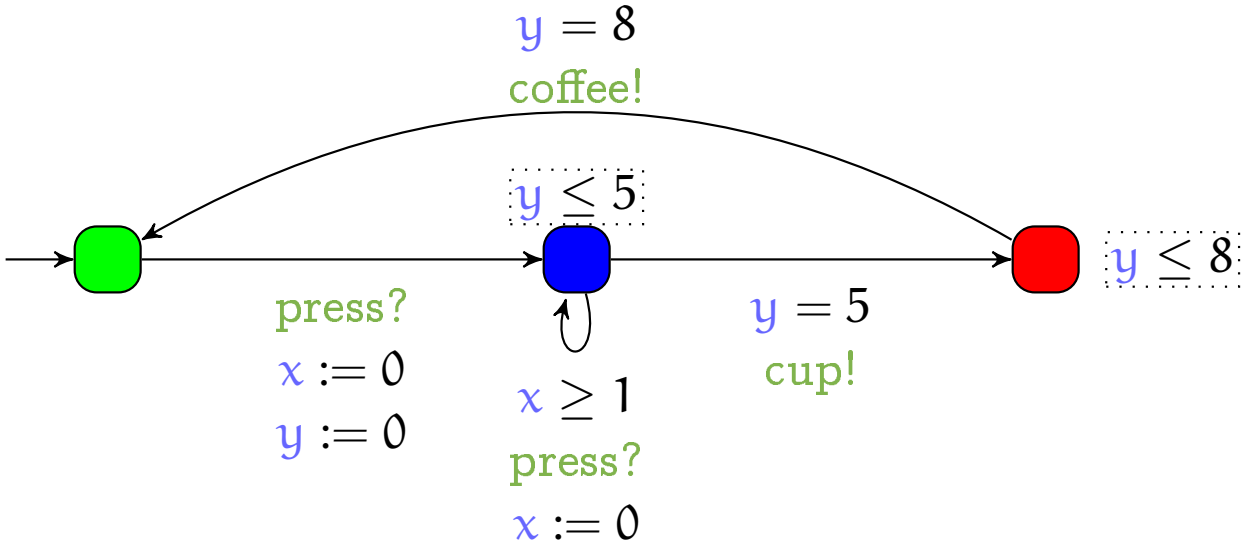


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

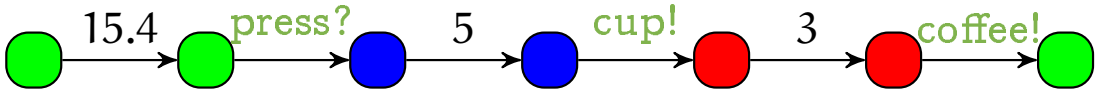
  - Coffee with 2 doses of sugar

x	0	0
y	0	0

Examples of concrete runs

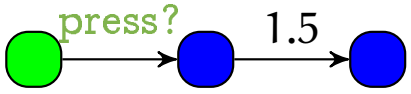


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



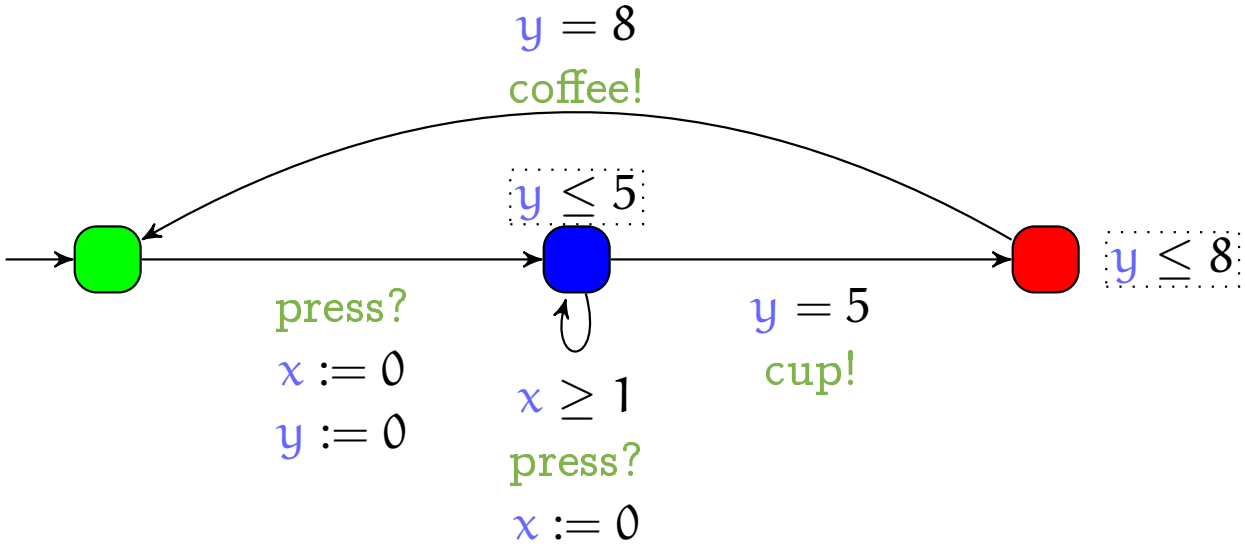
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

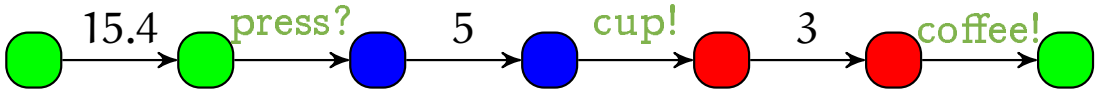


x	0	0	1.5
y	0	0	1.5

Examples of concrete runs

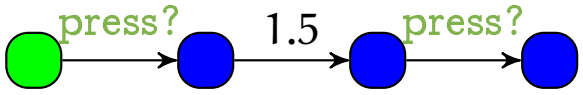


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



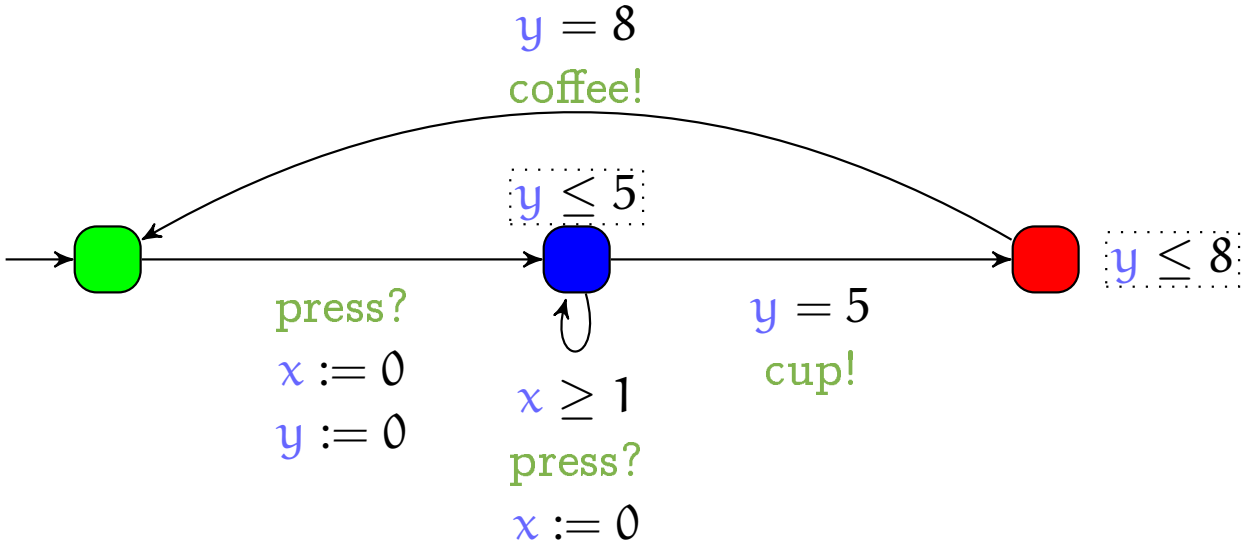
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

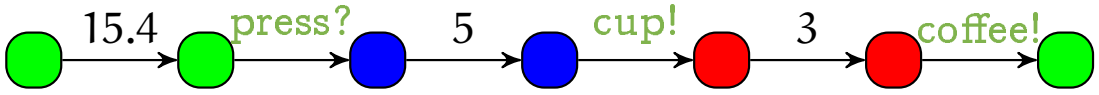


x	0	0	1.5	0
y	0	0	1.5	1.5

Examples of concrete runs

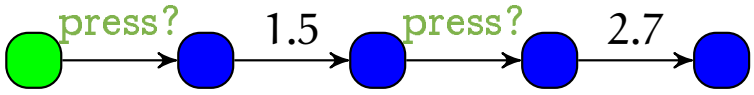


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



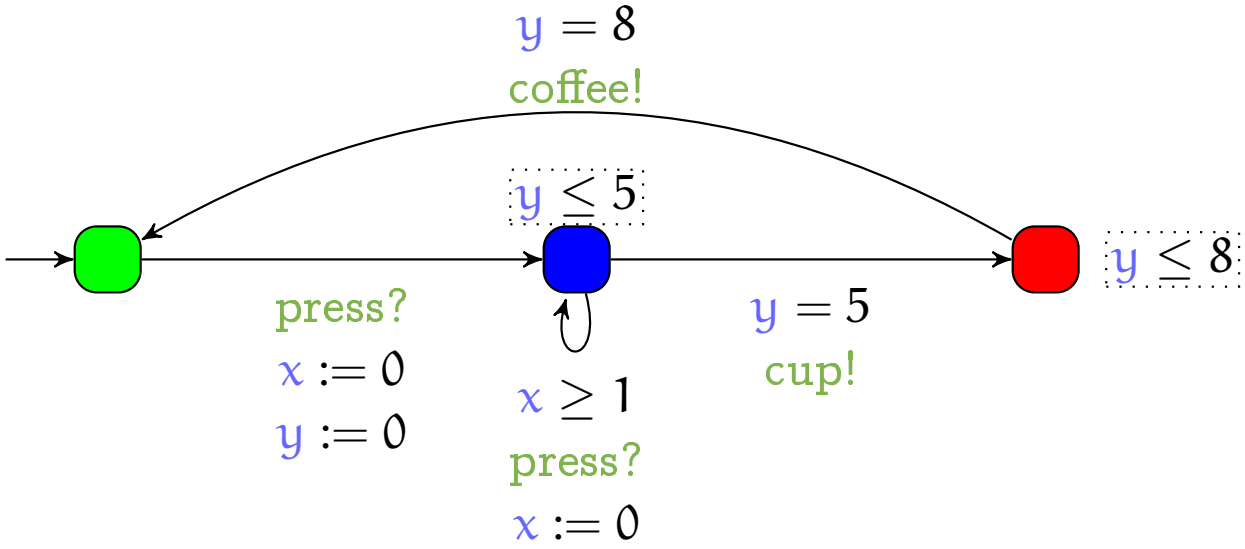
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

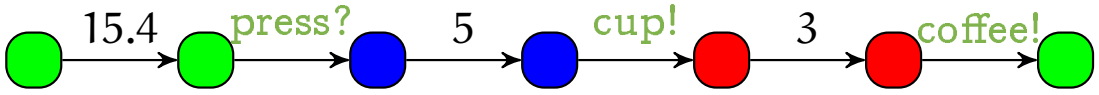


x	0	0	1.5	0	2.7
y	0	0	1.5	1.5	4.2

Examples of concrete runs



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

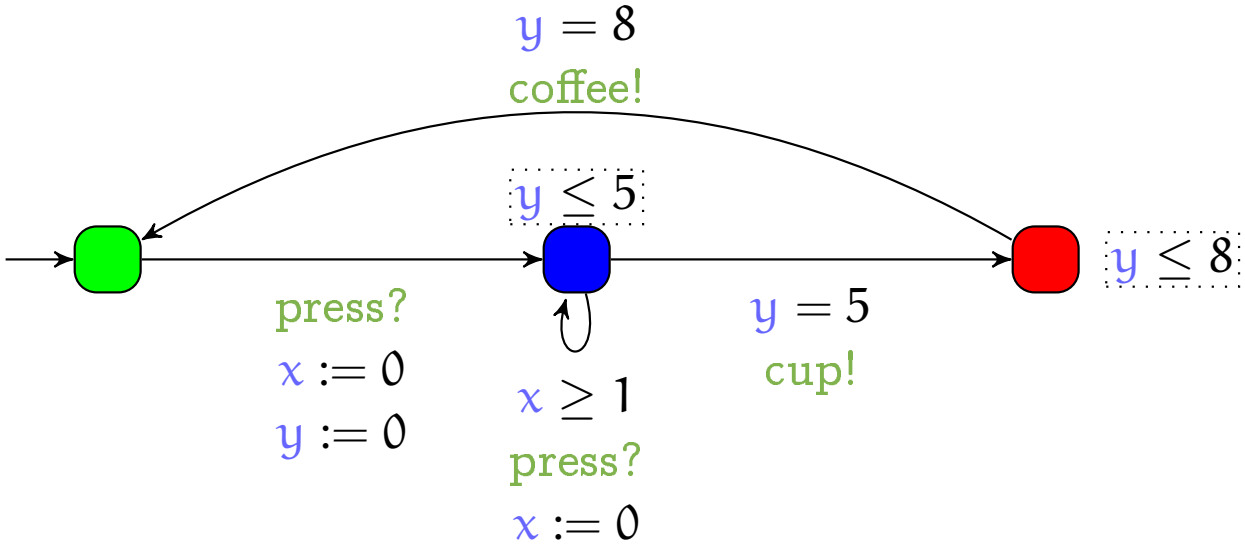


x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

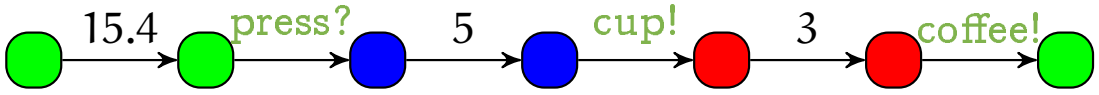
  - Coffee with 2 doses of sugar

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

Examples of concrete runs

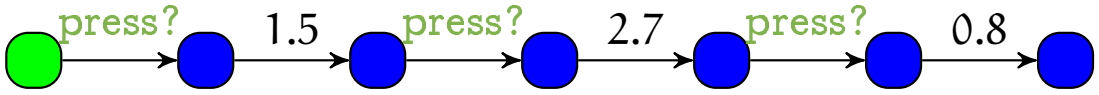


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



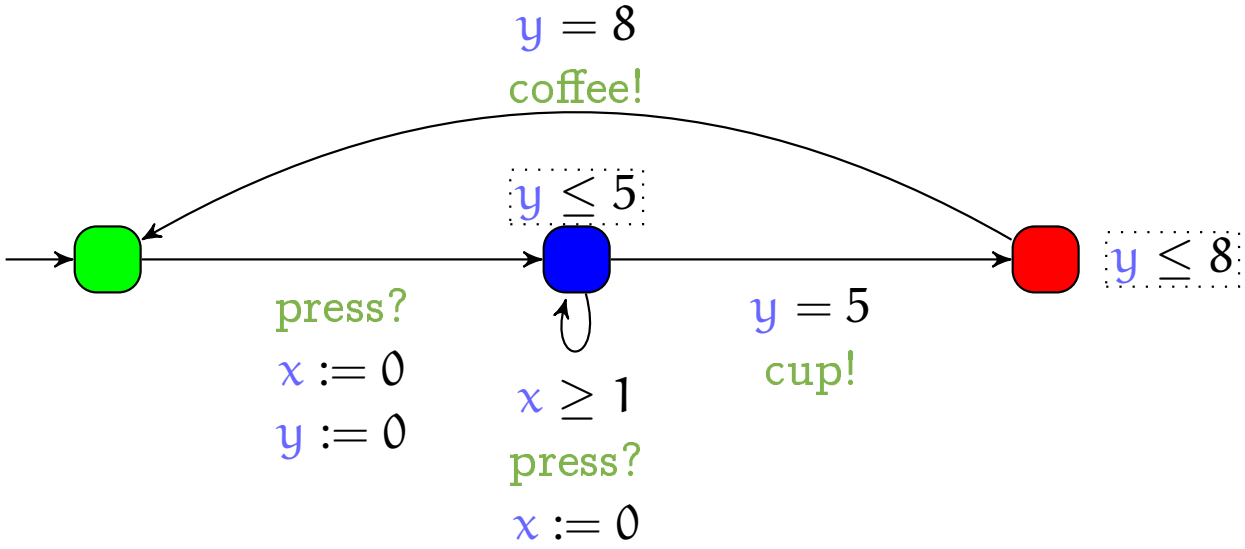
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

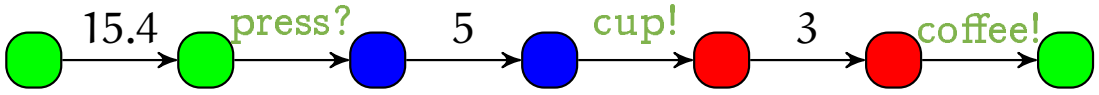


x	0	0	1.5	0	2.7	0	0.8
y	0	0	1.5	1.5	4.2	4.2	5

Examples of concrete runs

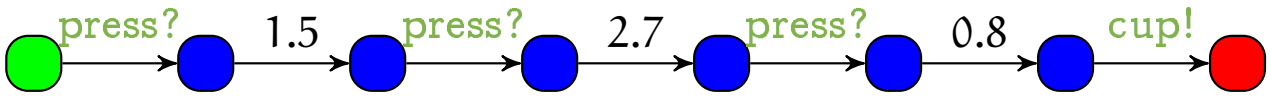


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



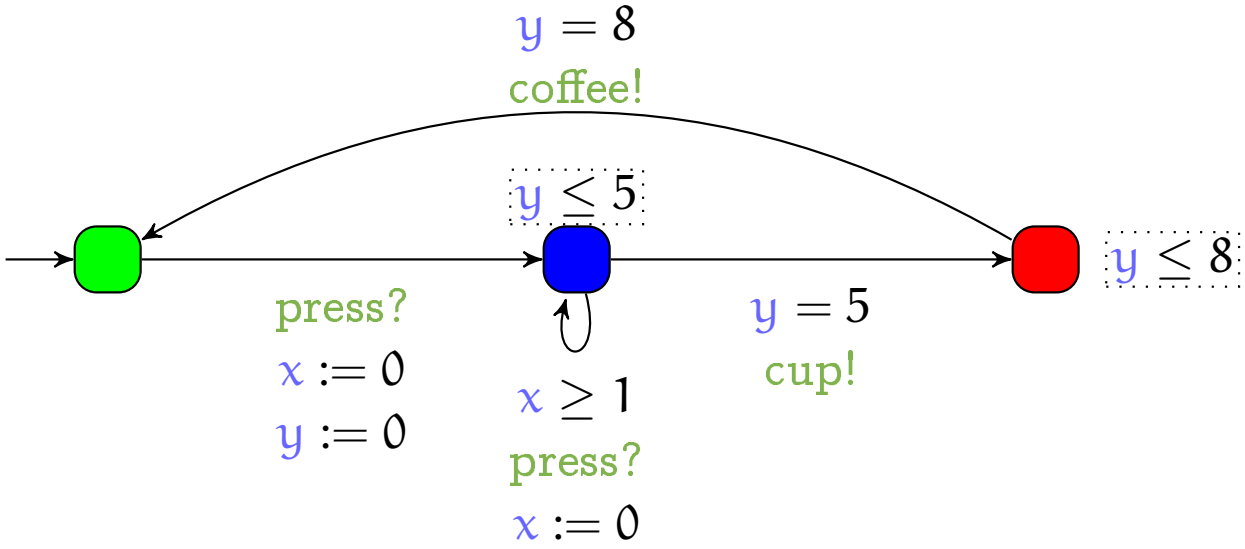
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

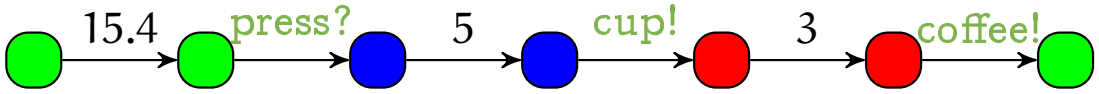


x	0	0	1.5	0	2.7	0	0.8	0.8
y	0	0	1.5	1.5	4.2	4.2	5	5

Examples of concrete runs

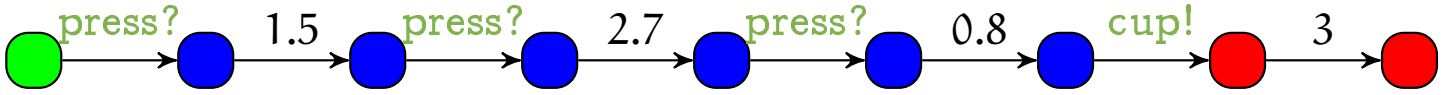


- Possible concrete runs for the coffee machine
  - Coffee with no sugar



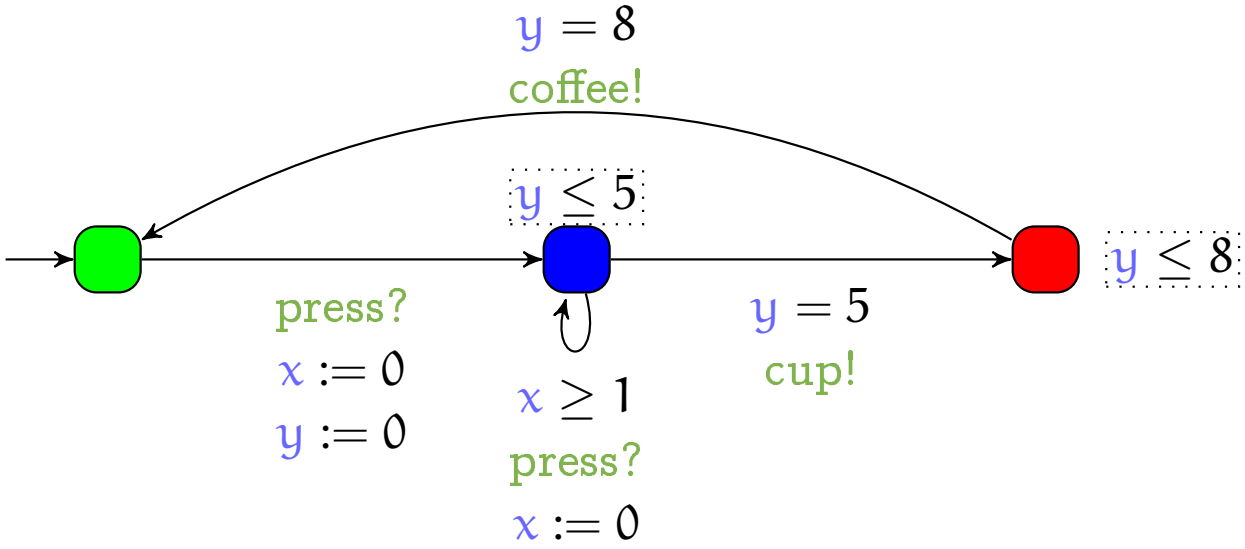
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - Coffee with 2 doses of sugar

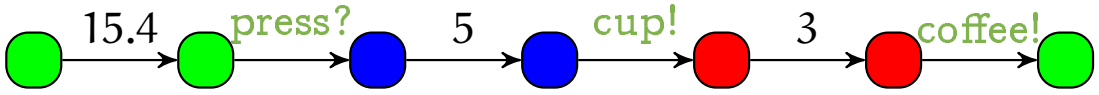


x	0	0	1.5	0	2.7	0	0.8	0.8	3.8
y	0	0	1.5	1.5	4.2	4.2	5	5	8

Examples of concrete runs



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



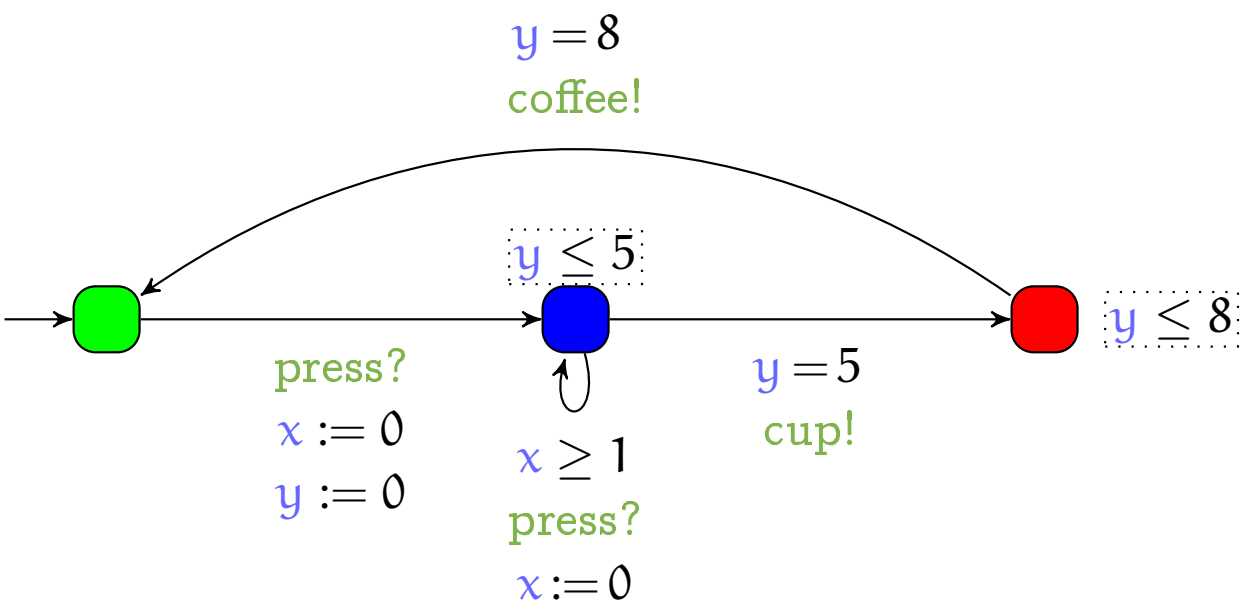
x	0	15.4	0	5	5	8	8
y	0	15.4	0	5	5	8	8

  - 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

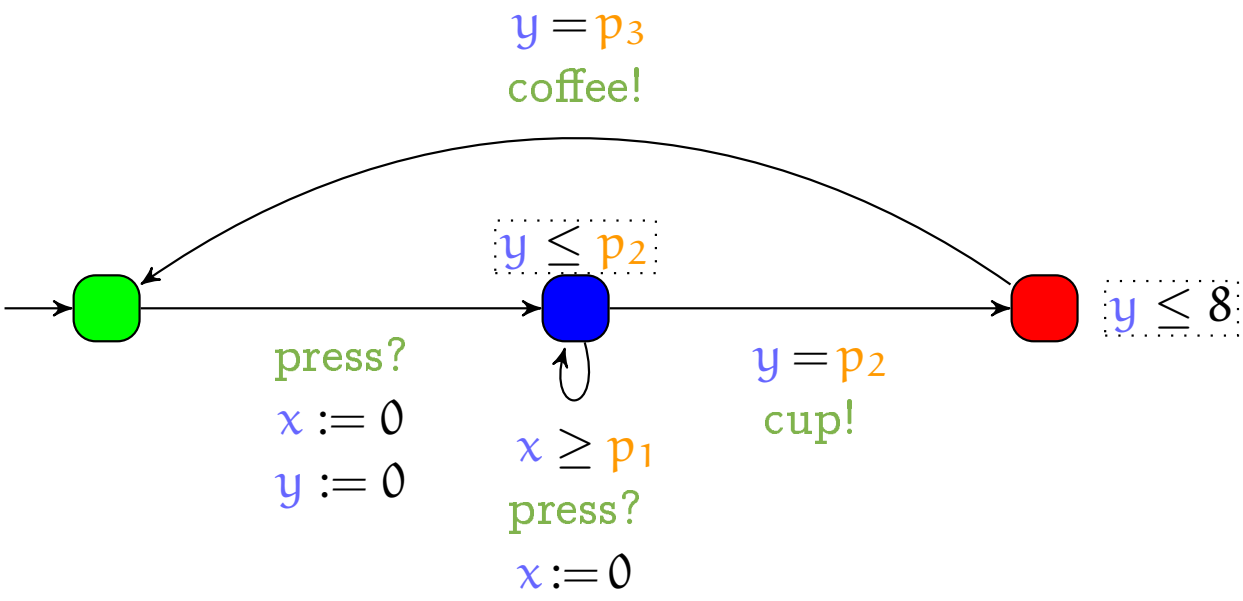
Parametric timed automaton (PTA)

- 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



Outline

1 Parametric Timed Automata

2 Modeling and Verifying Real-Time Systems

3 Verifying a Real-time System under Uncertainty

4 Distributed Verification of Distributed Systems

5 Conclusion and Perspectives

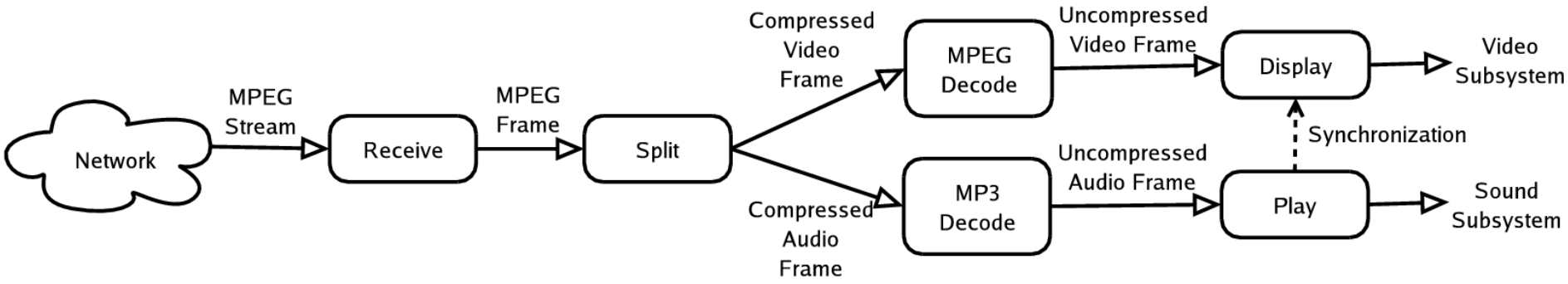
Context: Hard real-time embedded systems

- Modern hard real-time embedded systems are distributed in nature
- Many of them have critical timing requirements:
  - automotive systems (modern cars have 10-20 embedded boards connected by one or more CAN bus)
  - avionics systems (several distributed control boards connected by one or more dedicated networks)

- To analyze the schedulability of such systems, it is very important to estimate the (worst-case) computation times of the tasks
- Estimating WCET is very difficult in modern architectures

Real-time pipelines

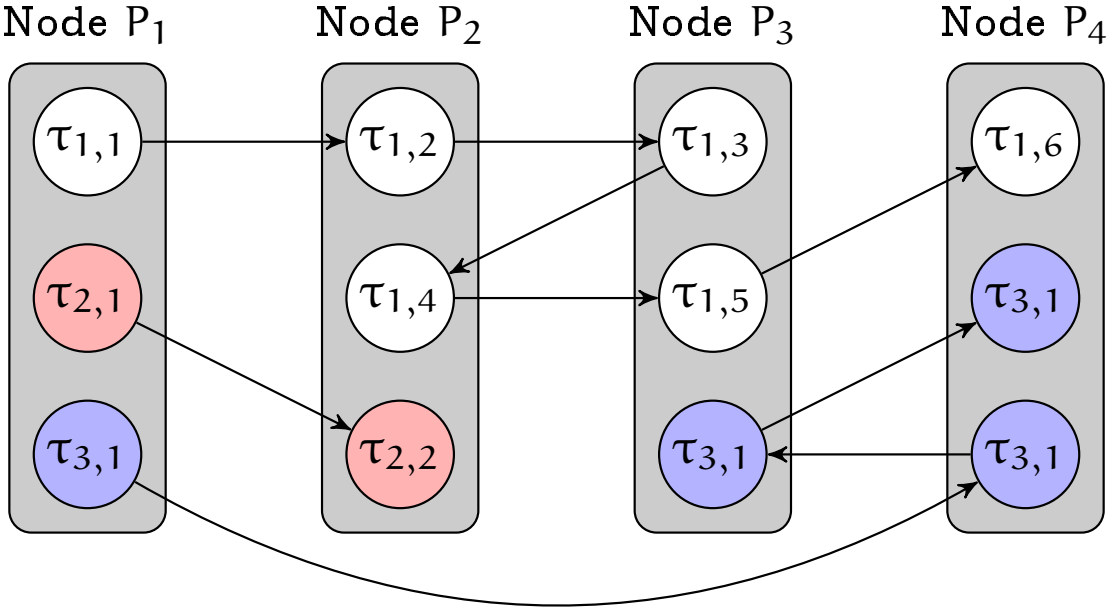
- Many real-time applications can be modeled as pipelines (also called transactions) of tasks



- Executed on a distributed (or multicore) system
- Activated cyclically (periodic or sporadic)

Model

- A set of pipelines $\{\mathcal{P}^{(1)}, \ldots, \mathcal{P}^{(p)}\}$ distributed over m nodes
- Each pipeline $\mathcal{P}^{(i)}$ is a chain of n_i tasks $\{\tau_{i,1}, \ldots, \tau_{i,n_i}\}$
- Pipeline $\mathcal{P}^{(i)}$ has an end-to-end (E2E) deadline D_i and period T_i

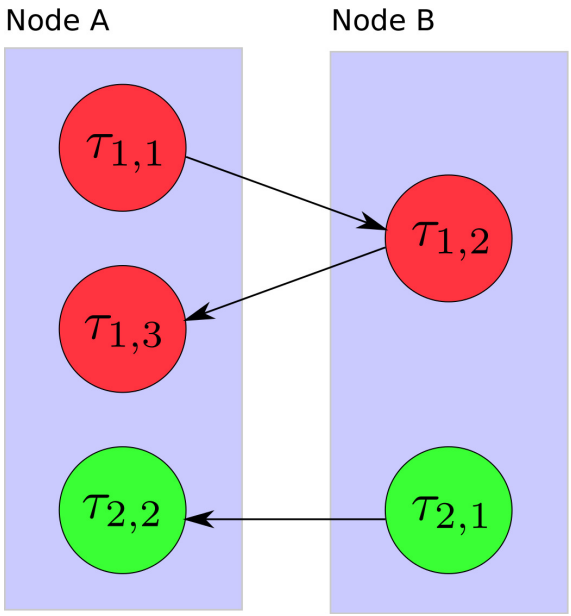


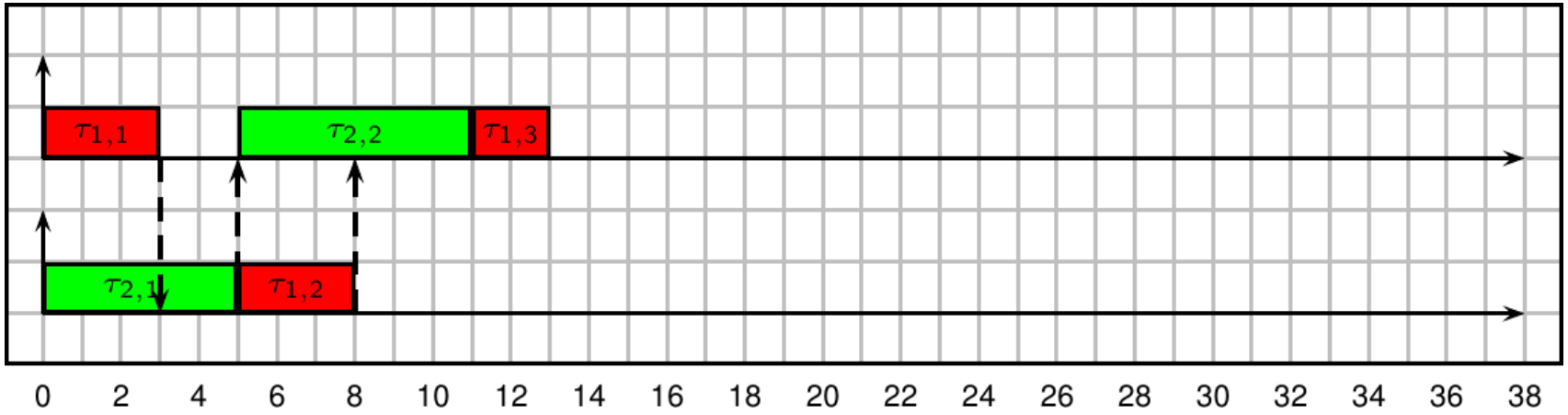
- Scheduling Problem: Guarantee that all pipelines complete before their E2E deadlines

Activations, jitter, deadline

- An example

Task	Per.	E2E	Comp.	Resp.	Jitter	prio
$\tau_{1,1}$	15		3	3	0	HI
$\tau_{1,2}$	-		3	8	0-2	LO
$\tau_{1,3}$	-	15	2	13	3	LO
$\tau_{2,1}$	12		5	6	0	HI
$\tau_{2,2}$	-	12	6	?	0-1	ME

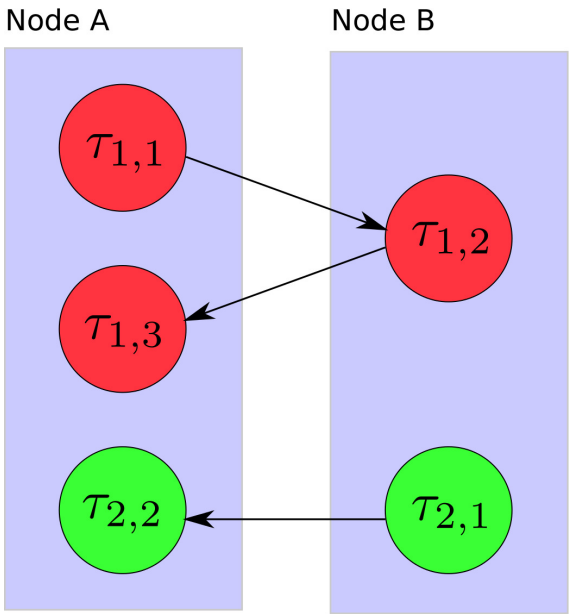


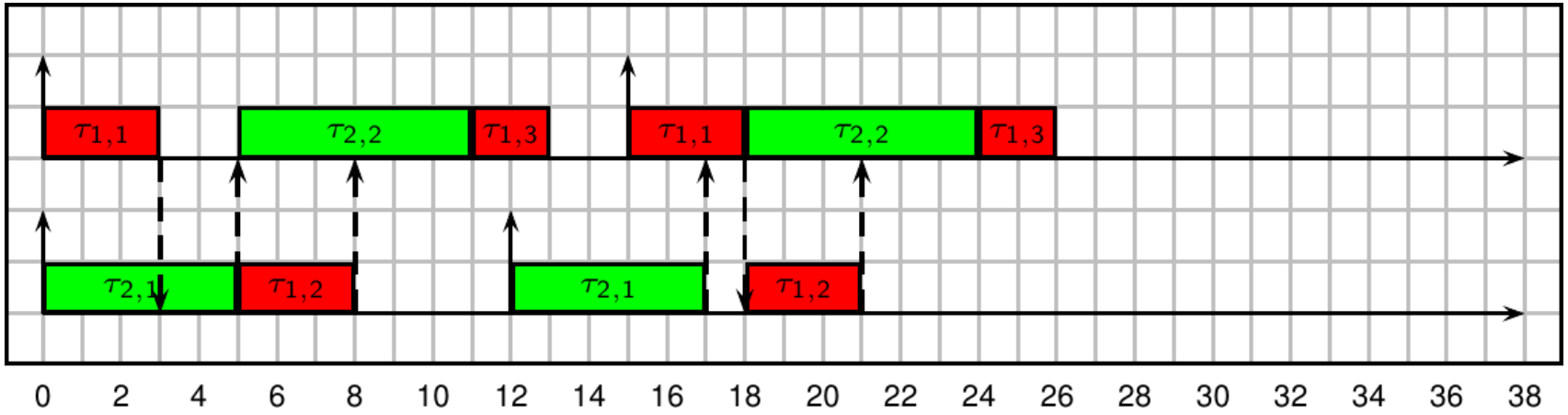


Activations, jitter, deadline

- An example

Task	Per.	E2E	Comp.	Resp.	Jitter	prio
$\tau_{1,1}$	15		3	3	0	HI
$\tau_{1,2}$	-		3	8	0-2	LO
$\tau_{1,3}$	-	15	2	13	3	LO
$\tau_{2,1}$	12		5	6	0	HI
$\tau_{2,2}$	-	12	6	?	0-1	ME

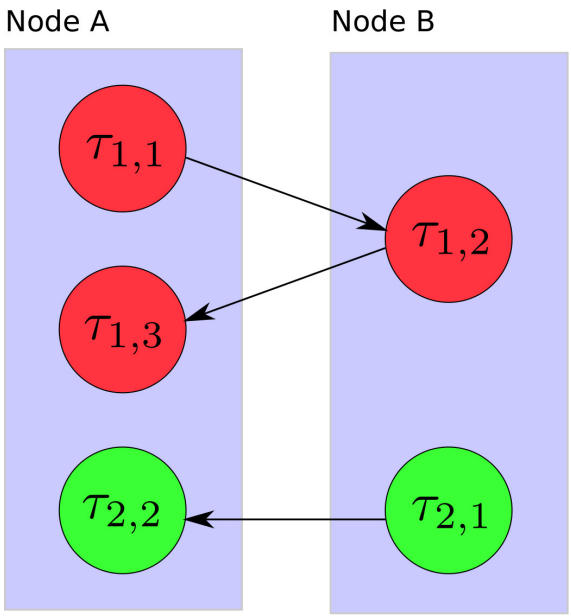


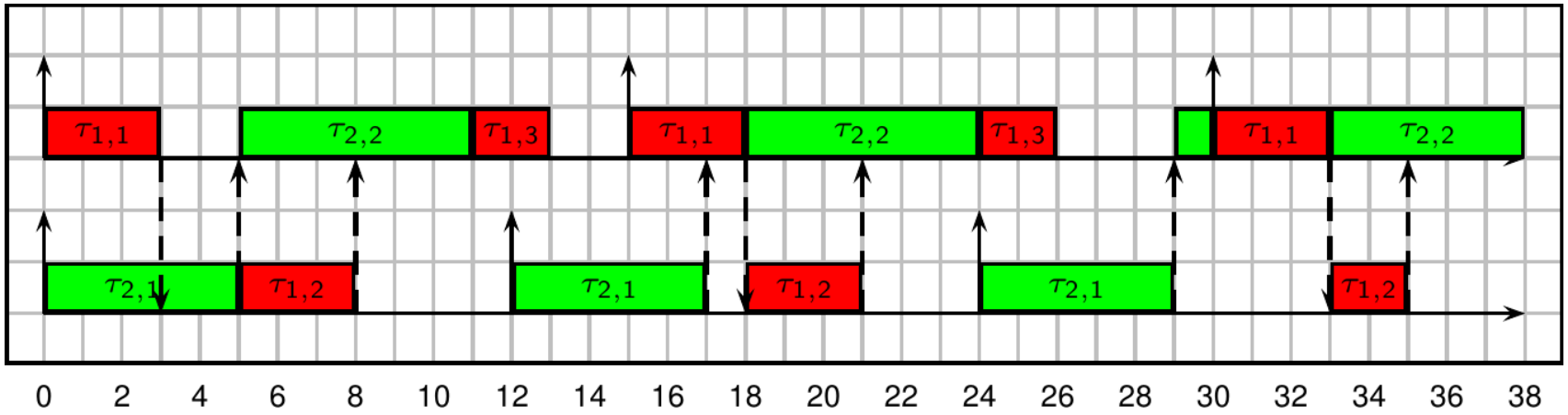


Activations, jitter, deadline

- An example

Task	Per.	E2E	Comp.	Resp.	Jitter	prio
$\tau_{1,1}$	15		3	3	0	HI
$\tau_{1,2}$	-		3	8	0-2	LO
$\tau_{1,3}$	-	15	2	13	3	LO
$\tau_{2,1}$	12		5	6	0	HI
$\tau_{2,2}$	-	12	6	**14**	0-1	ME





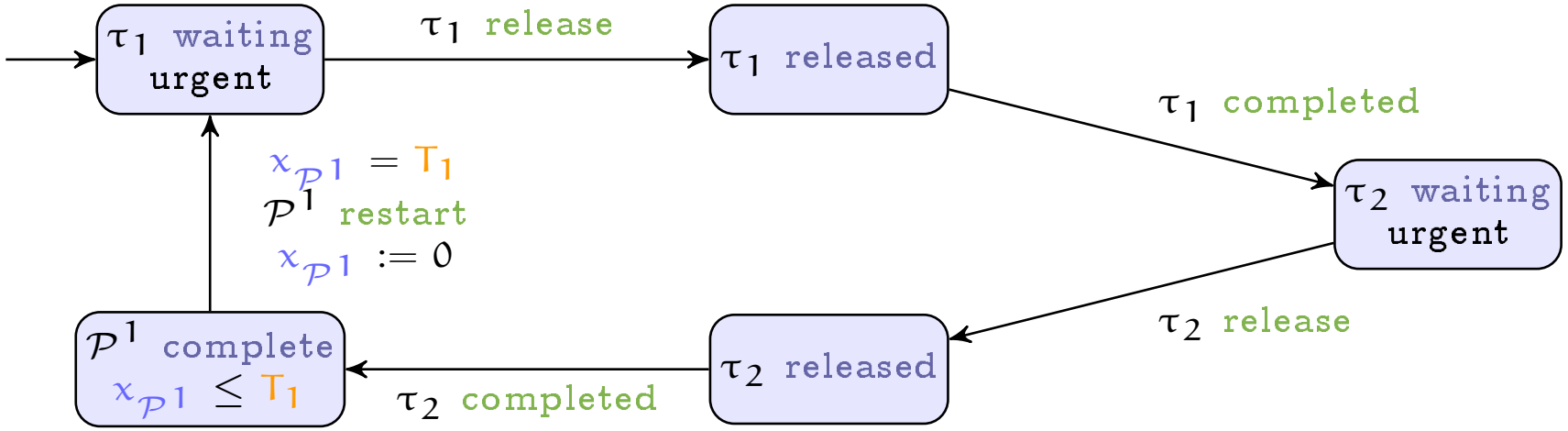
Model of the network

- Tasks executing on processors are scheduled by Full Preemptive Fixed Priority Scheduler (FPFPS)
- Tasks send messages over the network to signal finishing of instance and activating the subsequent task
- Messages are scheduled over the network according to Non-Preemptive Fixed Priority
  - In the industrial domain, the CAN Bus features such a policy

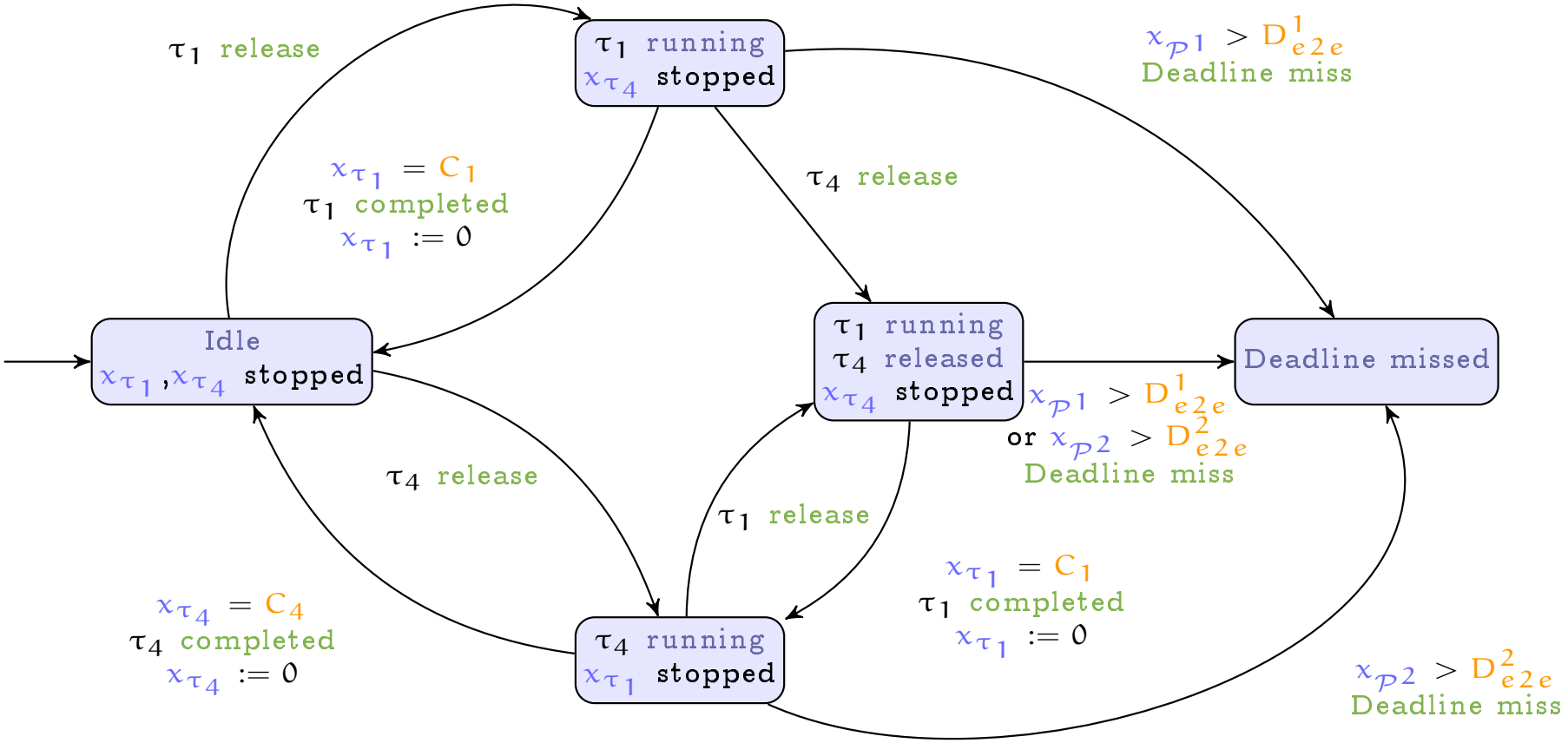
Parametric model

- A task is identified by three parameters:
  - C_i is the worst-case computation time (or worst-case transmission time, in case it models a message)
  - R_i is the task worst-case response time, i. e., the worst case finishing time of any task instance relative to the activation of its pipeline.
  - J_i is the task worst-case activation jitter, i. e., the greatest time since its activation that a task must wait for all preceding tasks to complete their execution
- The only one of interest is the computation time

Modeling a task / pipeline



Modeling the fixed priority scheduler (preemptive)



Actually a PTA extended with stopwatches [Sun et al., 2013]

an extension of stopwatch automata [Adbeddaïm and Maler, 2002]

Parametric verification of real-time systems

Many problems can be reduced to parametric reachability (EFsynth):
find parameter valuations for which a given state is unreachable

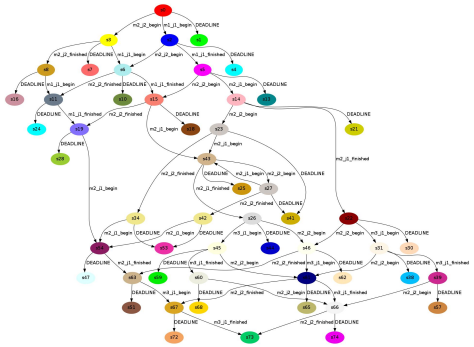
- ☹ This problem is undecidable for PTAs and many subclasses [A., FTSCS 2015]
- ☺ But we can still compute part (and often all) of the solution

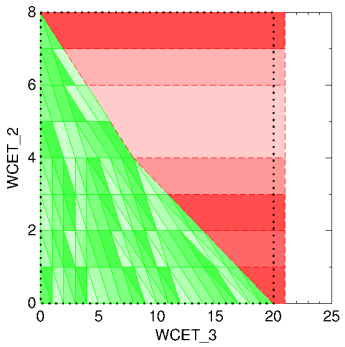
Interesting problems:

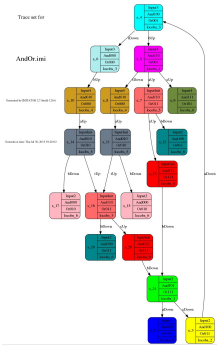
- Find parameter valuations for which no deadline violation occurs (i. e., for which the system is schedulable)
- Compute the worst-case computation time
- Find the parametric WCET when the jitter is unknown

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
- Verification
  - Computation of the symbolic state space
  - Parametric model checking (using a subset of TCTL)
  - Language and trace preservation, and robustness analysis
  - Parametric deadlock-freeness checking
  - Behavioral cartography







IMITATOR

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

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., 2012]

A library of benchmarks

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

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

Try it!

www.imitator.fr

Some success stories

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

Outline

1 Parametric Timed Automata

2 Modeling and Verifying Real-Time Systems

3 Verifying a Real-time System under Uncertainty

4 Distributed Verification of Distributed Systems

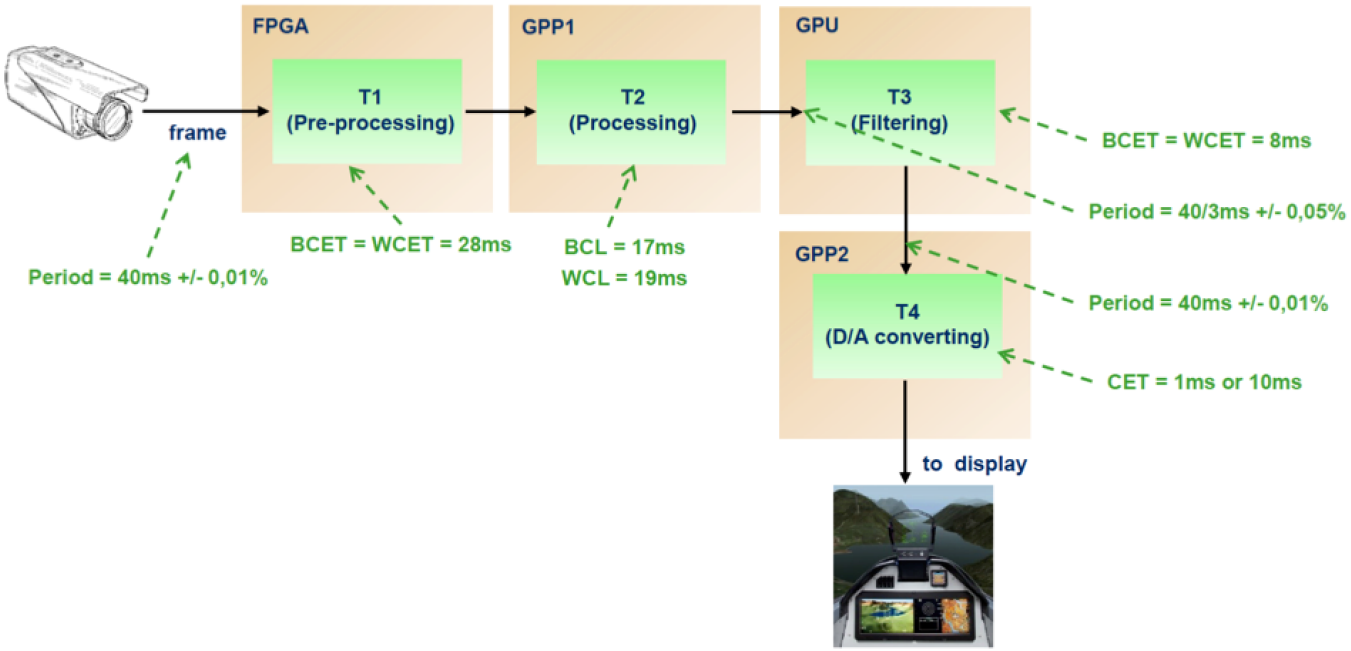
5 Conclusion and Perspectives

The FMTV 2015 Challenge (1/2)

Challenge by Thales proposed during the WATERS 2014 workshop
Solutions presented at WATERS 2015

System: an unmanned aerial video system with uncertain periods

- Period constant but with a small uncertainty (typically 0.01 %)
- Not a jitter!



The FMTV 2015 Challenge (2/2)

Goal

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

The FMTV 2015 Challenge (2/2)

Goal

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

☹ Not a typical parameter synthesis problem?

- No parameters in the specification

The FMTV 2015 Challenge (2/2)

Goal

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

☹ Not a typical parameter synthesis problem?

- No parameters in the specification

☺ A typical parameter synthesis problem

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

Methodology

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

Methodology

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

Methodology

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

Methodology

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

Methodology

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

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

Methodology

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

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

To build the PTA model

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

To build the PTA model

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

To build the PTA model

- Uncertainties in the system:
  - $P1 \in [40 - 0.004, 40 + 0.004]$
  - $P3 \in [\frac{40}{3} - \frac{1}{150}, \frac{40}{3} + \frac{1}{150}]$
  - $P4 \in [40 - 0.004, 40 + 0.004]$
- Parameters:
  - P1_uncertain
  - P3_uncertain
  - P4_uncertain
- The end-to-end latency (another parameter): E2E

To build the PTA model

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

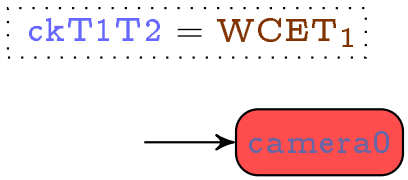
Simplification

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

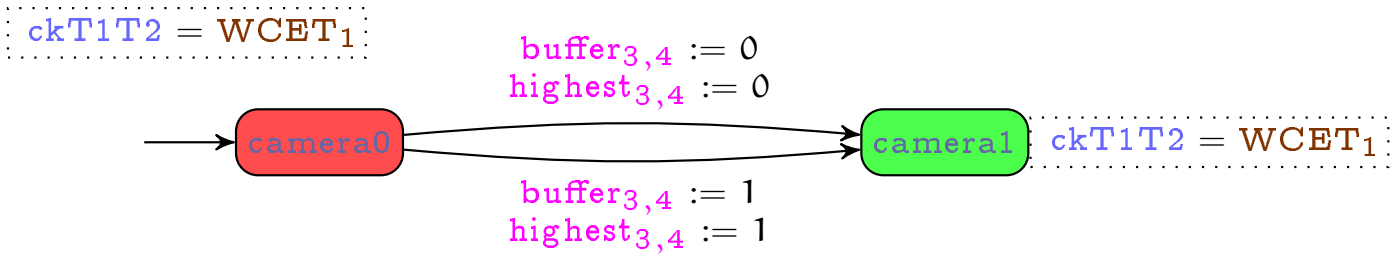
Simplification

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

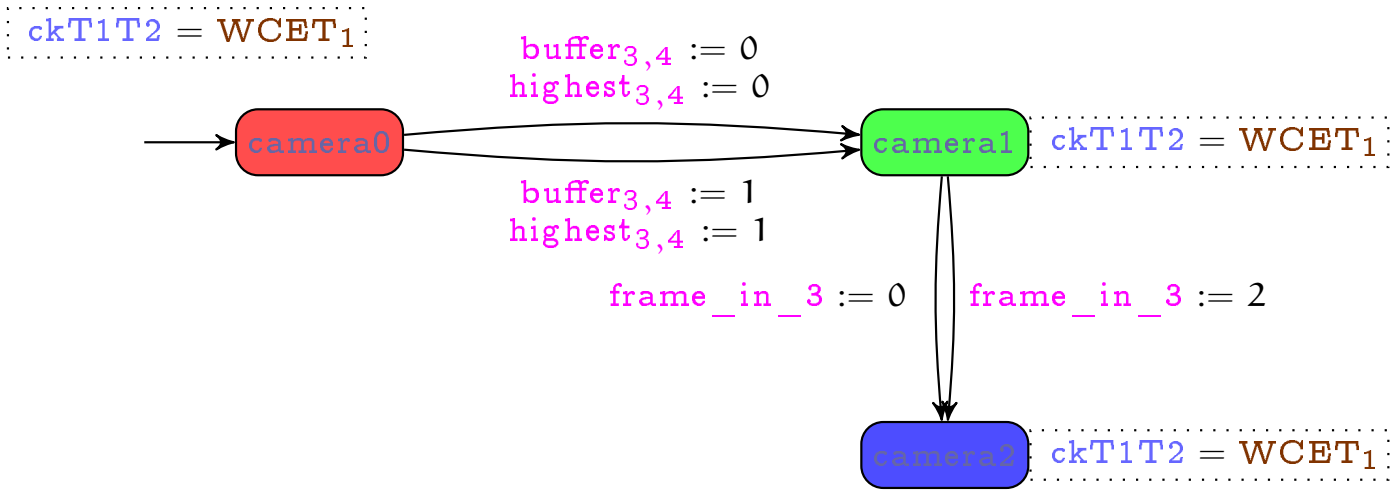
The initialization automaton



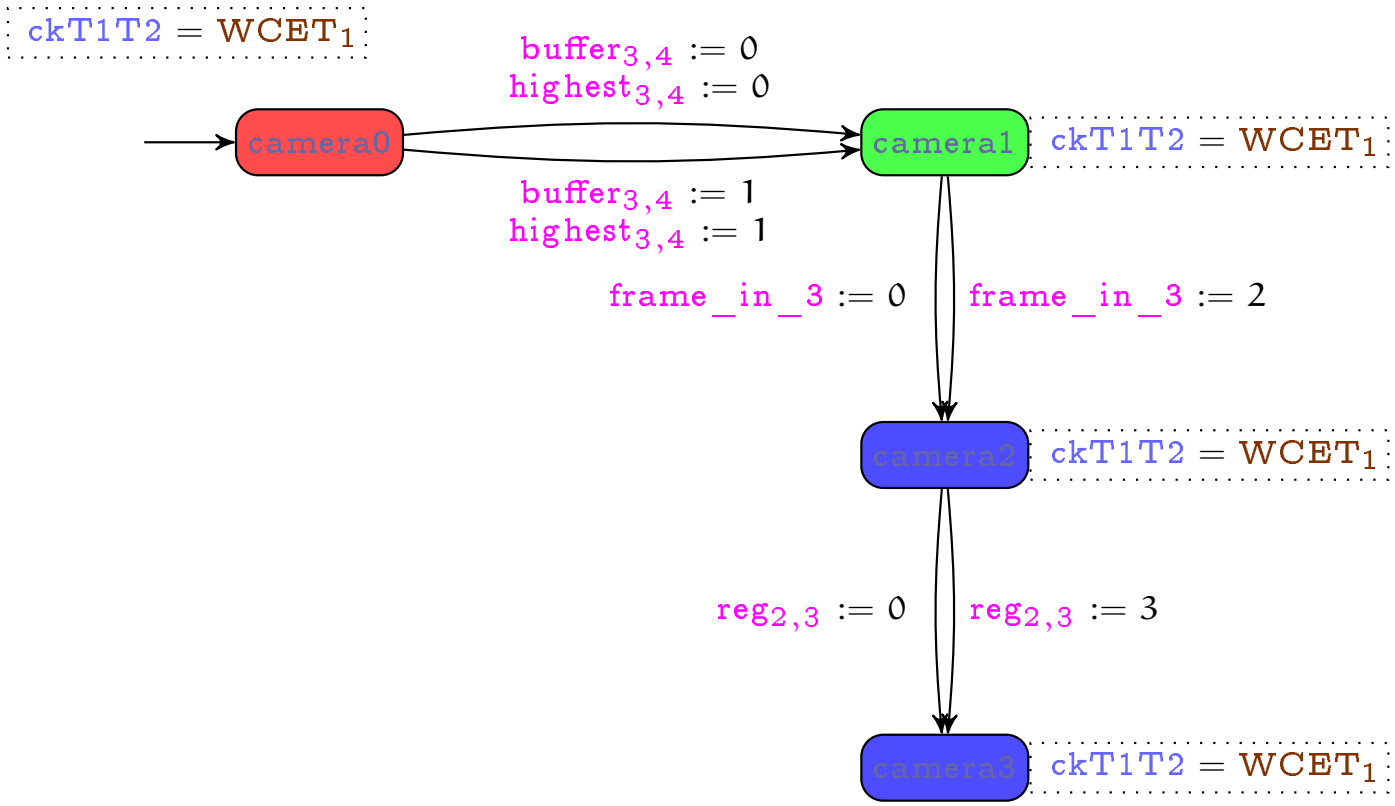
The initialization automaton



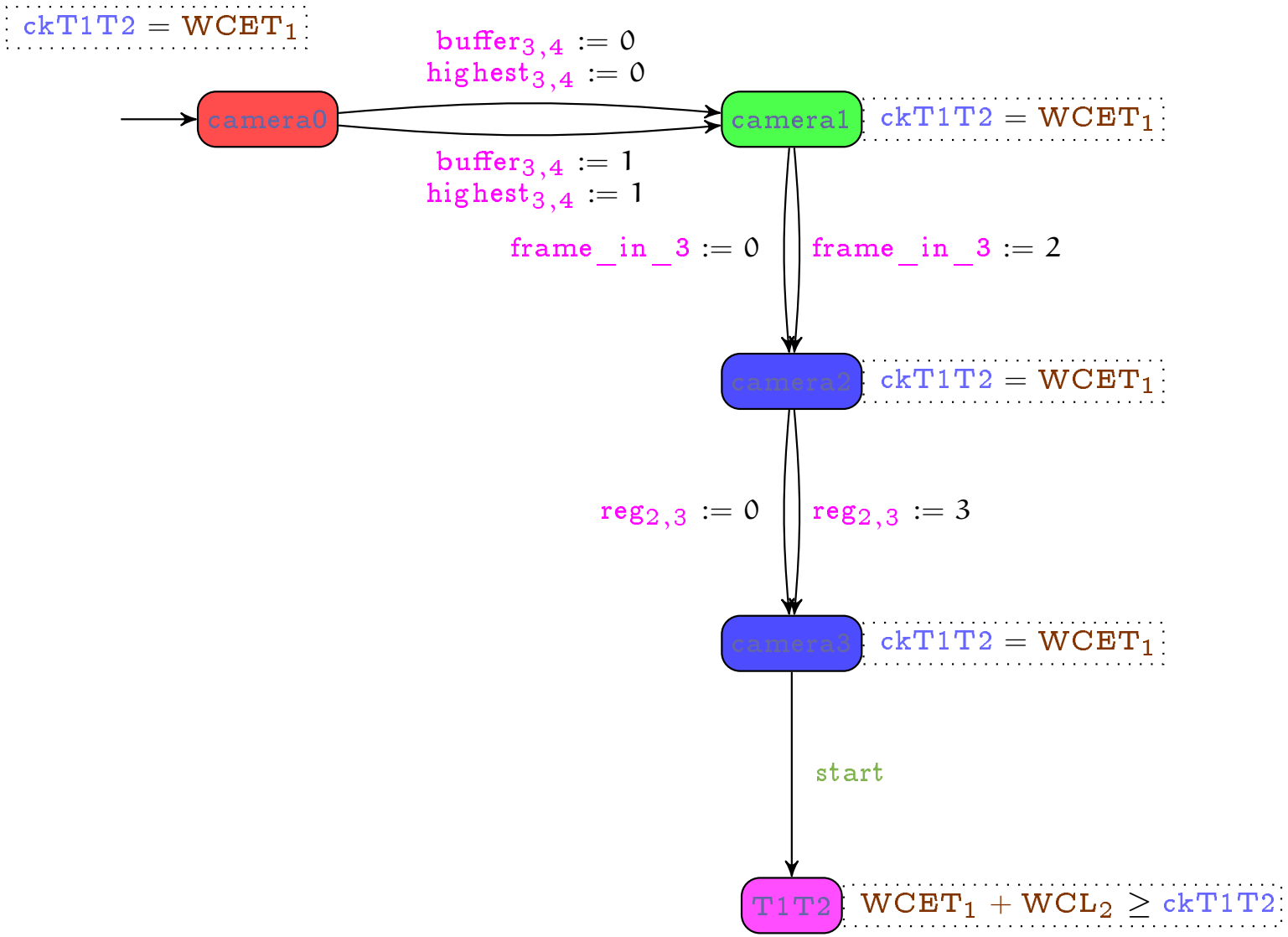
The initialization automaton



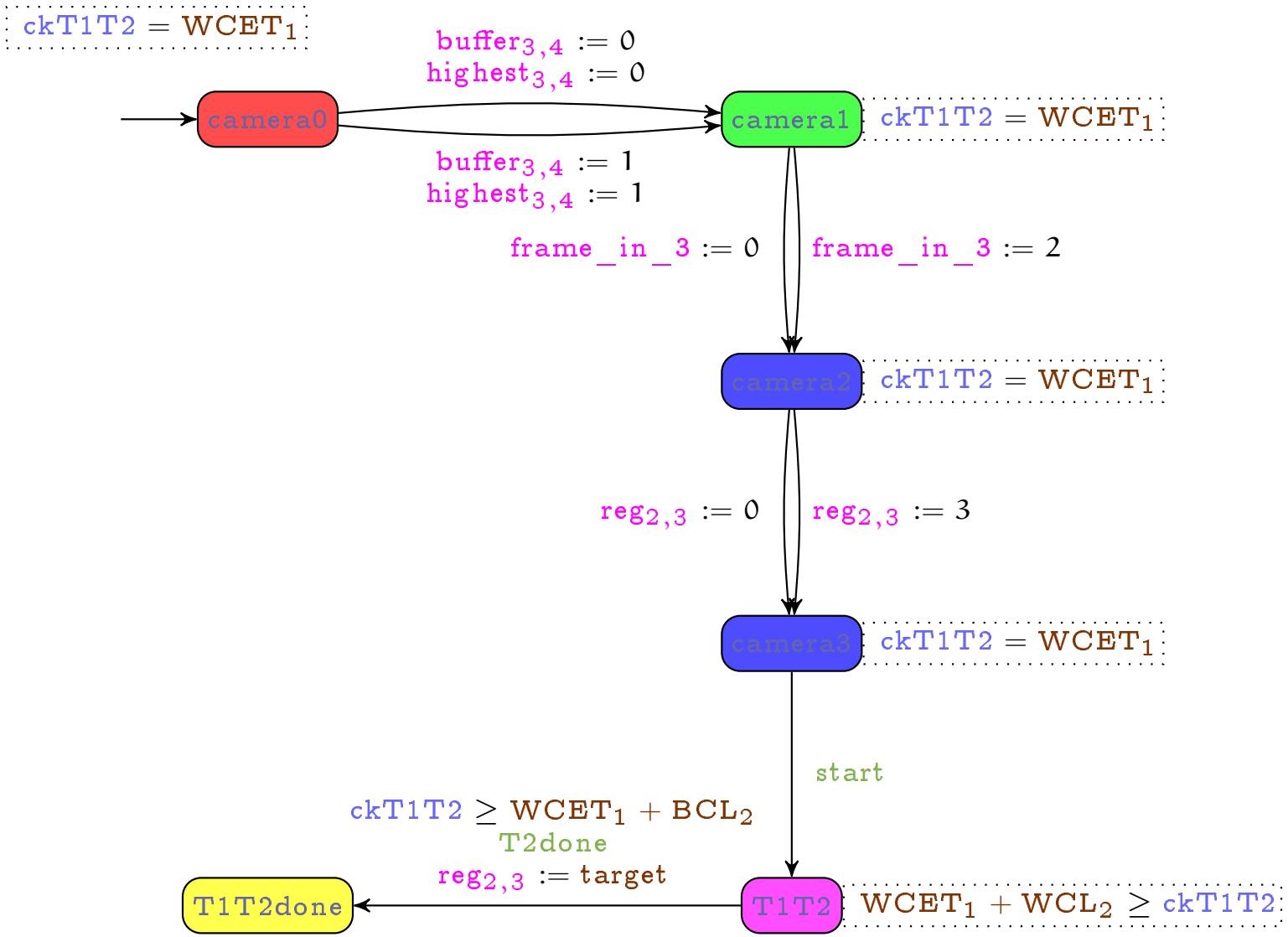
The initialization automaton



The initialization automaton

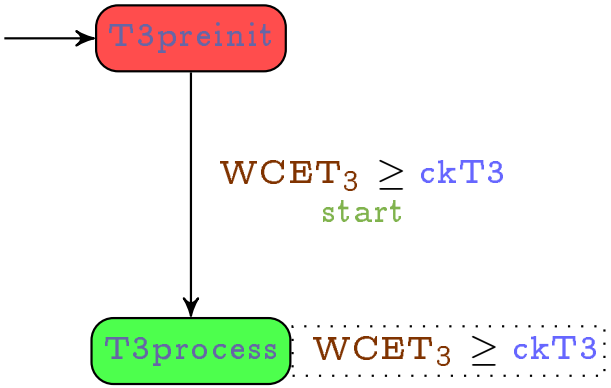


The initialization automaton

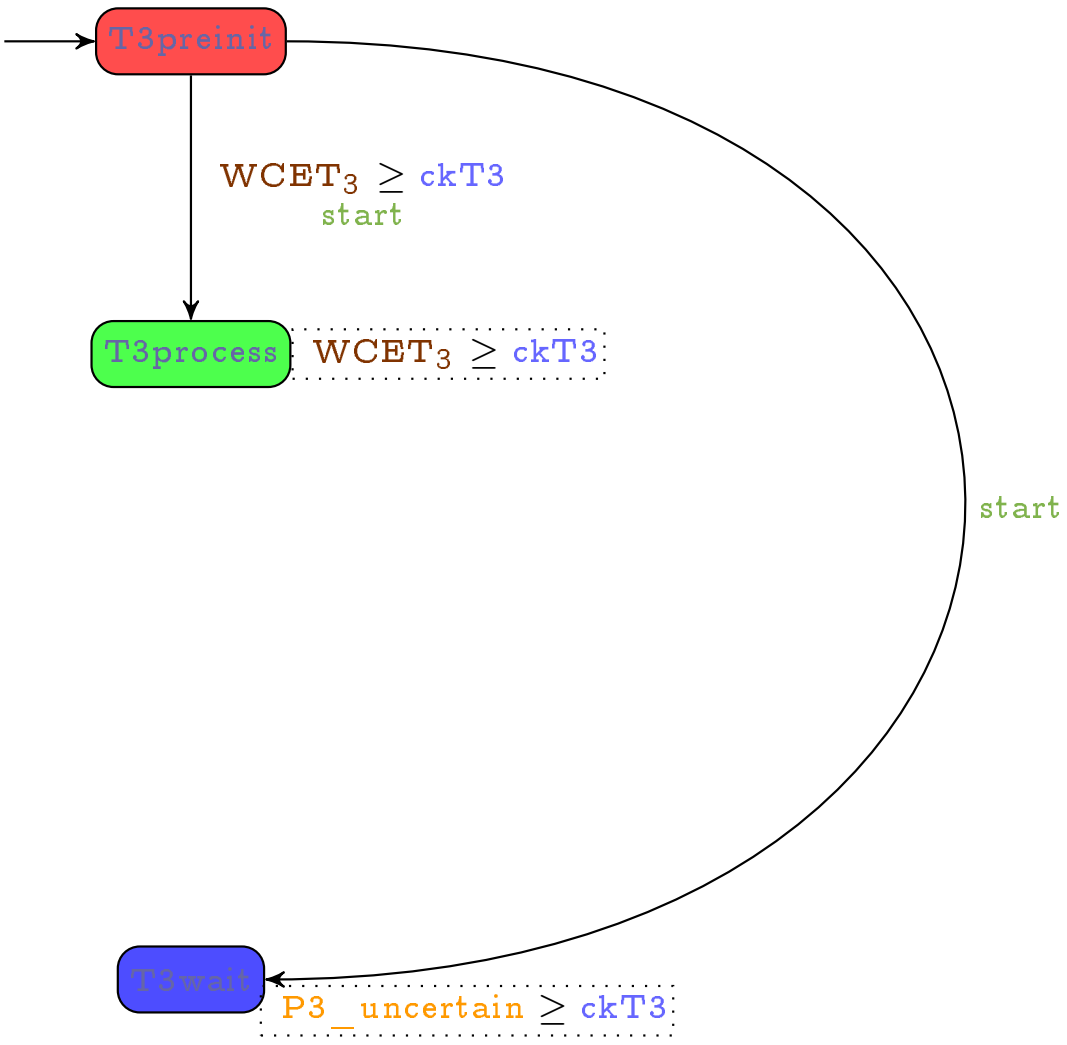


Task T3

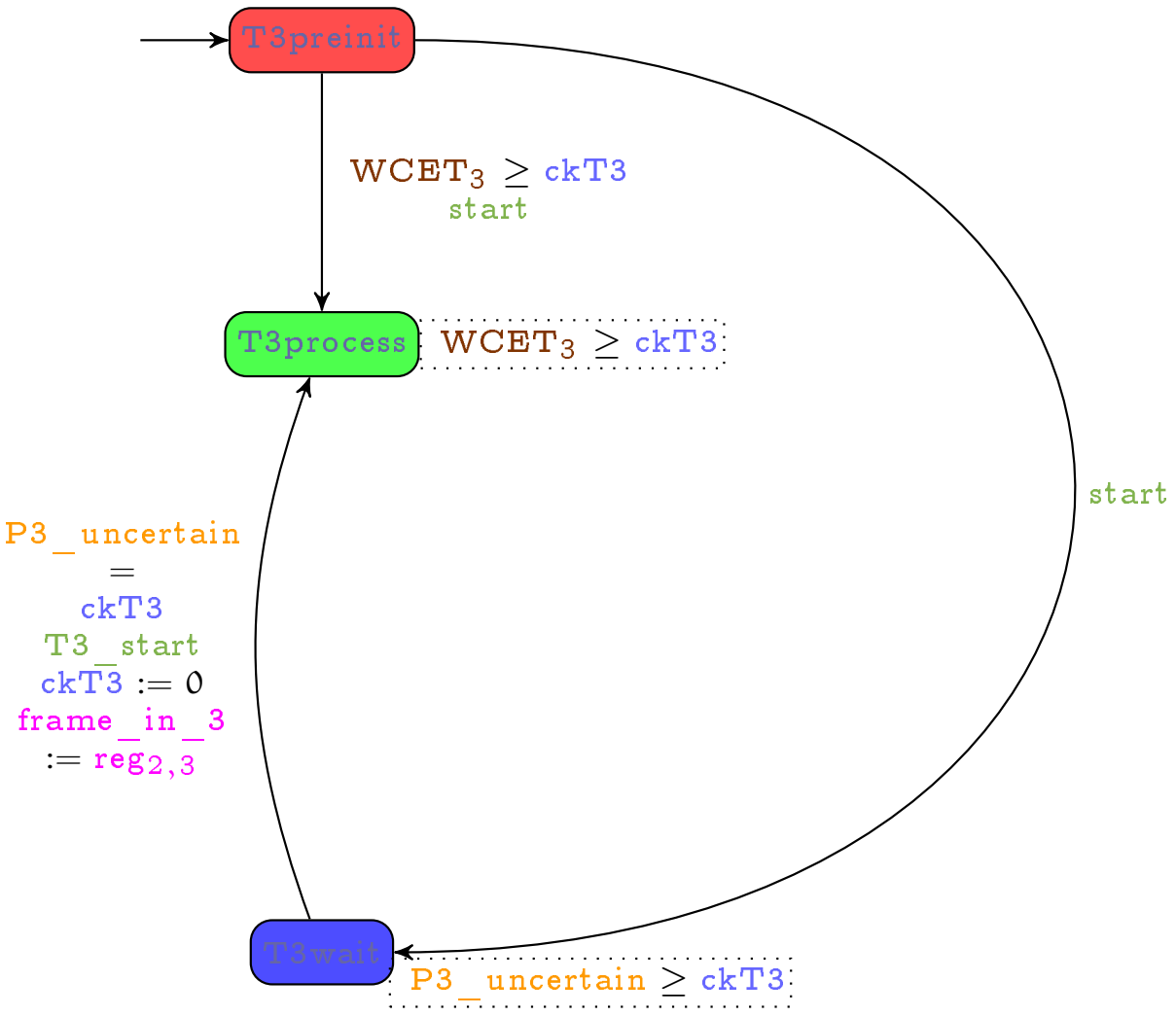
Task T3


T3preinit
$\mathrm{WCET}_3 \geq \mathrm{ckT3}$
start
T3process
$\mathrm{WCET}_3 \geq \mathrm{ckT3}$

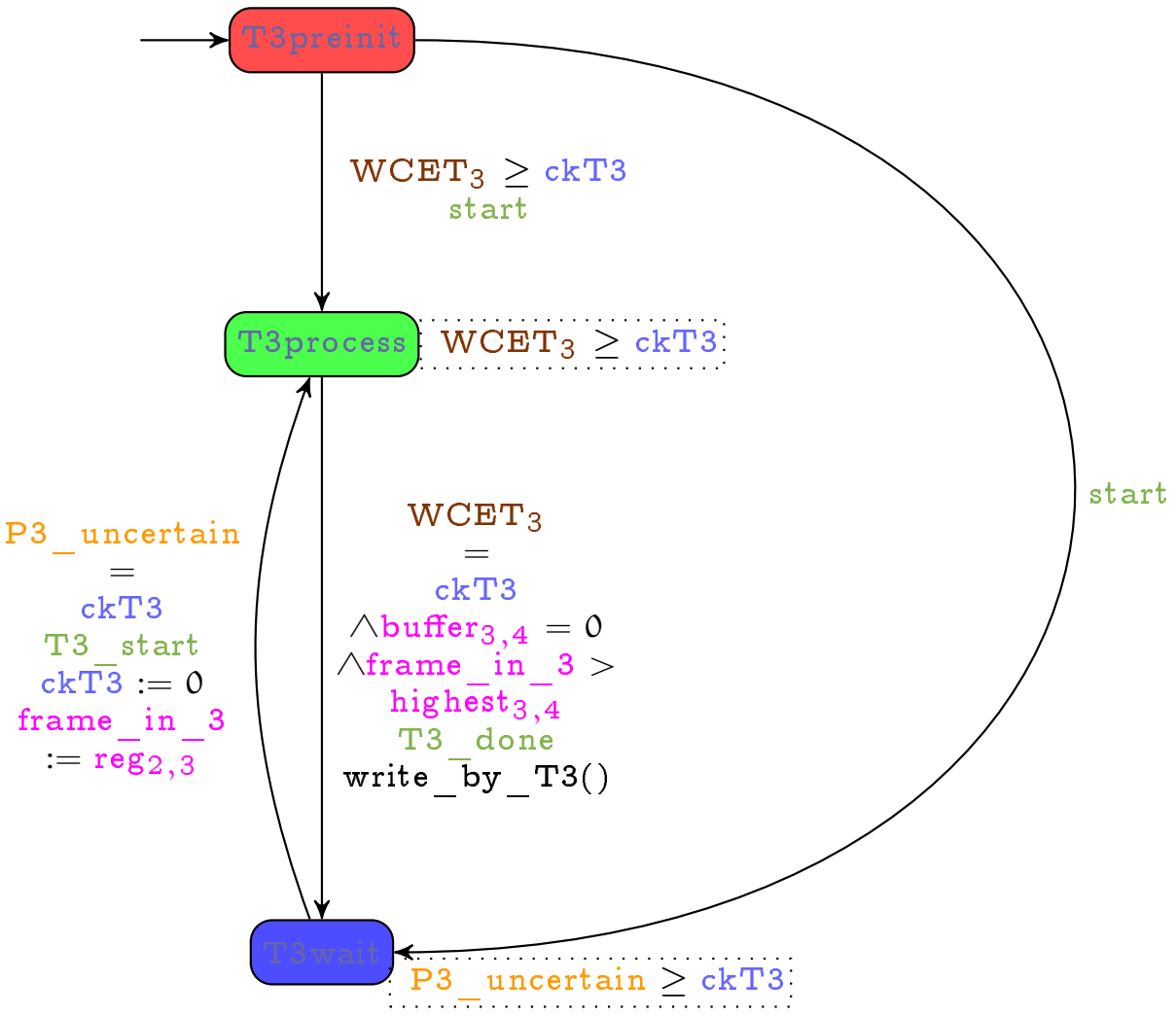
Task T3


T3preinit
$WCET_3 \geq ckT3$
start
T3process
$WCET_3 \geq ckT3$
start
T3wait
$P3\_uncertain \geq ckT3$

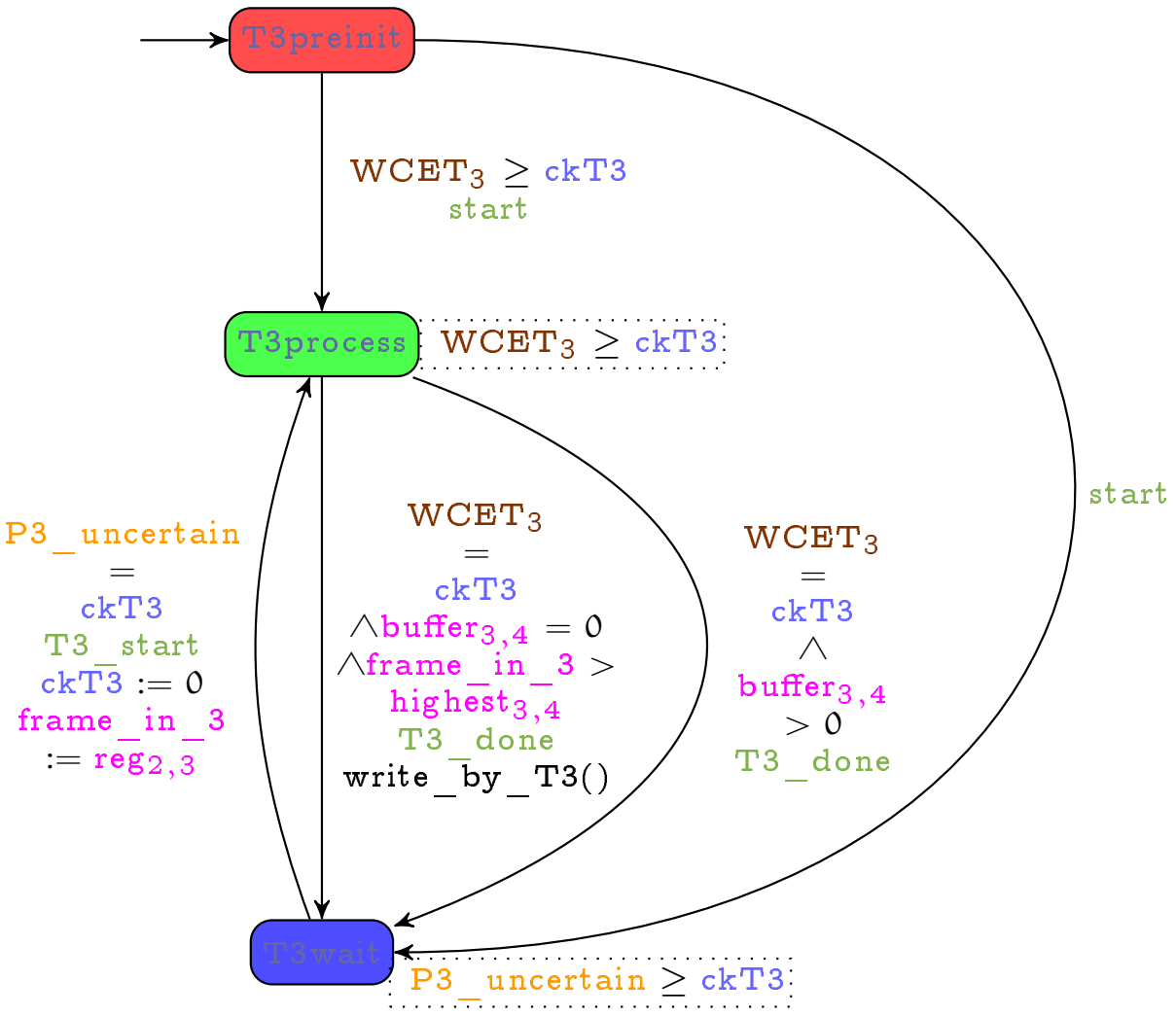
Task T3


T3preinit
$WCET_3 \geq ckT3$
start
T3process
$WCET_3 \geq ckT3$
start
P3_uncertain
=
ckT3
T3_start
ckT3 := 0
frame_in_3
:= $reg_{2,3}$
T3wait
P3_uncertain $\geq$ ckT3

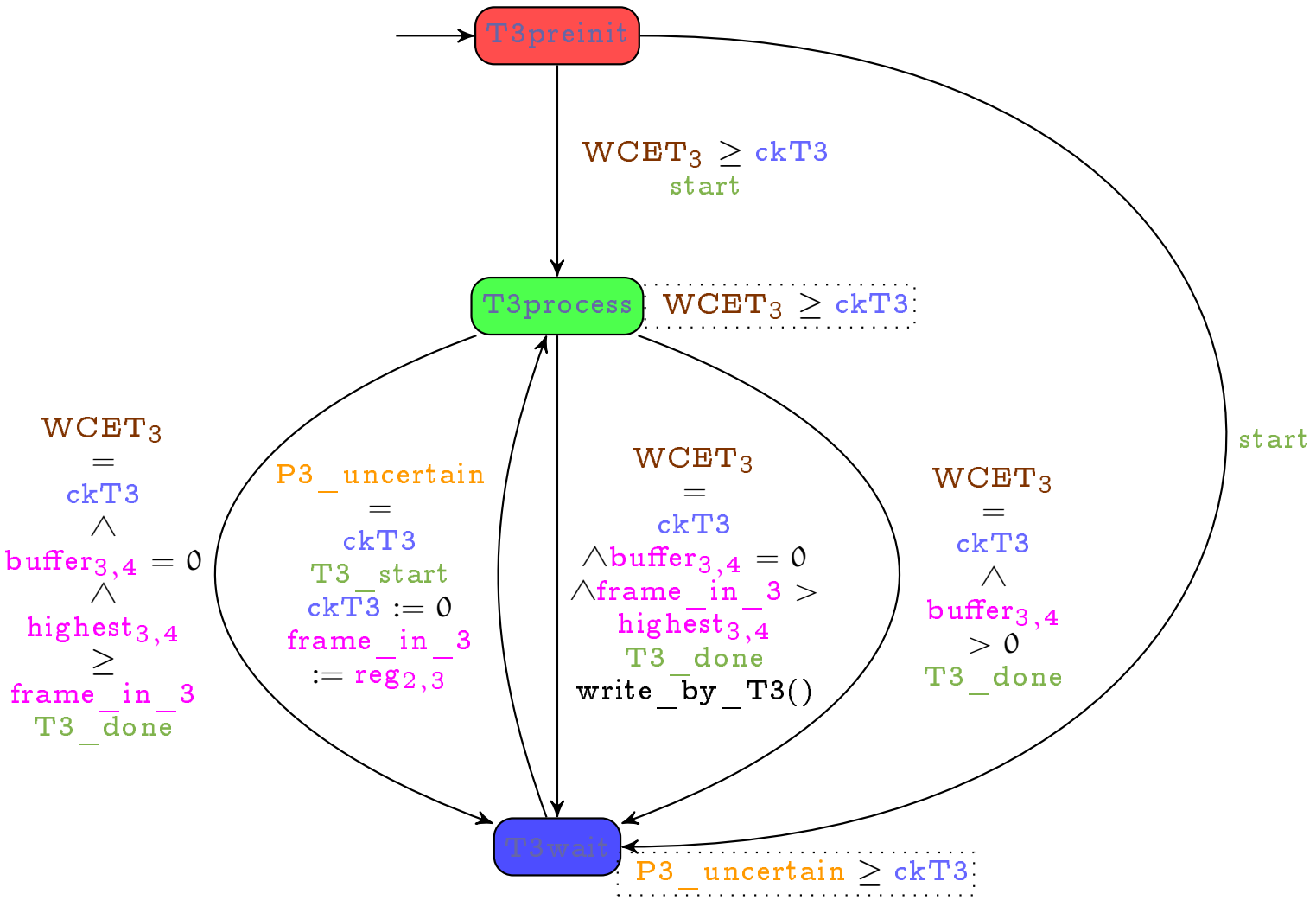
Task T3


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

Task T3

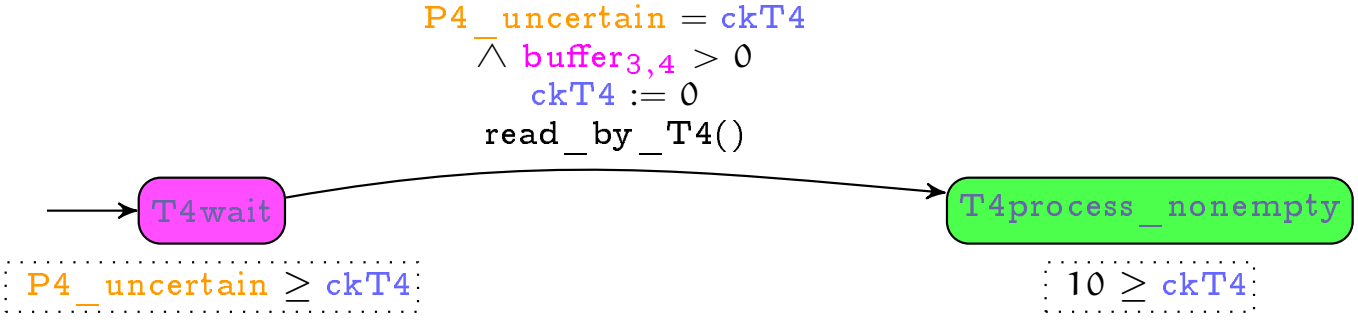

T3preinit
WCET3 ≥ ckT3
start
T3process
WCET3 ≥ ckT3
start
P3_uncertain
=
ckT3
T3_start
ckT3 := 0
frame_in_3
:= reg2,3
WCET3
=
ckT3
∧buffer3,4 = 0
∧frame_in_3 >
highest3,4
T3_done
write_by_T3()
WCET3
=
ckT3
∧
buffer3,4
> 0
T3_done
T3wait
P3_uncertain ≥ ckT3

Task T3

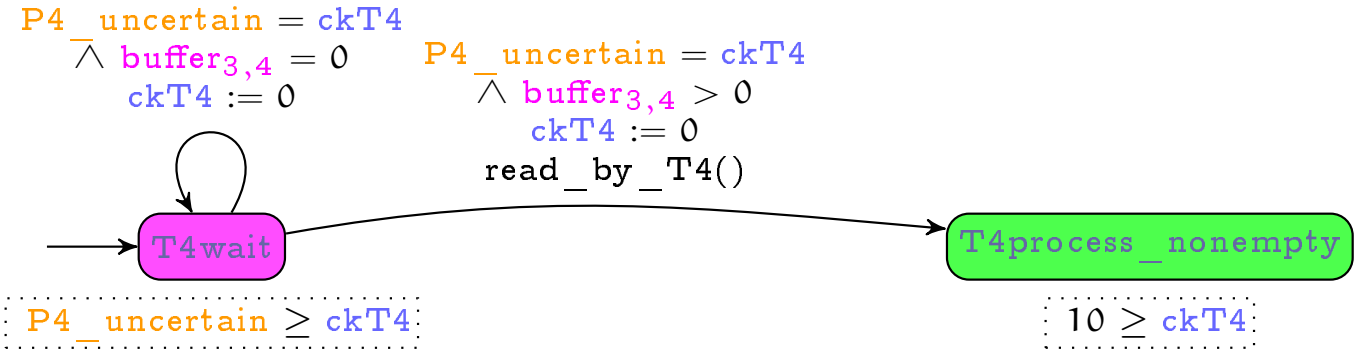

T3preinit
WCET3 ≥ ckT3
start
T3process
WCET3 ≥ ckT3
start
WCET3 = ckT3 ∧ buffer3,4 = 0 ∧ highest3,4 ≥ frame_in_3
T3_done
P3_uncertain = ckT3
T3_start
ckT3 := 0
frame_in_3 := reg2,3
WCET3 = ckT3 ∧buffer3,4 = 0 ∧frame_in_3 > highest3,4
T3_done
write_by_T3()
WCET3 = ckT3 ∧ buffer3,4 > 0
T3_done
T3wait
P3_uncertain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

Task T4

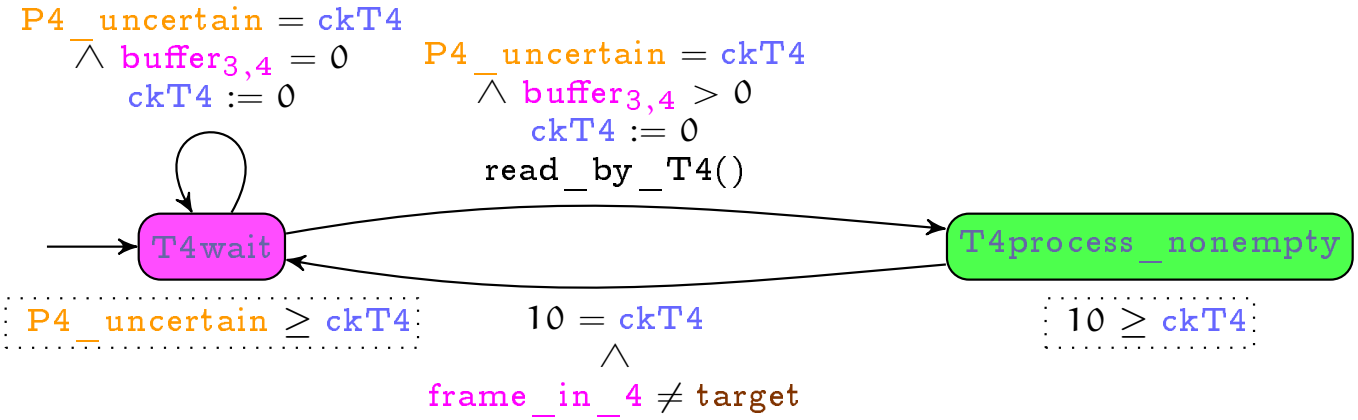
Task T4


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

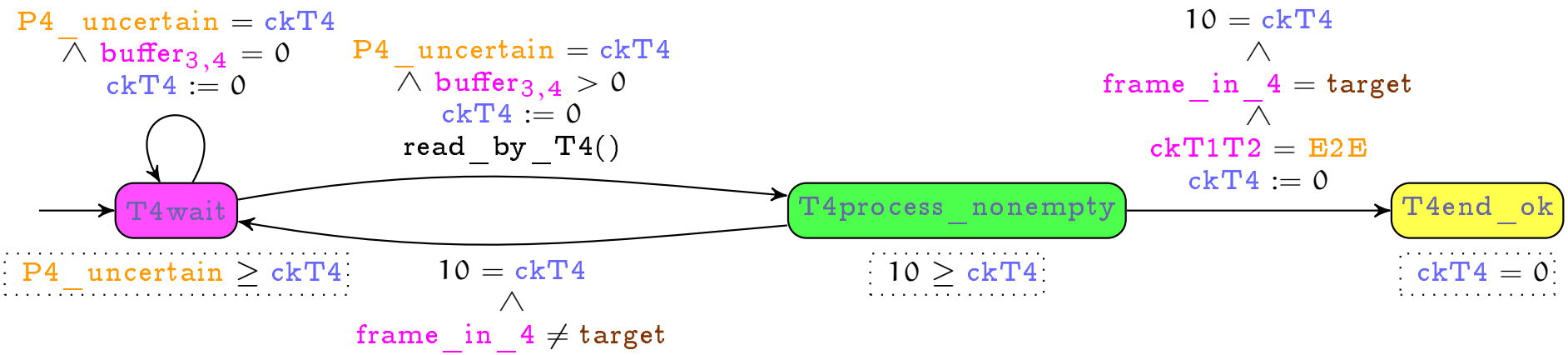
Task T4


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

Task T4


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

Task T4


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

Results

E2E latency results for $n = 1$ and $n = 3$

	$n = 1$	$n = 3$
min E2E	63 ms	63 ms
max E2E	145.008 ms	225.016 ms

Outline

1 Parametric Timed Automata

2 Modeling and Verifying Real-Time Systems

3 Verifying a Real-time System under Uncertainty

4 Distributed Verification of Distributed Systems

5 Conclusion and Perspectives

Why distributed algorithms?

Algorithms for parameter synthesis for PTA are very costly

- time
- memory

Some reasons:

- expensive operations on polyhedra
  - In IMITATOR: PPL [Bagnara et al., 2008]
- no known efficient data structure (such as BDDs or DBMs for timed automata)

Why distributed algorithms?

Algorithms for parameter synthesis for PTA are very costly

- time
- memory

Some reasons:

- expensive operations on polyhedra
  - In IMITATOR: PPL [Bagnara et al., 2008]
- no known efficient data structure (such as BDDs or DBMs for timed automata)

Idea: benefit from the power of clusters

- Cluster: large set of nodes (computers with their own memory and processor)
- Communication between nodes over a network

A first naive approach

Naive approach to distribute EFsynth:

- Each node handles a subpart of the parameter domain
- Each node launches EFsynth on its parameter domain

Drawback: bad performances if the analysis is much more costly in some subdomains than in others

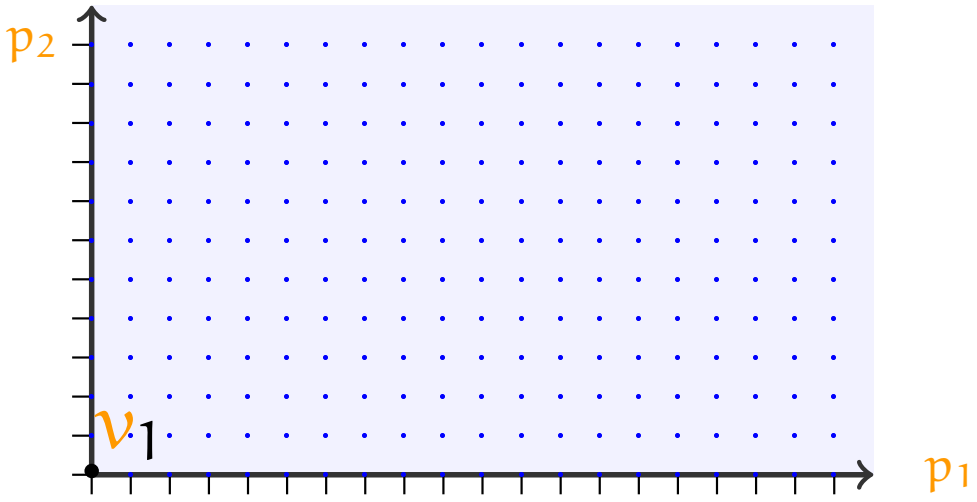
A more elaborate master-worker approach

Workers: run a "hybrid" algorithm

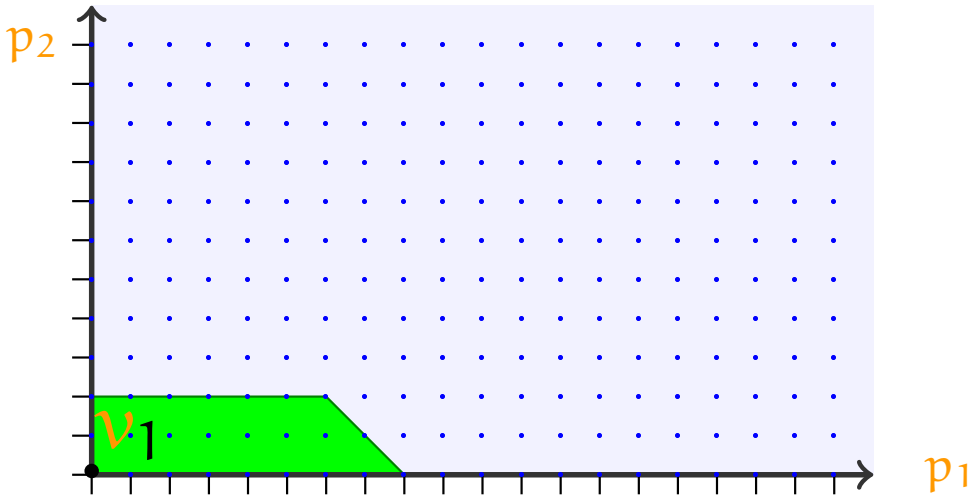
- PRP: parametric reachability preservation
- based on (integer) points: generalizes the reachability of the bad state as in the reference valuation (point)
- inspired by both EFsynth (to look for bad valuations) and TPsynth (to only explore a limited part of the symbolic state space, while "imitating" a reference valuation)
- guarantees the coverage of all integer points (but rational-valued points may be missing)

Master: responsible for gathering results and distributing reference valuations ("points") among workers

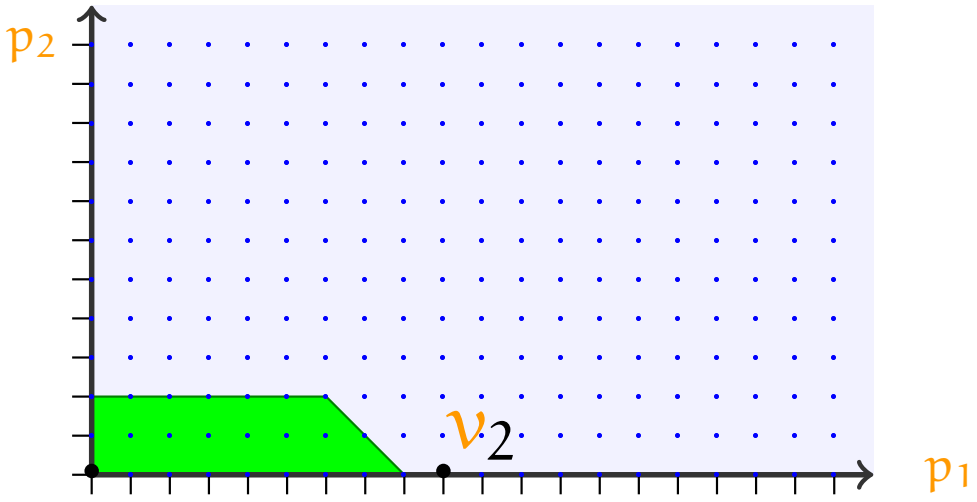
Parametric reachability preservation cartography (PRPC)



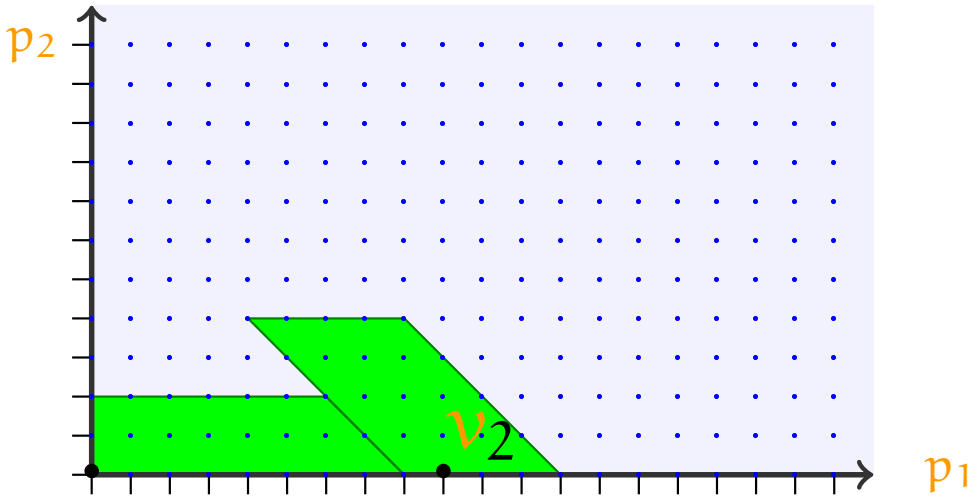
Parametric reachability preservation cartography (PRPC)



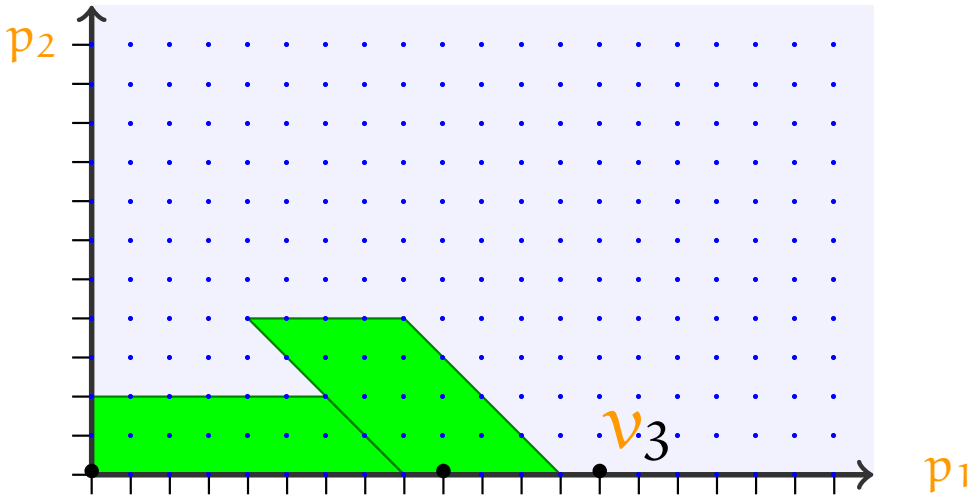
Parametric reachability preservation cartography (PRPC)



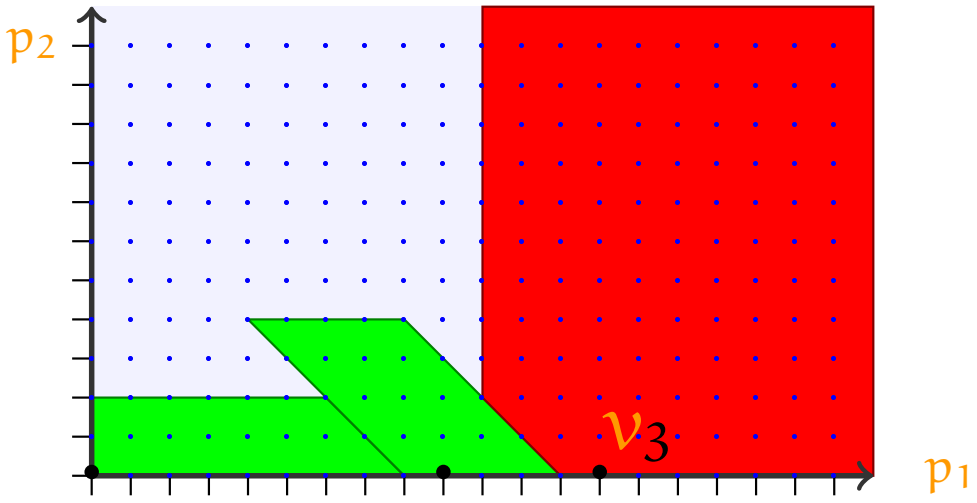
Parametric reachability preservation cartography (PRPC)



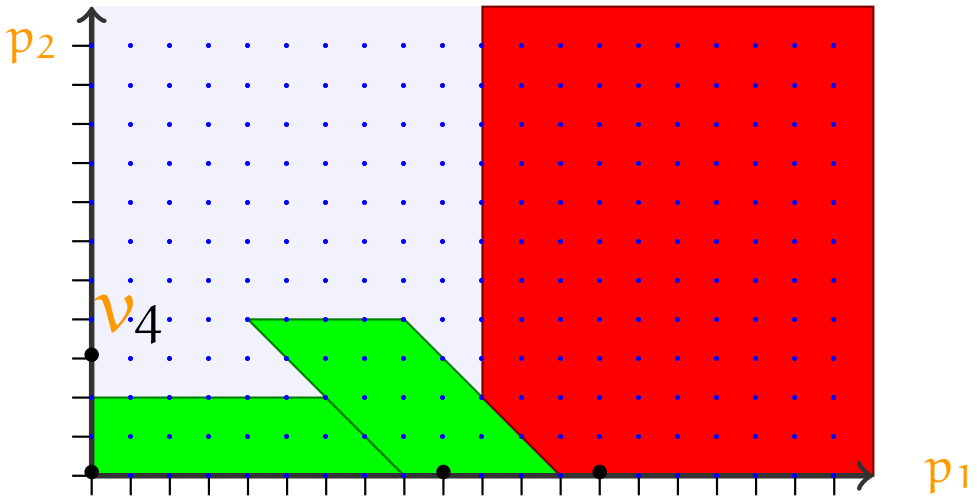
Parametric reachability preservation cartography (PRPC)



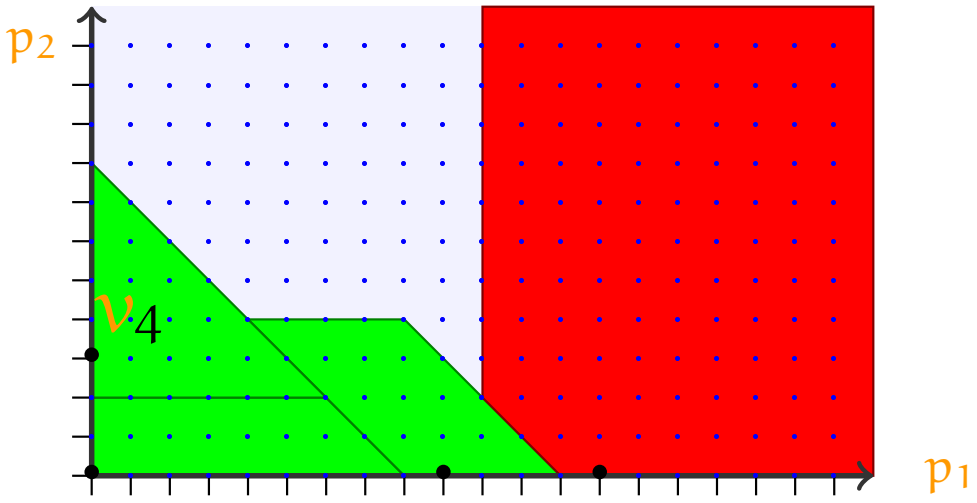
Parametric reachability preservation cartography (PRPC)



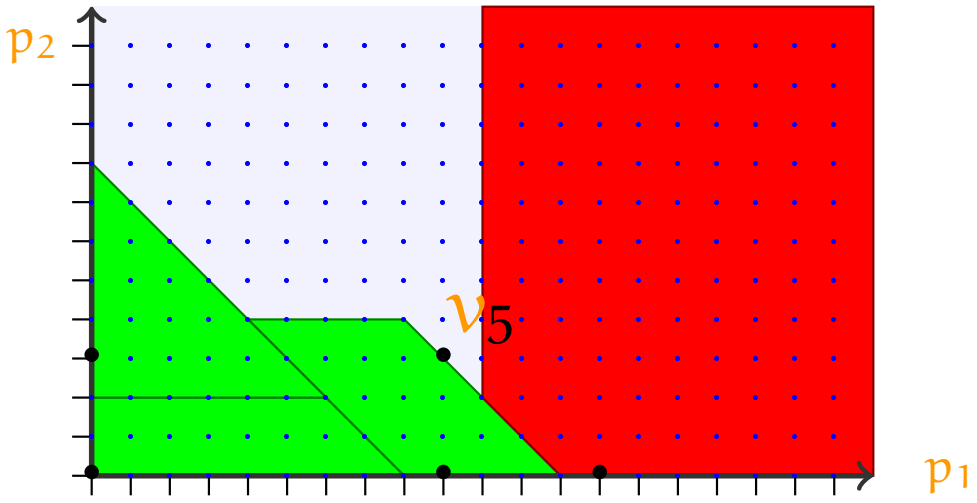
Parametric reachability preservation cartography (PRPC)



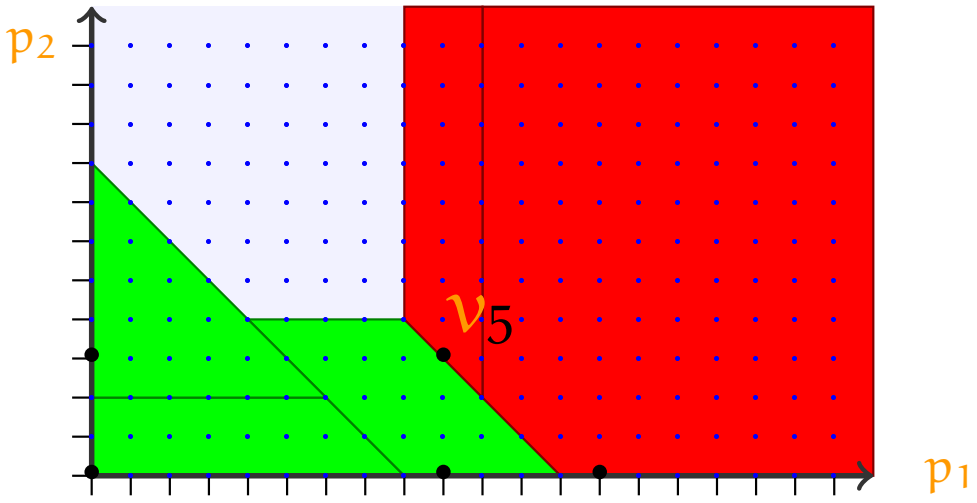
Parametric reachability preservation cartography (PRPC)



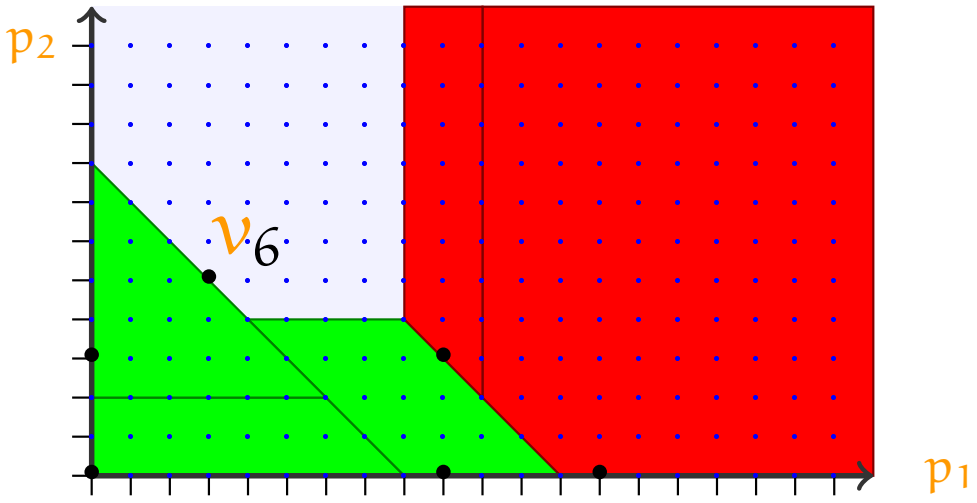
Parametric reachability preservation cartography (PRPC)



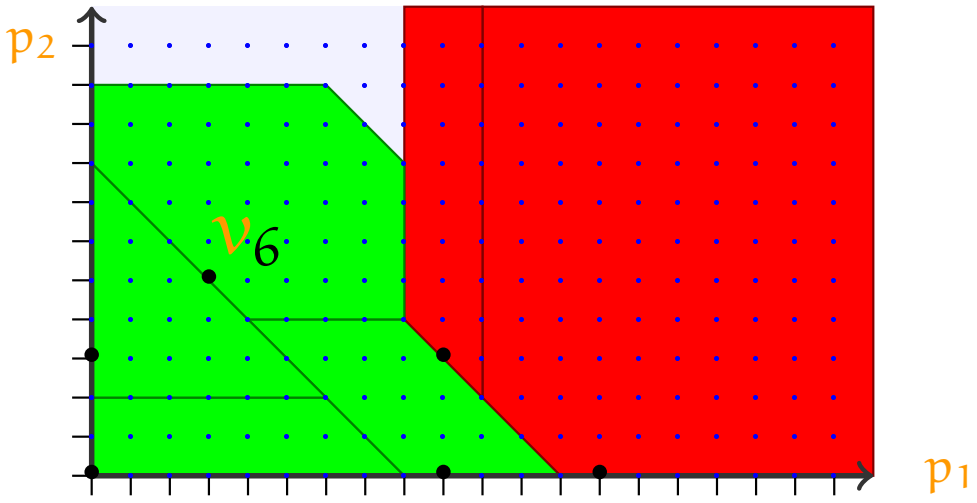
Parametric reachability preservation cartography (PRPC)



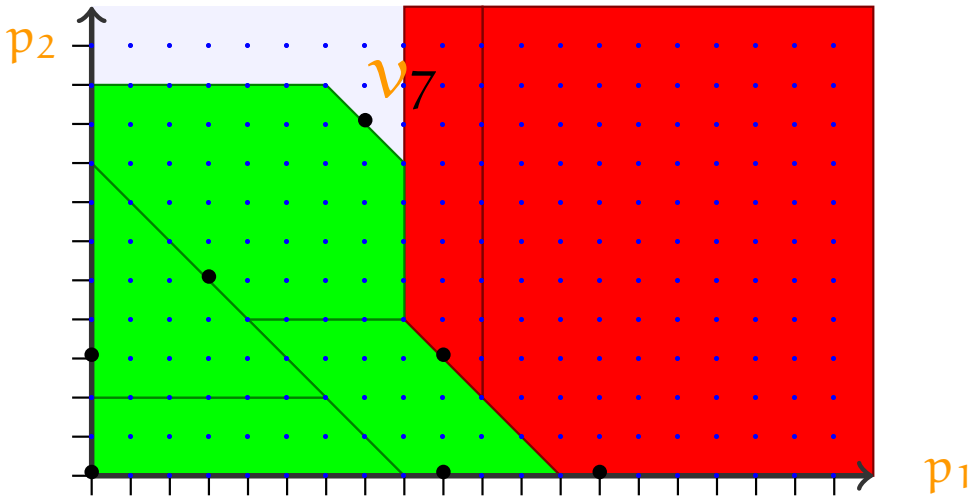
Parametric reachability preservation cartography (PRPC)



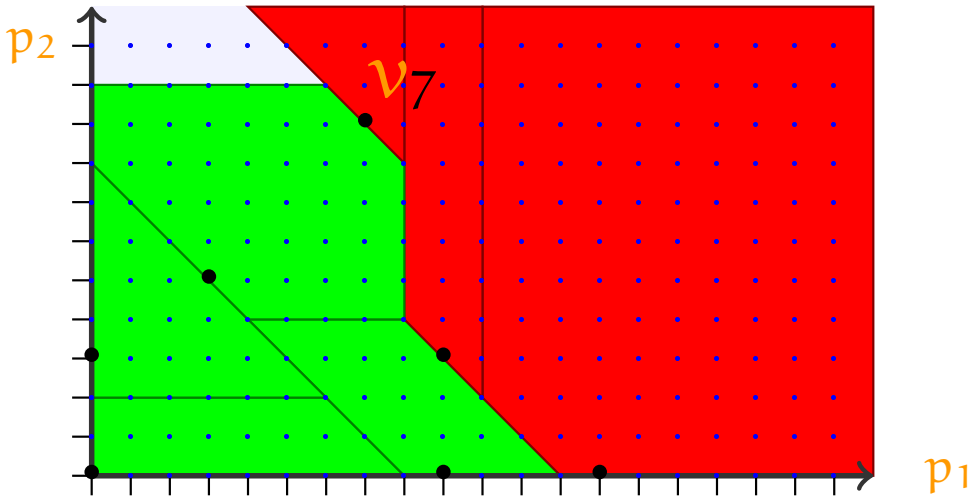
Parametric reachability preservation cartography (PRPC)



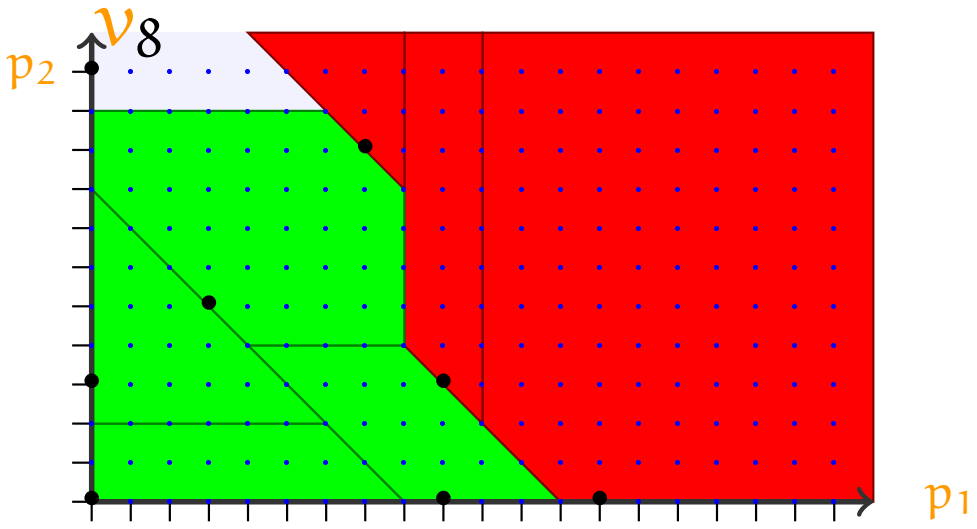
Parametric reachability preservation cartography (PRPC)



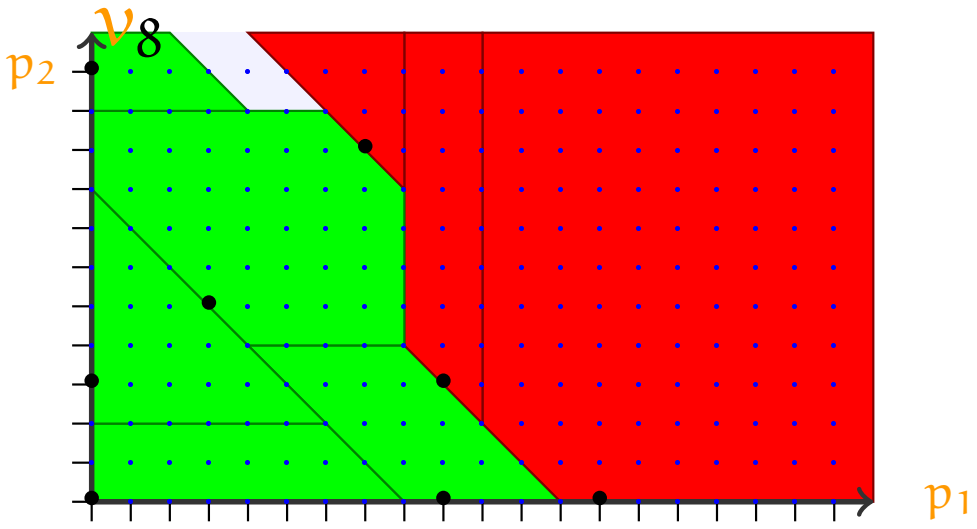
Parametric reachability preservation cartography (PRPC)



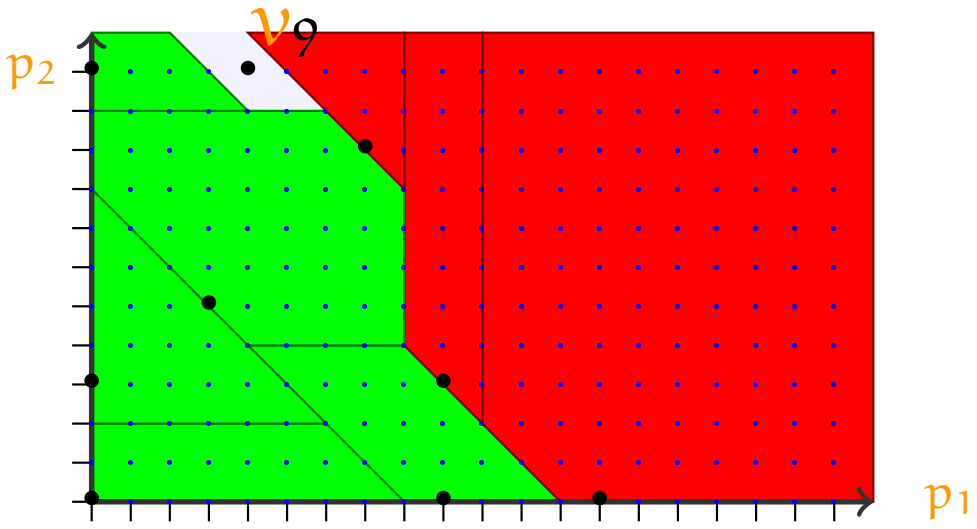
Parametric reachability preservation cartography (PRPC)



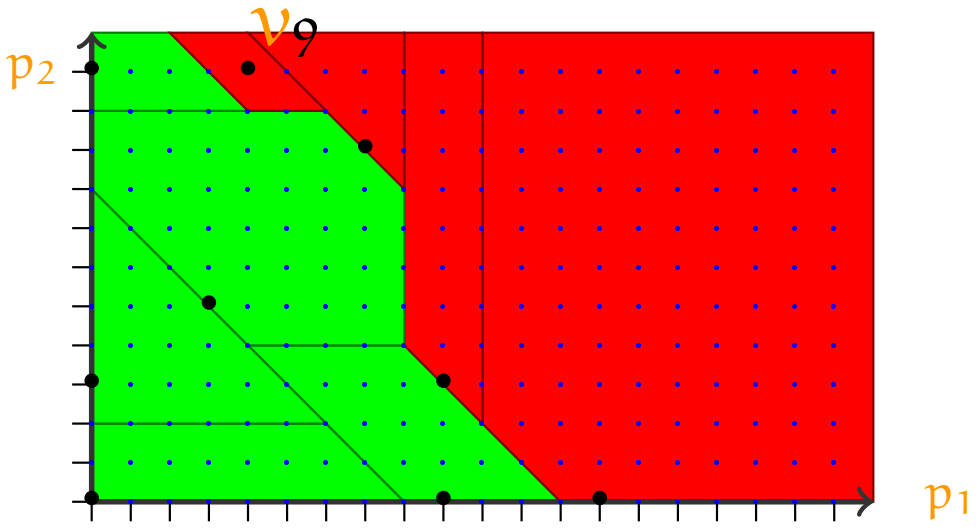
Parametric reachability preservation cartography (PRPC)



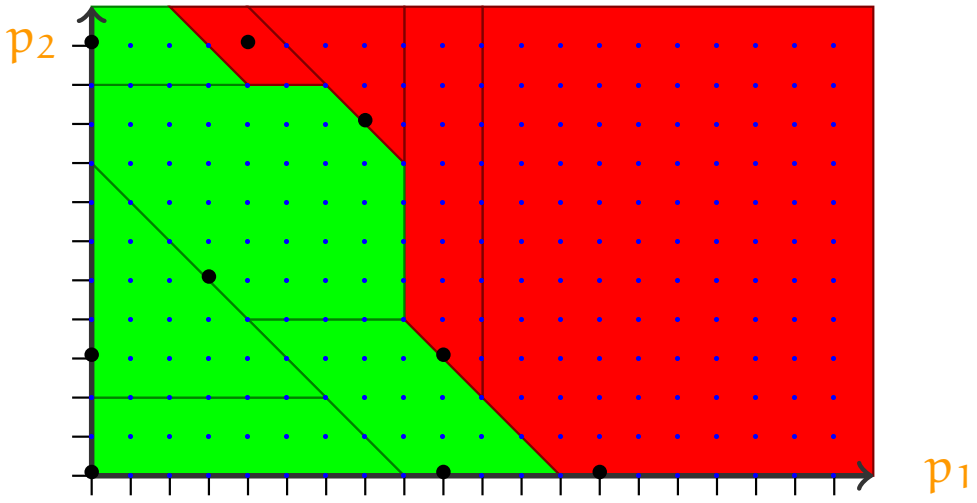
Parametric reachability preservation cartography (PRPC)



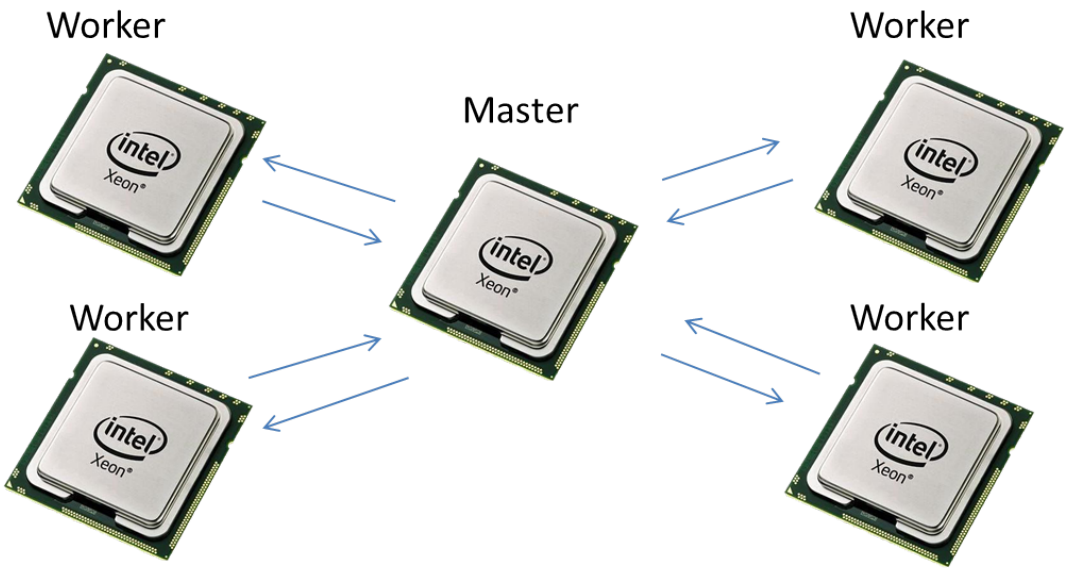
Parametric reachability preservation cartography (PRPC)



Parametric reachability preservation cartography (PRPC)



Master worker scheme



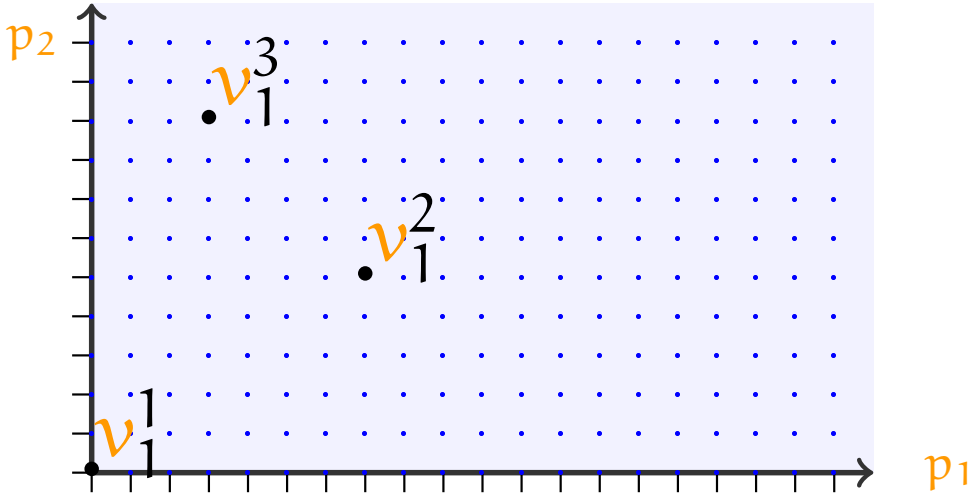
Master-worker distribution scheme:

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

Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

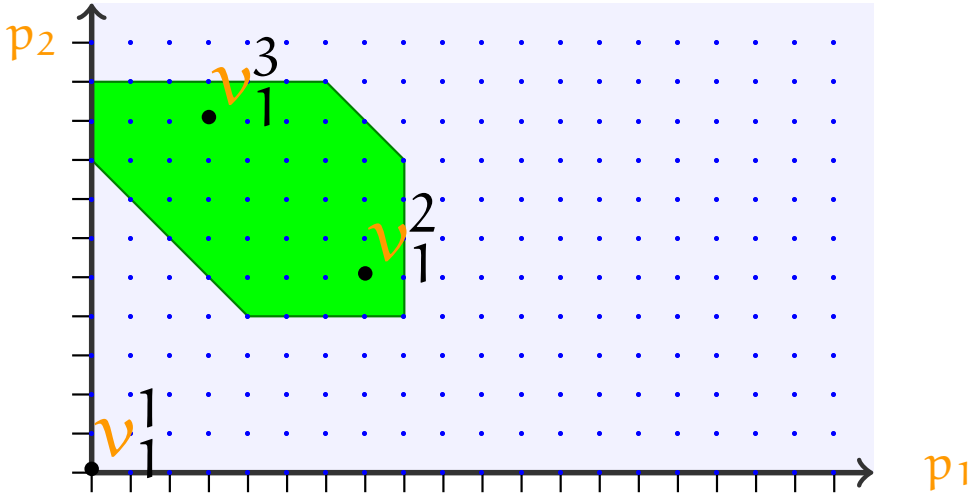
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

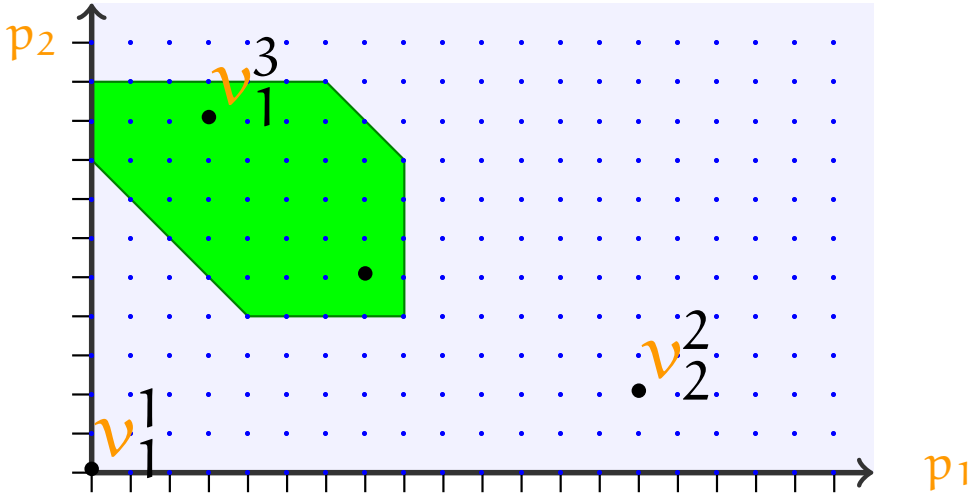
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

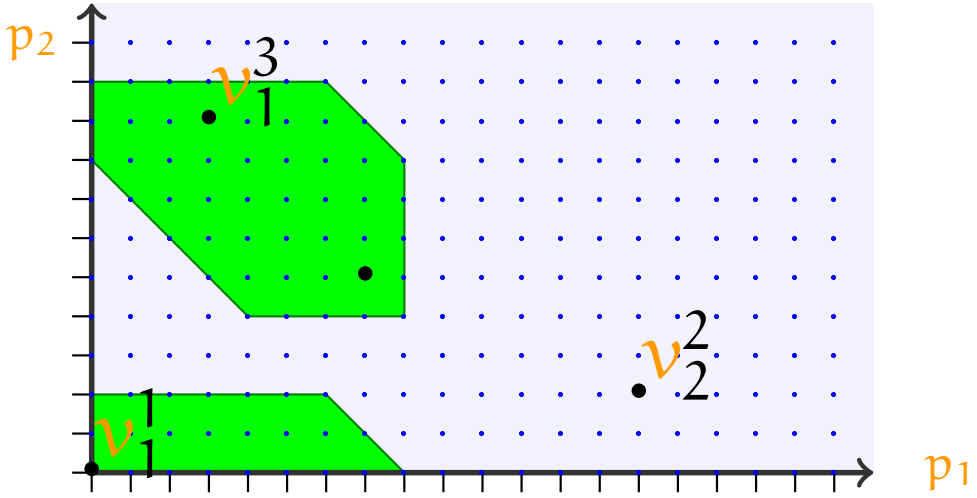
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

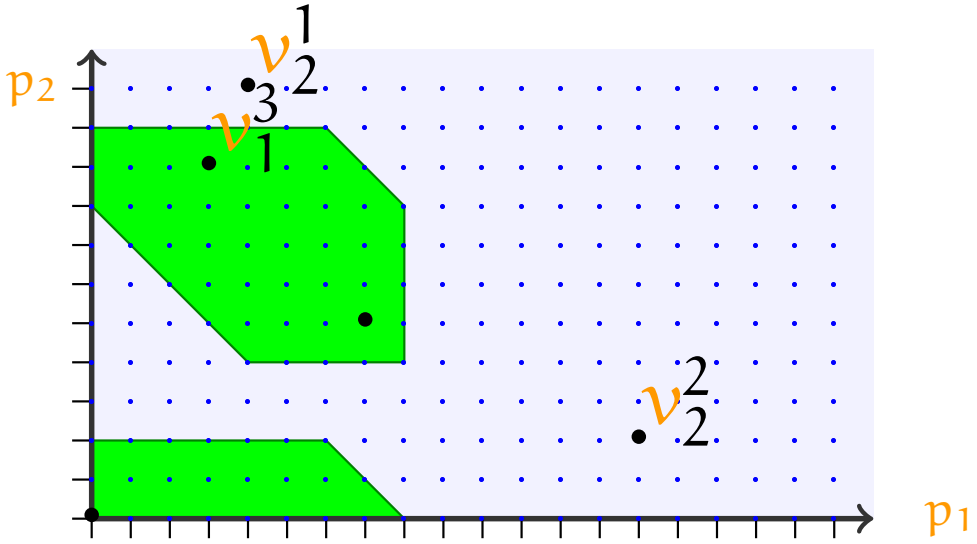
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

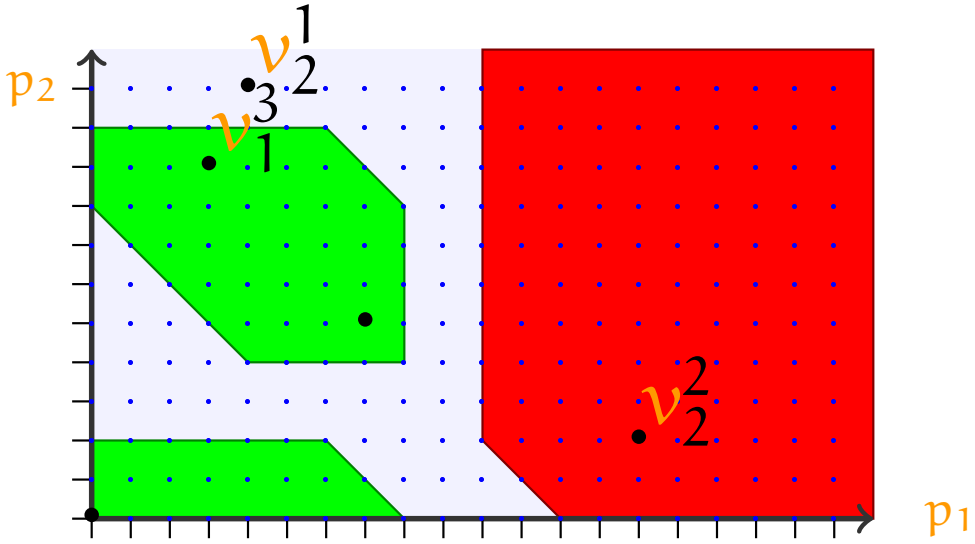
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

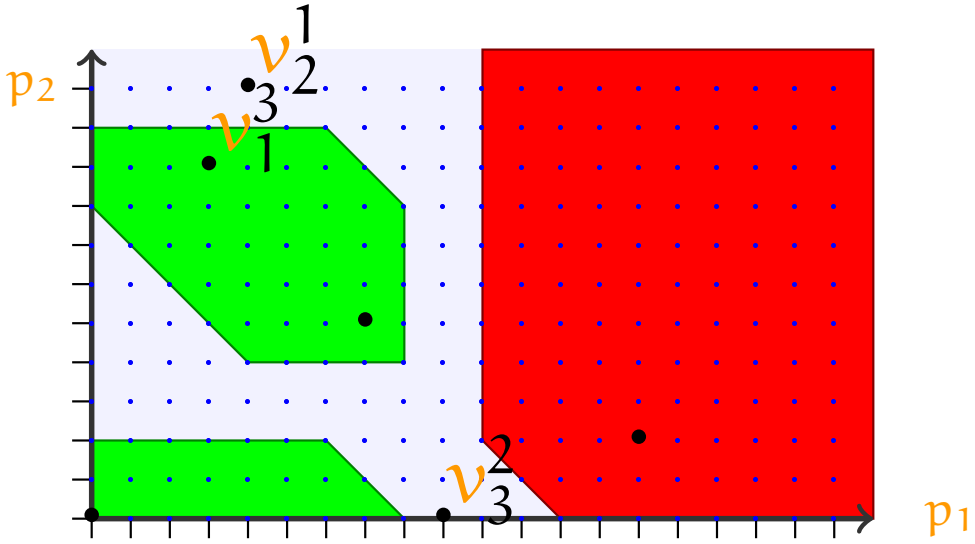
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

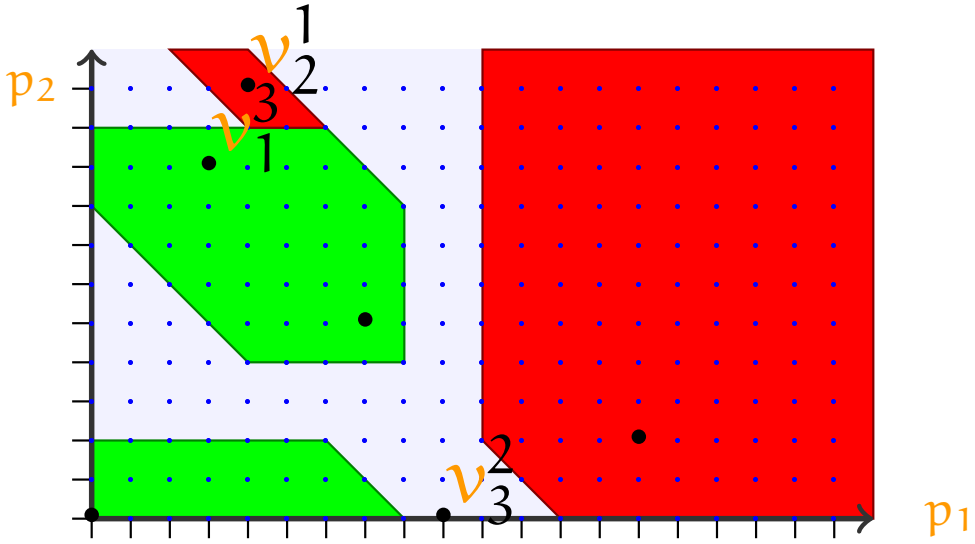
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

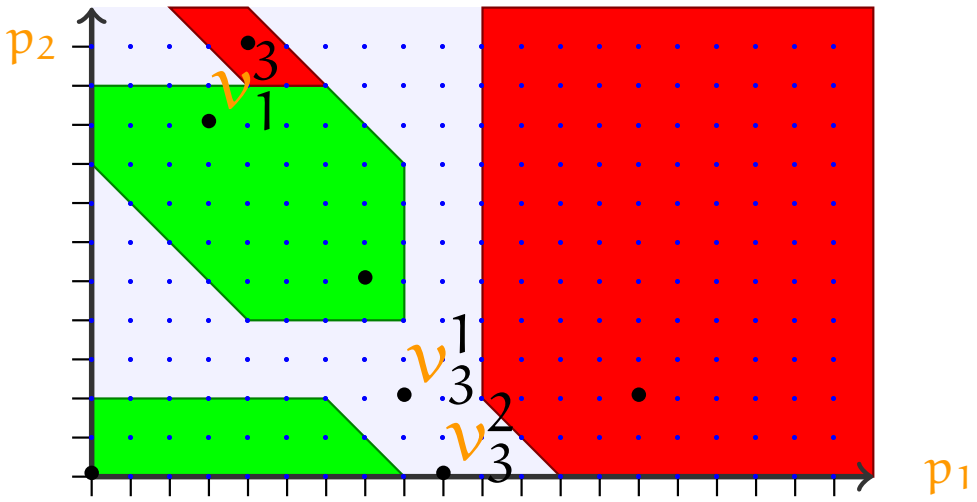
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

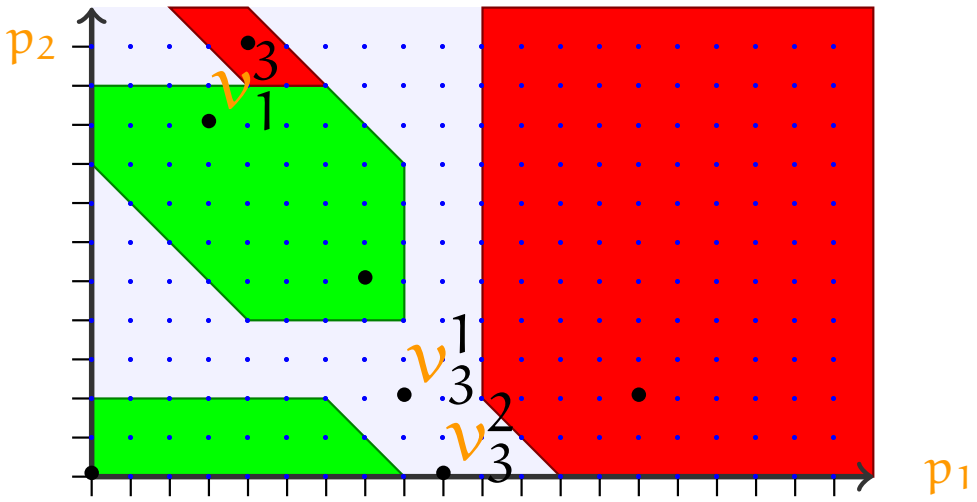
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

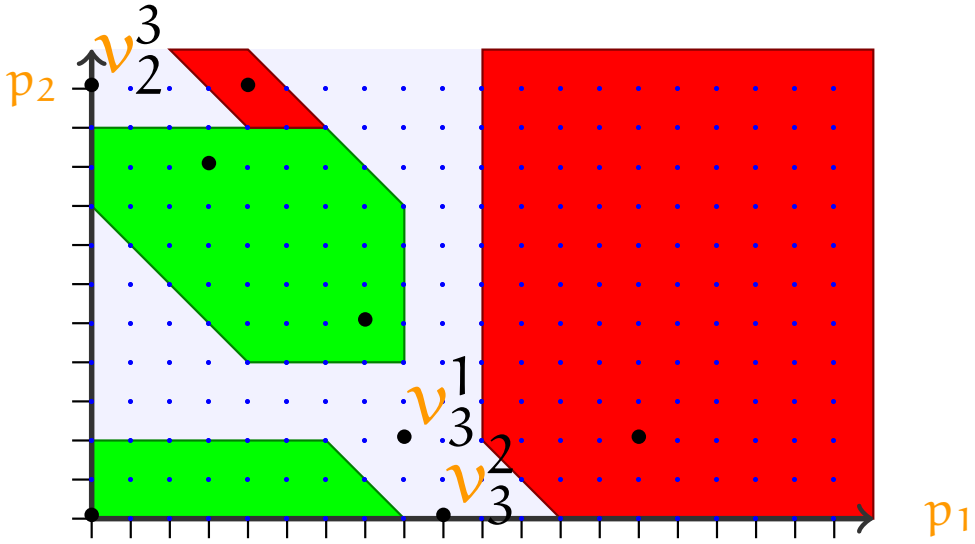
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

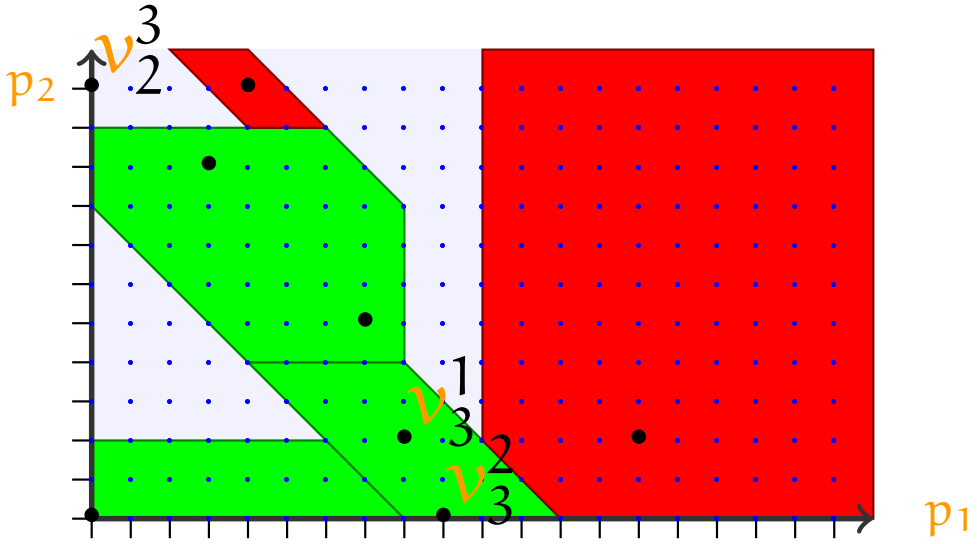
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

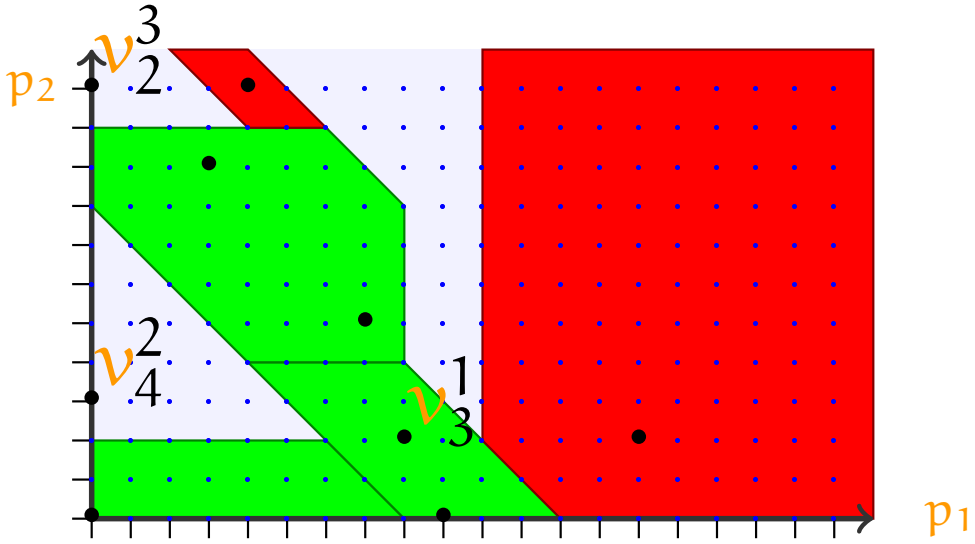
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

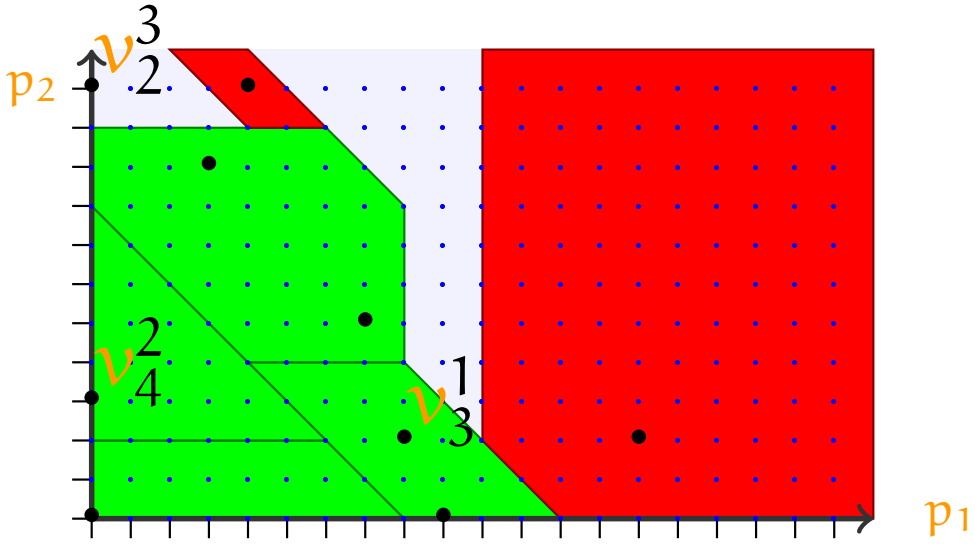
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

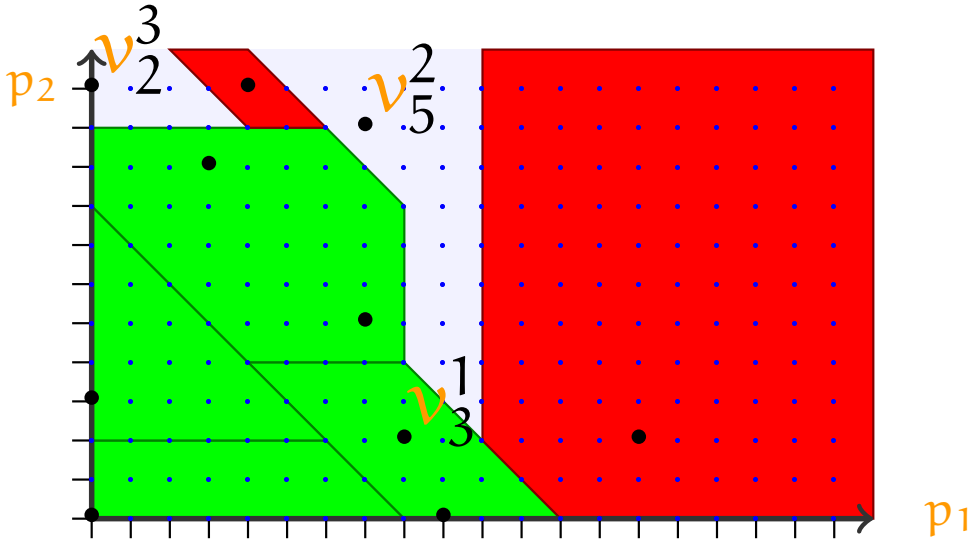
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

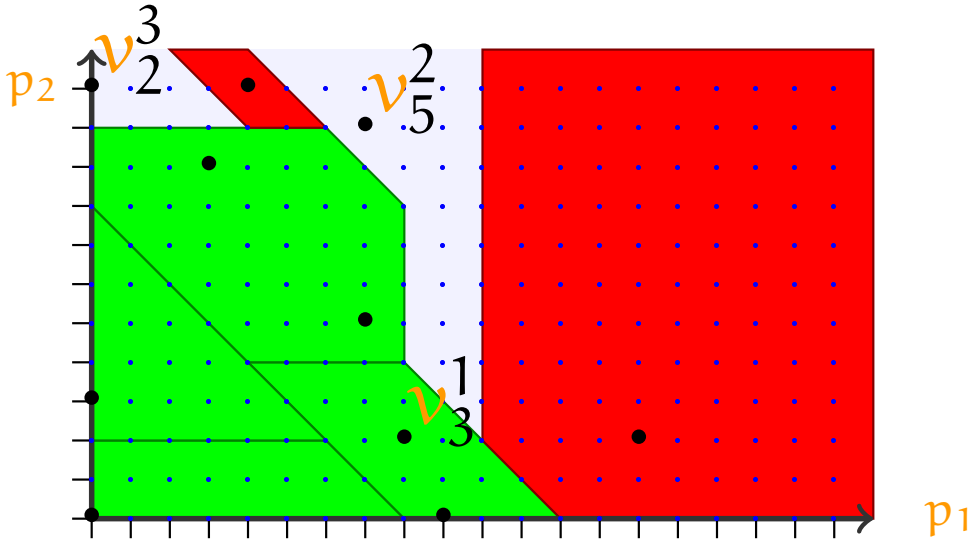
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

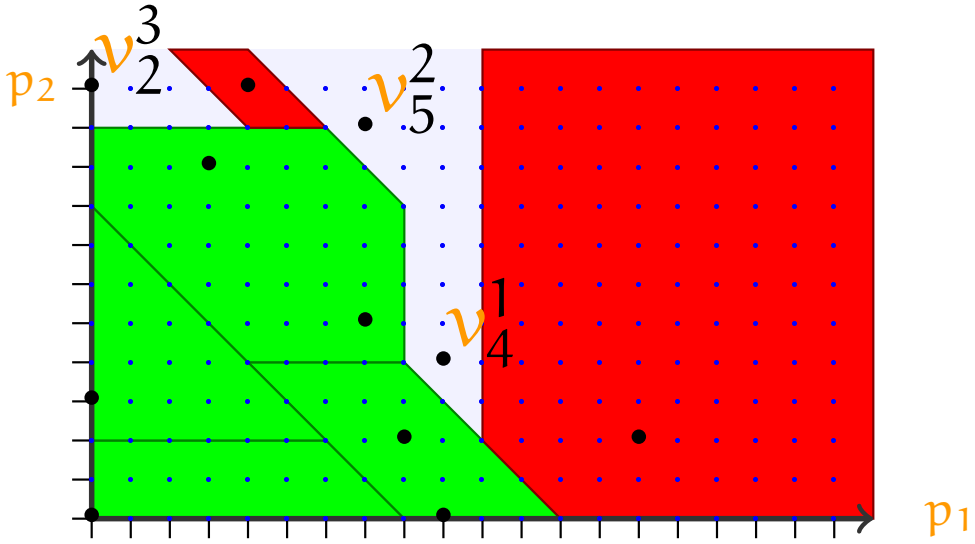
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

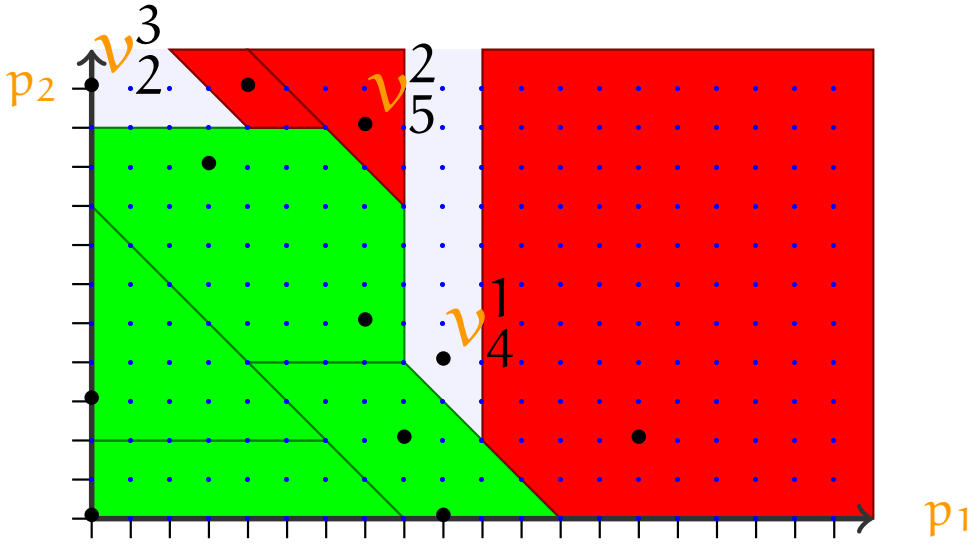
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

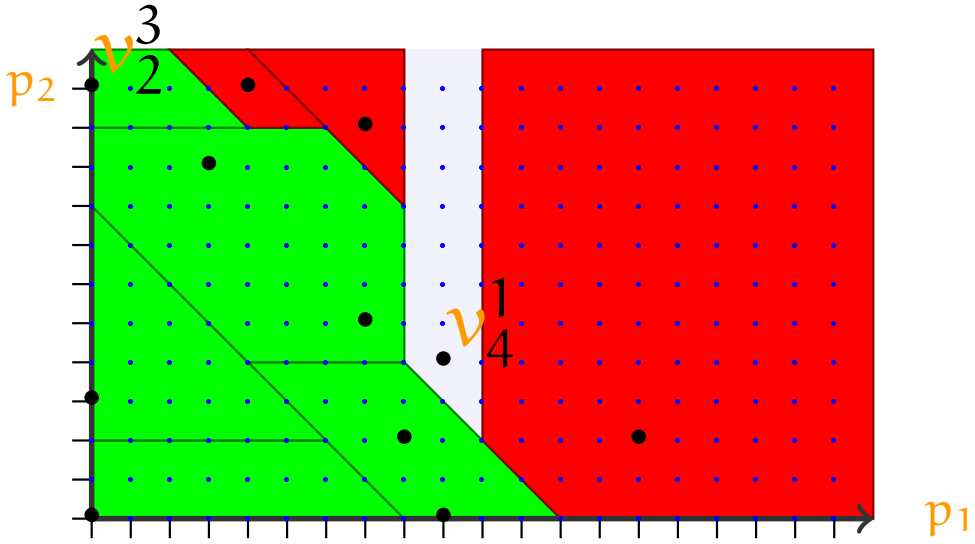
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

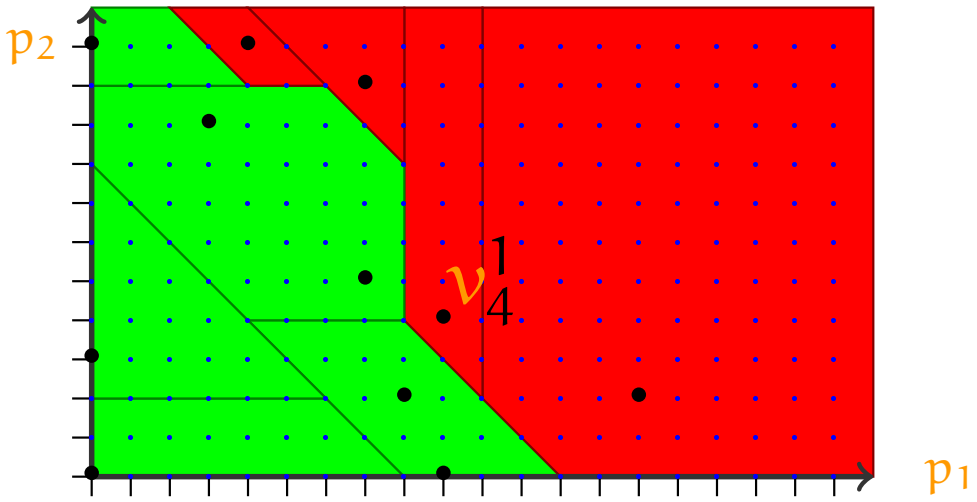
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

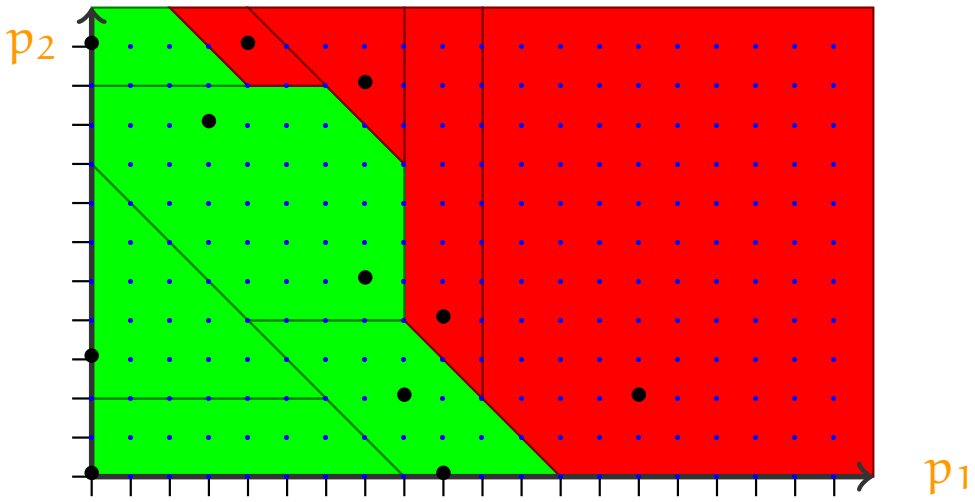
- 1 master
- 3 workers



Distributing PRPC: general scheme

Example with 2 parameters (p_1 and p_2) and 4 nodes:

- 1 master
- 3 workers



Problem:

- redundant computations

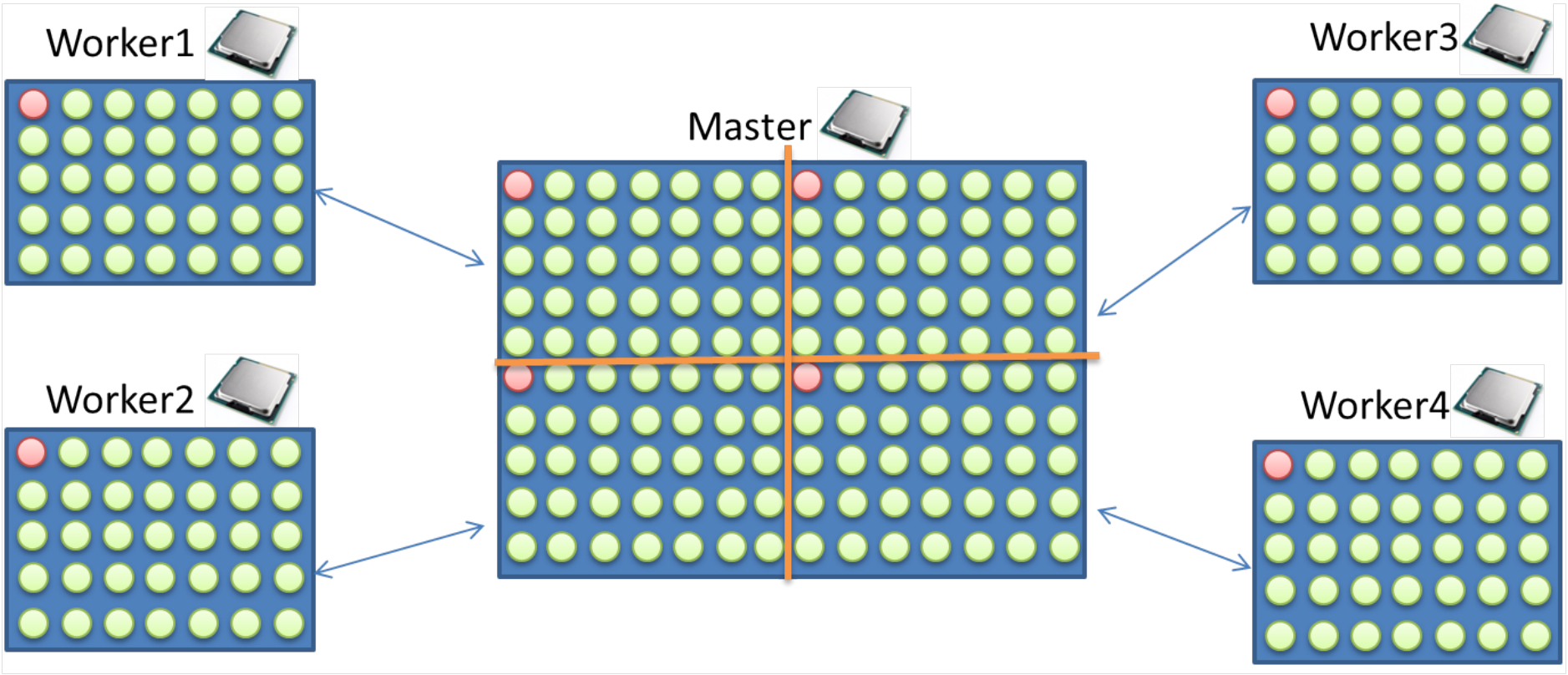
Dynamic domain decomposition

Most efficient distributed algorithm (so far!):

"Domain decomposition" scheme [A., Coti, Nguyen 2015]

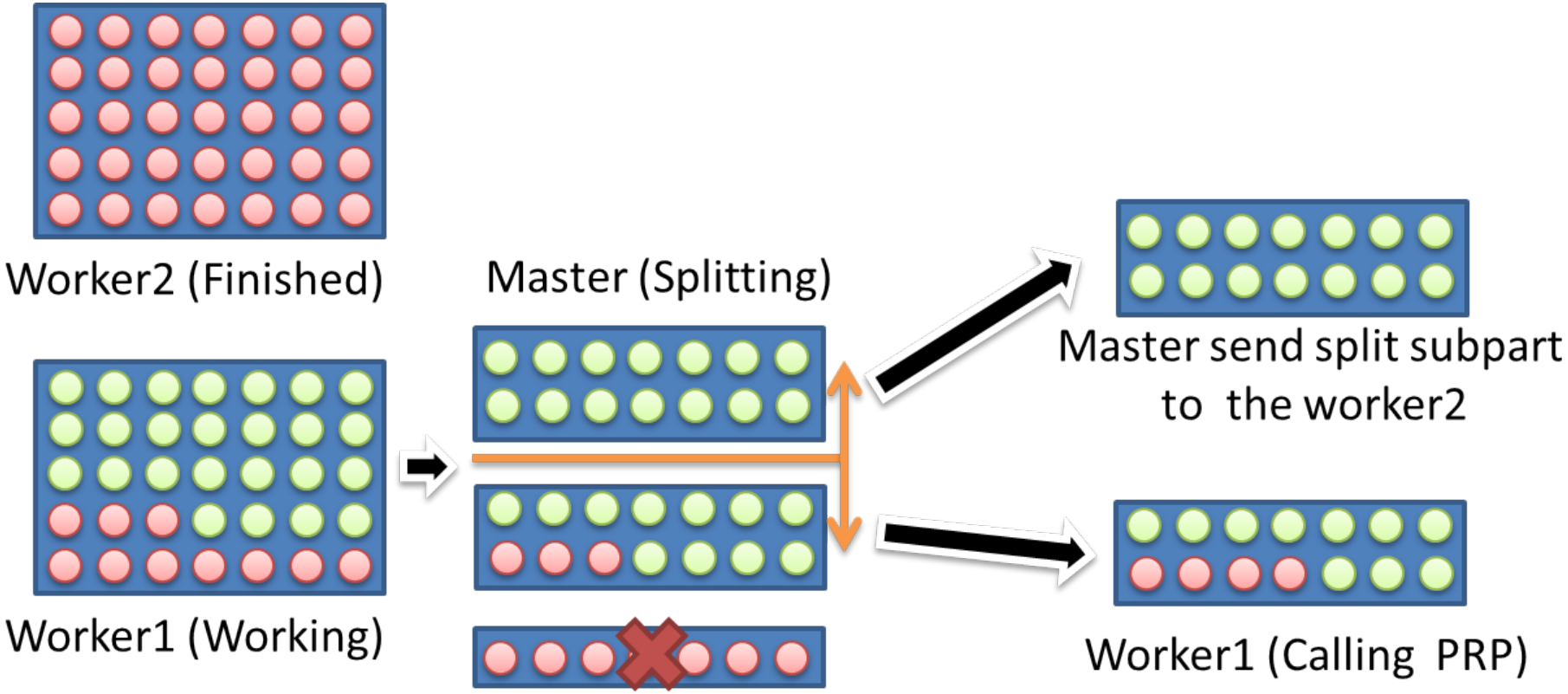
- Master
  1. initially splits the parameter domain into subdomains and send them to the workers
  2. when a worker has completed its subdomain, the master splits another subdomain, and sends it to the idle worker
- Workers
  1. receive the subdomain from the master
  2. call PRP on the points of this subdomain
  3. send the results (list of constraints) back to the master
  4. ask for more work

Domain decomposition: Initial splitting



- Prevent choosing close points
- Prevent bottleneck phenomenon at the master's side
  - Master only responsible for gathering constraints and splitting subdomains

Domain decomposition: Dynamic splitting



- Master can balance workload between workers

Implementation in IMITATOR

Implemented in IMITATOR using the MPI paradigm (message passing interface)

Distributed version up to 44 times faster using 128 nodes than the monolithic EFsynth [André et al., 2015a]

Outline

1 Parametric Timed Automata

2 Modeling and Verifying Real-Time Systems

3 Verifying a Real-time System under Uncertainty

4 Distributed Verification of Distributed Systems

5 Conclusion and Perspectives

Summary

- Parametric timed automata: a powerful (though undecidable) formalism to model and verify real-time systems
  - with preemption
  - with unknown or uncertain periods, jitters or WCETs

Perspectives

Address harder problems

- Thales challenge: what is the minimum time between two lost frames? (due to the uncertain periods)
  - Requires to model check thousands of frame processings

Improve the efficiency of parameter synthesis techniques

- Promising heuristics: approximations using the integer hull [Jovanović et al., 2015, André et al., 2015b]
- Distributed parameter synthesis
  - Multi-core synthesis [Laarman et al., 2013]
  - Distributed synthesis based on locations [Zhang et al., 2016]

And to finish...

Advertisement

Interested in the domain of verification with parameters?

We have post-doc positions in Nantes, Paris 7 and Paris 13!

Starting anytime.

Bibliography

References I

Adbeddaïm, Y. and Maler, O. (2002).
Preemptive job-shop scheduling using stopwatch automata.
In *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag.

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. (2015).
What's decidable about parametric timed automata?
In *FTSCS*, volume 596 of *Communications in Computer and Information Science*, pages 1–17. Springer.

André, É., Coti, C., and Nguyen, H. G. (2015a).
Enhanced distributed behavioral cartography of parametric timed automata.
In *ICFEM*, Lecture Notes in Computer Science. Springer.

References II

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., Lime, D., and Roux, O. H. (2015b).
Integer-complete synthesis for bounded parametric timed automata.
In *RP*, volume 9058 of *Lecture Notes in Computer Science*. 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.

Fanchon, L. and Jacquemard, F. (2013).
Formal timing analysis of mixed music scores.
In *ICMC 2013 (International Computer Music Conference)*.

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

References III

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.

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

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*, pages 968–983, Heidelberg, Germany. Springer.

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

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

References IV

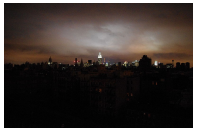
Sun, J., Liu, Y., Dong, J. S., and Pang, J. (2009).
PAT: Towards flexible verification under fairness.
In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer.

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

Zhang, Z., Nielsen, B., and Larsen, K. G. (2016).
Distributed algorithms for time optimal reachability analysis.
In *FORMATS 2016*, volume 9884 of *Lecture Notes in Computer Science*, pages 157–173. Springer.

Additional explanation

Explanation for the 4 pictures in the beginning



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

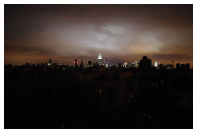
Error screen on the earliest versions of Macintosh

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

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

Licensing

Source of the graphics used I



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

Title: Sad mac
Author: Przemub
Source: https://commons.wikimedia.org/wiki/File:Sad_mac.png
License: Public domain

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

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

Source of the graphics used II

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

Title: Renault Twizy
Author: Citron
Source: https://commons.wikimedia.org/wiki/File:Renault_Twizy.jpg
License: CC BY-SA 3.0

Title: Airbus A380
Author: Axwel
Source: https://commons.wikimedia.org/wiki/File:Airbus_A380_blue_sky.jpg
License: CC BY 2.0

Source of the graphics used III

License of this document

Author: Étienne André