Petri Nets Tutorial, Parametric Verification (session 1)

Étienne André, Didier Lime, Wojciech Penczek, Laure Petrucci

Etienne.Andre@lipn.univ-paris13.fr — LIPN, Université Paris 13
Didier.Lime@ec-nantes.fr — IRCCyN, École Centrale de Nantes
penczek@ipipan.waw.pl — IPI-PAN, Warsaw
Laure.Petrucci@lipn.univ-paris13.fr — LIPN, Université Paris 13

June 21st, 2016

Thanks

Thanks for their support to...

project PACS ANR-14-CE28-0002
IPI-PAN, IRCCyN, LIPN

and of course...

All the developers of the tools

Outline

- General Introduction
  - Why parameters and of what kind?
  - Modelling languages: PN, PTA and their extensions.
  - Problems of interest.
- Parametric Timed Automata
  - Basic definitions and examples.
  - Decidability results.
  - EFSynth and IM algorithms.
  - Distributed algorithms.
  - IMITATOR in a nutshell.
- Parametric Interval Markov Chains
  - Basic definitions and examples.
  - Algorithm for Parameter Synthesis.
  - Detailed example.


ORTVS
AVRORA

General Introduction

First of all…

You know about automata and/or Petri nets:

- about their structure
- about their behaviour
- some analysis techniques

First of all…

You know about automata and/or Petri nets:

- about their structure
- about their behaviour
- some analysis techniques

Nice means to model and analyse concurrent systems…

…but …

- need for tuning the model
- need for parametrisation

First of all…

You know about automata and/or Petri nets:

- about their structure
- about their behaviour
- some analysis techniques

Nice means to model and analyse concurrent systems…

…but …

- need for tuning the model
- need for parametrisation

Let us have a deeper look into this now

Why parameters and of what kind?

- Why parameters?
  1. several copies of a same process or component, dimensioning, e.g.:
     - sensors in a wireless sensor network
  2. multiple *a priori* possible actions, e.g.:
     - modelling different design choices
  3. several hardware characteristics, e.g.:
     - different response time of electronic components

Why parameters and of what kind?

- Why parameters?
  1. several copies of a same process or component, dimensioning, e.g.:
     - sensors in a wireless sensor network
  2. multiple *a priori* possible actions, e.g.:
     - modelling different design choices
  3. several hardware characteristics, e.g.:
     - different response time of electronic components

- What kind of parameters?
  1. instances numbering
  2. enabled/disabled actions
  3. time or probabilities

Usual modelling languages are not sufficient:

- numbering possible with CPN, but fixed *a priori*
- no specific handling of (un)controllable actions
- timing included in TA or TPN, but also fixed

Problems of interest

- model parts of interest with parameters
- find some constraints on parameters guaranteeing desired properties
- find all parameter values guaranteeing these properties

Conclusion

At this stage:

- you have an idea on parametric modelling issues
  - instances
  - (un)controllable actions
  - time or probability constraints
- . . . and problems to address

Conclusion

At this stage:

- you have an idea on parametric modelling issues
  - instances
  - (un)controllable actions
  - time or probability constraints
- . . . and problems to address

Let us start with timing parameters (next sequence)


ORTVS

Parametric Timed Automata: Basic definitions and examples

First of all...

You have an idea on:

- parametric modelling issues
  - instances
  - (un)controllable actions
  - time or probability constraints
- problems to address

First of all...

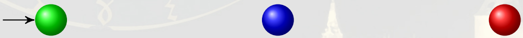
You have an idea on:

- parametric modelling issues
  - instances
  - (un)controllable actions
  - time or probability constraints
- problems to address

Let us introduce timing parameters now

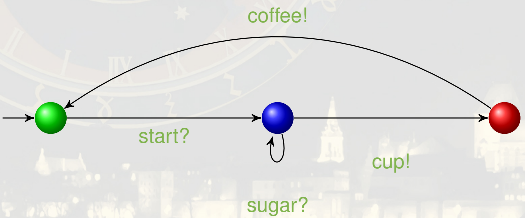
Timed automaton (TA)

- Finite state automaton (sets of locations)



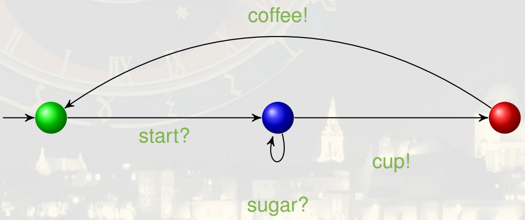
Timed automaton (TA)

- Finite state automaton (sets of locations and actions)



Timed automaton (TA)

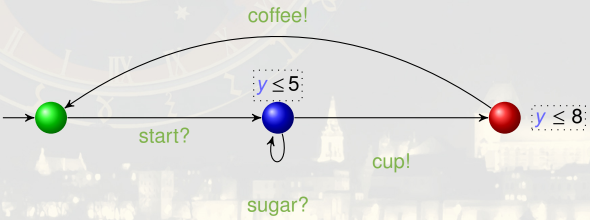
- Finite state automaton (sets of locations and actions) augmented with a set X of clocks Alur and Dill [1994]
  - Real-valued variables evolving linearly at the same rate



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

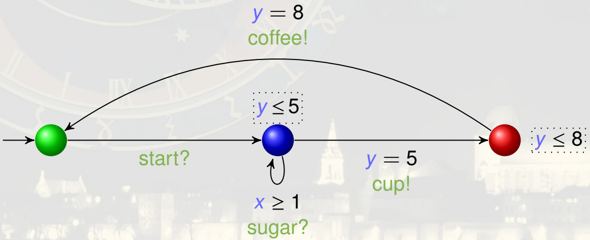
- Features
  - Location invariant: property to be verified to stay at a location



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

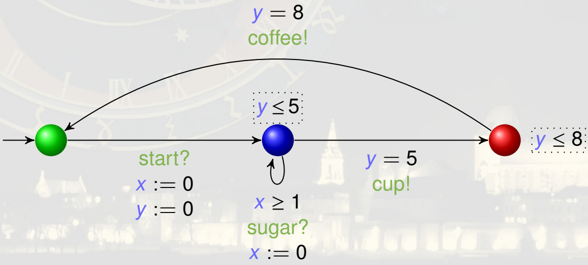
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition



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

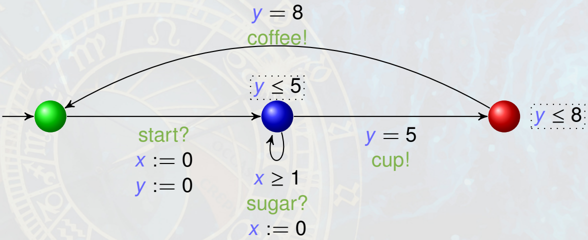
- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition



Concrete semantics of timed automata

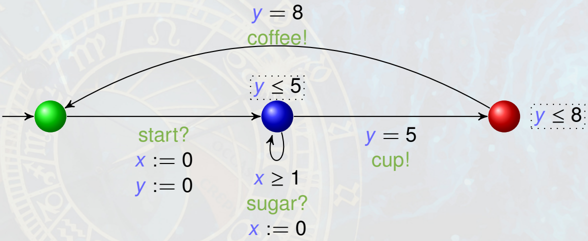
- Concrete state of a TA: pair (l, w), where
  - l is a location,
  - w is a valuation of each clock
- Concrete run: alternating sequence of concrete states and actions or time elapse

Examples of concrete runs

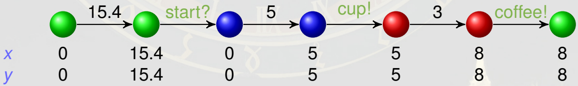


- Possible concrete runs for the coffee machine

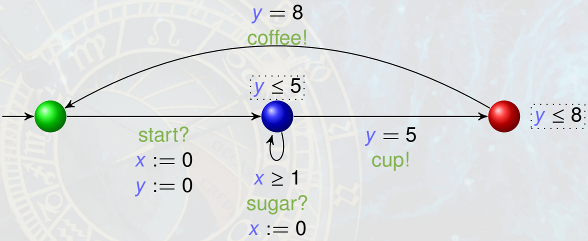
Examples of concrete runs



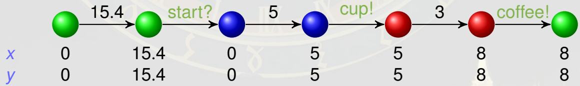
- Possible concrete runs for the coffee machine
  - Coffee with no sugar



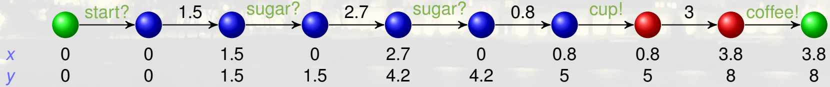
Examples of concrete runs



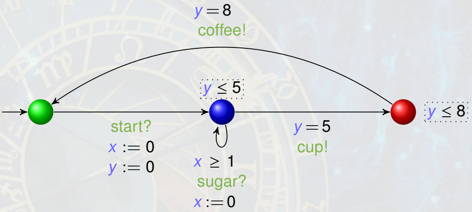
- Possible concrete runs for the coffee machine
  - Coffee with no sugar



  - Coffee with 2 doses of sugar



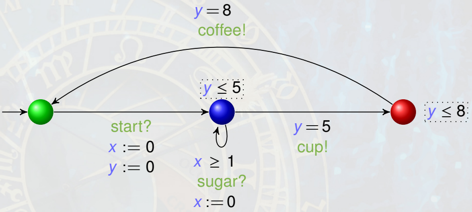
Verification of (timed) properties



Decide whether the following properties are satisfied for the timed coffee vending machine

> “Once the cup is delivered, coffee will come next within 2 seconds.”
>
> “It is possible to get a coffee with 5 doses of sugar.”
>
> “After the start button is pressed, a coffee is always eventually delivered.”
>
> “It is impossible to press the sugar button twice within 1 second.”

Verification of (timed) properties



Decide whether the following properties are satisfied for the timed coffee vending machine

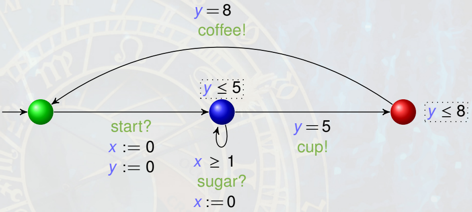
× "Once the cup is delivered, coffee will come next within 2 seconds."

"It is possible to get a coffee with 5 doses of sugar."

"After the start button is pressed, a coffee is always eventually delivered."

"It is impossible to press the sugar button twice within 1 second."

Verification of (timed) properties



Decide whether the following properties are satisfied for the timed coffee vending machine

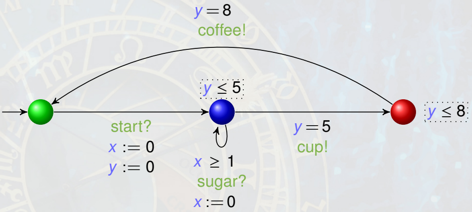
× "Once the cup is delivered, coffee will come next within 2 seconds."

√ "It is possible to get a coffee with 5 doses of sugar."

"After the start button is pressed, a coffee is always eventually delivered."

"It is impossible to press the sugar button twice within 1 second."

Verification of (timed) properties



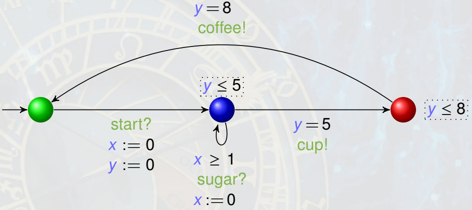
Decide whether the following properties are satisfied for the timed coffee vending machine

× "Once the cup is delivered, coffee will come next within 2 seconds."

√ "It is possible to get a coffee with 5 doses of sugar."

√ "After the start button is pressed, a coffee is always eventually delivered."

"It is impossible to press the sugar button twice within 1 second."



Decide whether the following properties are satisfied for the timed coffee vending machine

- × "Once the cup is delivered, coffee will come next within 2 seconds."
- √ "It is possible to get a coffee with 5 doses of sugar."
- √ "After the start button is pressed, a coffee is always eventually delivered."
- × "It is impossible to press the sugar button twice within 1 second."

Why timing parameters?

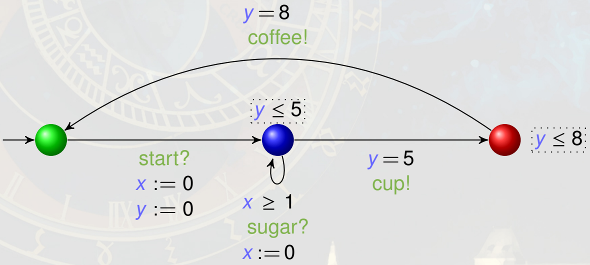
- Challenge 1: systems incompletely specified
  - Some delays may not be known yet, or may change
- Challenge 2: Robustness Markey [2011]
  - What happens if 8 is implemented with 7.99?
  - Can I really get a coffee with 5 doses of sugar?
- Challenge 3: Optimisation of timing constants
  - Up to which value of the delay between two actions sugar? can I still order a coffee with 3 doses of sugar?
- Challenge 4: Avoid numerous verifications
  - If one of the timing delays of the model changes, should I model check again the whole system?

Why timing parameters?

- Challenge 1: systems incompletely specified
  - Some delays may not be known yet, or may change
- Challenge 2: Robustness Markey [2011]
  - What happens if 8 is implemented with 7.99?
  - Can I really get a coffee with 5 doses of sugar?
- Challenge 3: Optimisation of timing constants
  - Up to which value of the delay between two actions sugar? can I still order a coffee with 3 doses of sugar?
- Challenge 4: Avoid numerous verifications
  - If one of the timing delays of the model changes, should I model check again the whole system?
- A solution: Parametric analysis
  - Consider that timing constants are unknown (parameters)
  - Find good values for the parameters s.t. the system behaves well

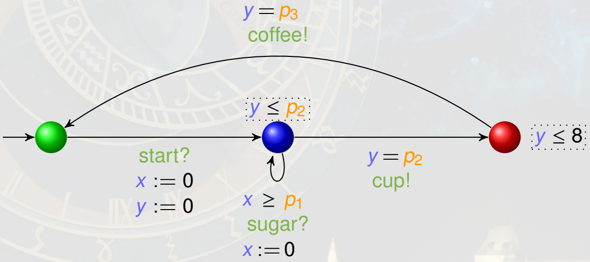
Parametric Timed Automaton (PTA)

- 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



Conclusion

At this stage:

- you have an idea on Parametric Timed Automata
- and the challenges for parametric analysis

Conclusion

At this stage:

- you have an idea on Parametric Timed Automata
- and the challenges for parametric analysis

Let us go for decidability results (next sequence)


ORTVS
AVRORA

Decidability results for Parametric Timed Automata

First of all. . .

You have an idea on:

- Parametric Timed Automata
- the challenges for parametric analysis

First of all...

You have an idea on:

- Parametric Timed Automata
- the challenges for parametric analysis

Let us now see some decidability results

What is decidability?

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

What is decidability?

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

Examples:

- “given three integers, is one of them the product of the other two?”
- “given a timed automaton, does there exist a run from the initial state to a given location l?”
- “given a context-free grammar, does it generate all strings?”
- “given a Turing machine, will it eventually halt?”

What is decidability?

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

Examples:

- √ "given three integers, is one of them the product of the other two?"
- "given a timed automaton, does there exist a run from the initial state to a given location l?"
- "given a context-free grammar, does it generate all strings?"
- "given a Turing machine, will it eventually halt?"

What is decidability?

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

Examples:

- √ "given three integers, is one of them the product of the other two?"
- √ "given a timed automaton, does there exist a run from the initial state to a given location l?"
- "given a context-free grammar, does it generate all strings?"
- "given a Turing machine, will it eventually halt?"

What is decidability?

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

Examples:

- √ “given three integers, is one of them the product of the other two?”
- √ “given a timed automaton, does there exist a run from the initial state to a given location l?”
- × “given a context-free grammar, does it generate all strings?”
- “given a Turing machine, will it eventually halt?”

What is decidability?

A decision problem is decidable if one can design an algorithm that, for any input of the problem, can answer yes or no (in a finite time, with a finite memory).

Examples:

- √ "given three integers, is one of them the product of the other two?"
- √ "given a timed automaton, does there exist a run from the initial state to a given location l?"
- × "given a context-free grammar, does it generate all strings?"
- × "given a Turing machine, will it eventually halt?"

Why studying decidability?

If a decision problem is undecidable, it is hopeless to look for algorithms yielding exact solutions for computation problems (because that is impossible)

Why studying decidability?

If a decision problem is undecidable, it is hopeless to look for algorithms yielding exact solutions for computation problems (because that is impossible)

However, one can:

- design semi-algorithms: if the algorithm halts, then its result is correct
- design algorithms yielding over- or under-approximations

Decision and computation problems for PTA

- EF-Emptiness "Does there exist a parameter valuation for which a given location l is reachable?"
  Example: "Does there exist at least one parameter valuation for which I can get a coffee with 2 sugars?"
- EF-Universality "Do all parameter valuations allow to reach a given location l?"
  Example: "Are all parameter valuations such that I may eventually get a coffee?"
- Preservation of the untimed language "Given a parameter valuation, does there exist another valuation with the same untimed language?"
  Example: "Given the valuation $p_1 = 1, p_2 = 5, p_3 = 8$, do there exist other valuations with the same possible untimed behaviours?"
- EF-Synthesis "Find all parameter valuations for which a given location l is reachable"
  Example: "What are all parameter valuations such that one may eventually get a coffee?"

Decision and computation problems for PTA

- EF-Emptiness "Does there exist a parameter valuation for which a given location l is reachable?"
  Example: "Does there exist at least one parameter valuation for which I can get a coffee with 2 sugars?" $\surd$, e.g. $p_1 = 1, p_2 = 5, p_3 = 8$
- EF-Universality "Do all parameter valuations allow to reach a given location l?"
  Example: "Are all parameter valuations such that I may eventually get a coffee?"
- Preservation of the untimed language "Given a parameter valuation, does there exist another valuation with the same untimed language?"
  Example: "Given the valuation $p_1 = 1, p_2 = 5, p_3 = 8$, do there exist other valuations with the same possible untimed behaviours?"
- EF-Synthesis "Find all parameter valuations for which a given location l is reachable"
  Example: "What are all parameter valuations such that one may eventually get a coffee?"

Decision and computation problems for PTA

- EF-Emptiness "Does there exist a parameter valuation for which a given location l is reachable?"
  Example: "Does there exist at least one parameter valuation for which I can get a coffee with 2 sugars?" $\surd$, e.g. $p_1 = 1, p_2 = 5, p_3 = 8$
- EF-Universality "Do all parameter valuations allow to reach a given location l?"
  Example: "Are all parameter valuations such that I may eventually get a coffee?" $\times$, e.g. $p_1 = 1, p_2 = 5, p_3 = 2$
- Preservation of the untimed language "Given a parameter valuation, does there exist another valuation with the same untimed language?"
  Example: "Given the valuation $p_1 = 1, p_2 = 5, p_3 = 8$, do there exist other valuations with the same possible untimed behaviours?"
- EF-Synthesis "Find all parameter valuations for which a given location l is reachable"
  Example: "What are all parameter valuations such that one may eventually get a coffee?"

Decision and computation problems for PTA

- EF-Emptiness "Does there exist a parameter valuation for which a given location l is reachable?"
  Example: "Does there exist at least one parameter valuation for which I can get a coffee with 2 sugars?" √, e.g. $p_1 = 1, p_2 = 5, p_3 = 8$
- EF-Universality "Do all parameter valuations allow to reach a given location l?"
  Example: "Are all parameter valuations such that I may eventually get a coffee?" ×, e.g. $p_1 = 1, p_2 = 5, p_3 = 2$
- Preservation of the untimed language "Given a parameter valuation, does there exist another valuation with the same untimed language?"
  Example: "Given the valuation $p_1 = 1, p_2 = 5, p_3 = 8$, do there exist other valuations with the same possible untimed behaviours?" √
- EF-Synthesis "Find all parameter valuations for which a given location l is reachable"
  Example: "What are all parameter valuations such that one may eventually get a coffee?"

Decision and computation problems for PTA

- EF-Emptiness "Does there exist a parameter valuation for which a given location l is reachable?"
  Example: "Does there exist at least one parameter valuation for which I can get a coffee with 2 sugars?" $\surd$, e.g. $p_1 = 1, p_2 = 5, p_3 = 8$
- EF-Universality "Do all parameter valuations allow to reach a given location l?"
  Example: "Are all parameter valuations such that I may eventually get a coffee?" $\times$, e.g. $p_1 = 1, p_2 = 5, p_3 = 2$
- Preservation of the untimed language "Given a parameter valuation, does there exist another valuation with the same untimed language?"
  Example: "Given the valuation $p_1 = 1, p_2 = 5, p_3 = 8$, do there exist other valuations with the same possible untimed behaviours?" $\surd$
- EF-Synthesis "Find all parameter valuations for which a given location l is reachable"
  Example: "What are all parameter valuations such that one may eventually get a coffee?" $0 \leq p_2 \leq p_3 \leq 8$

Decidability for PTA

- EF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is reachable?"
  undecidable Alur et al. [1993]; Beneš et al. [2015]

Decidability for PTA

- EF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is reachable?"
  undecidable — Alur et al. [1993]; Beneš et al. [2015]

- EF-universality problem
  "Do all parameter valuations allow to reach a given location l?"
  undecidable — André et al. [2016]

Decidability for PTA

- EF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is reachable?"
  undecidable Alur et al. [1993]; Beneš et al. [2015]

- EF-universality problem
  "Do all parameter valuations allow to reach a given location l?"
  undecidable André et al. [2016]

- Preservation of the untimed language
  "Given a parameter valuation, does there exist another valuations with the same untimed language?"
  undecidable André and Markey [2015]

Decidability for PTA

- EF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is reachable?"
  undecidable — Alur et al. [1993]; Beneš et al. [2015]

- EF-universality problem
  "Do all parameter valuations allow to reach a given location l?"
  undecidable — André et al. [2016]

- Preservation of the untimed language
  "Given a parameter valuation, does there exist another valuations with the same untimed language?"
  undecidable — André and Markey [2015]

In fact most interesting problems for PTAs are undecidable — André [2015]

Limiting the number of clocks

Undecidability is achieved for a single parameter Miller [2000]; Beneš et al. [2015]

However, reducing the number of clocks yields decidability of the EF-emptiness problem:

Limiting the number of clocks

Undecidability is achieved for a single parameter — Miller [2000]; Beneš et al. [2015]

However, reducing the number of clocks yields decidability of the EF-emptiness problem:

- √ 1 parametric clock and arbitrarily many non-parametric clocks and integer-valued parameters — Beneš et al. [2015]

Limiting the number of clocks

Undecidability is achieved for a single parameter Miller [2000]; Beneš et al. [2015]

However, reducing the number of clocks yields decidability of the EF-emptiness problem:

- √ 1 parametric clock and arbitrarily many non-parametric clocks and integer-valued parameters Beneš et al. [2015]
- √ 1 parametric clock and arbitrarily many rational-valued parameters Miller [2000]

Limiting the number of clocks

Undecidability is achieved for a single parameter Miller [2000]; Beneš et al. [2015]

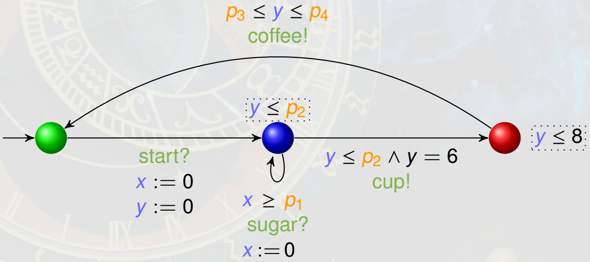
However, reducing the number of clocks yields decidability of the EF-emptiness problem:

- √ 1 parametric clock and arbitrarily many non-parametric clocks and integer-valued parameters Beneš et al. [2015]
- √ 1 parametric clock and arbitrarily many rational-valued parameters Miller [2000]
- √ 2 parametric clocks and 1 integer-valued parameter Bundala and Ouaknine [2014]

L/U-PTA

Definition

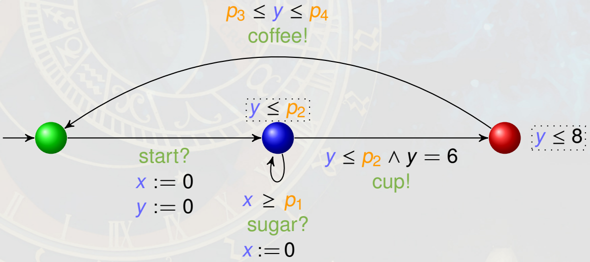
A lower/upper bound PTA (L/U-PTA) is a PTA in which each parameter p is always compared with clocks as an upper bound or always as a lower bound.



Lower-bound parameters:
Upped-bound parameters:

Definition

A lower/upper bound PTA (L/U-PTA) is a PTA in which each parameter p is always compared with clocks as an upper bound or always as a lower bound.



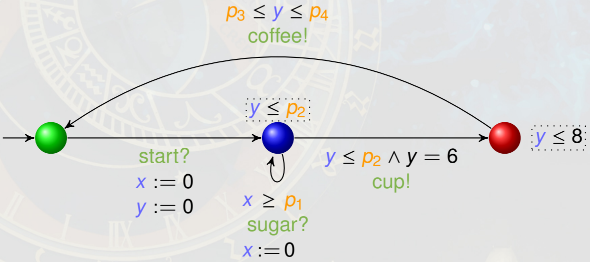
Lower-bound parameters: p_1, p_3

Upped-bound parameters:

L/U-PTA

Definition

A lower/upper bound PTA (L/U-PTA) is a PTA in which each parameter p is always compared with clocks as an upper bound or always as a lower bound.



Lower-bound parameters: p_1, p_3

Upped-bound parameters: p_2, p_4

Decidable problems for L/U-PTA

- EF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is reachable?"
  decidable — Hune et al. [2002]

Decidable problems for L/U-PTA

- EF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is reachable?"
  decidable — Hune et al. [2002]

- EF-universality problem
  "Do all parameter valuations allow to reach a given location l?"
  decidable — Bozzelli and La Torre [2009]

Decidable problems for L/U-PTA

- EF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is reachable?"
  decidable — Hune et al. [2002]

- EF-universality problem
  "Do all parameter valuations allow to reach a given location l?"
  decidable — Bozzelli and La Torre [2009]

- EF-finiteness problem
  "Is the set of parameter valuations allowing to reach a given location l finite?"
  decidable (for integer valuations) — Bozzelli and La Torre [2009]

Undecidable problems for L/U-PTA

- AF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is always eventually reachable?"
  undecidable — Jovanović et al. [2015]

Undecidable problems for L/U-PTA

- AF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is always eventually reachable?"
  undecidable — Jovanović et al. [2015]

- AF-universality problem
  "Are all valuations such that a given location l is always eventually reachable?"
  undecidable (but...) — André and Lime [2016]

Undecidable problems for L/U-PTA

- AF-emptiness problem
  "Does there exist a parameter valuation for which a given location l is always eventually reachable?"
  undecidable — Jovanović et al. [2015]

- AF-universality problem
  "Are all valuations such that a given location l is always eventually reachable?"
  undecidable (but...) — André and Lime [2016]

- language preservation emptiness problem
  "Given a parameter valuation v, can we find another valuation with the same untimed language?"
  undecidable — André and Markey [2015]

What can we do with L/U-PTA?

In an L/U PTA, can we syntactically...

- use an equality $(=)$ in a guard or invariant?
- use an equality $x = p$ in a guard or invariant?

What can we do with L/U-PTA?

In an L/U PTA, can we syntactically...

- use an equality ($=$) in a guard or invariant?
  yes (without parameters!)

- use an equality $x = p$ in a guard or invariant?

What can we do with L/U-PTA?

In an L/U PTA, can we syntactically. . .

- use an equality ($=$) in a guard or invariant?
  yes (without parameters!)

- use an equality $x = p$ in a guard or invariant?
  no!

What fits into the class of L/U-PTA?

- Any model with parametric delays given in the form of intervals
  - E.g.: $[p_{min}, p_{max}]$
- Many communication protocols
- All hardware circuits modeled using a bi-bounded inertial delay model

Conclusion

Most interesting problems are undecidable for PTA

…but some become decidable when bounding the number of clocks, or adding restrictions on the use of parameters (L/U-PTA)

Conclusion

Most interesting problems are undecidable for PTA

...but some become decidable when bounding the number of clocks, or adding restrictions on the use of parameters (L/U-PTA)

Let us go for some parameter synthesis algorithms (next sequence)


ORTVS

Parameter synthesis algorithms

First of all…

You know that:

- most problems are undecidable for Parametric Timed Automata
- but some are decidable on specific classes

First of all…

You know that:

- most problems are undecidable for Parametric Timed Automata
- but some are decidable on specific classes

Let us now see some parameter synthesis algorithms

Symbolic states for timed automata

- Objective: group all concrete states reachable by the same sequence of discrete actions
- Symbolic state: a location l and a (infinite) set of states Z
- For timed automata, Z can be represented by a convex polyhedron with a special form called zone, with constraints

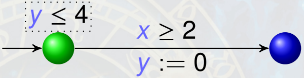
$$-d_{0i} \leq x_i \leq d_{i0} \text{ and } x_i - x_j \leq d_{ij}$$

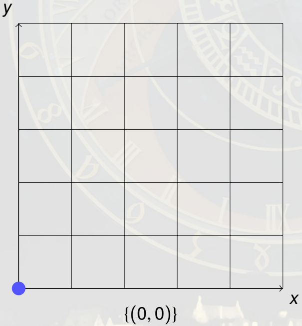
- Computation of successive reachable symbolic states can be performed symbolically with polyhedral operations: for edge $e = (l, a, g, R, l')$:

$$\text{Succ}((l, Z), e) = (l', (Z \cap g)[R] \cap Inv(l'))^{\nearrow} \cap Inv(l'))$$

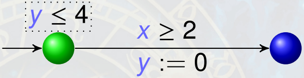
- With an additional technicality there is a finite number of reachable zones in a TA.

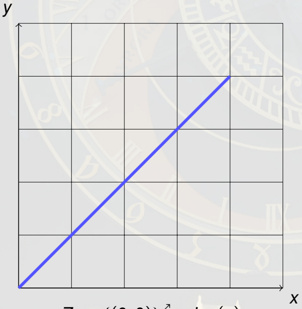
Symbolic states for timed automata: Example





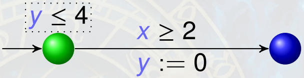
Symbolic states for timed automata: Example

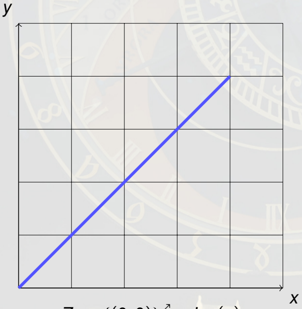




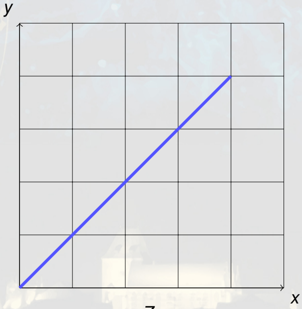
$Z_0 = \{(0,0)\}^{\nearrow} \cap Inv(\bullet)$

Symbolic states for timed automata: Example



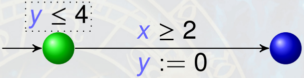


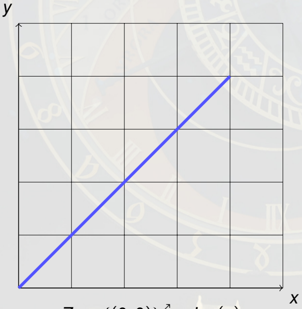
$Z_0 = \{(0,0)\}^{\nearrow} \cap Inv(\bullet)$



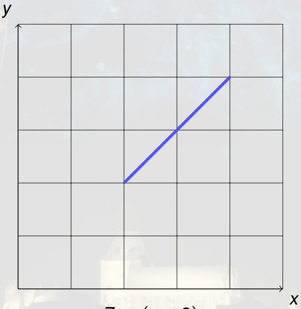
Z_0

Symbolic states for timed automata: Example



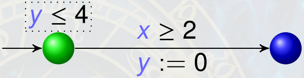


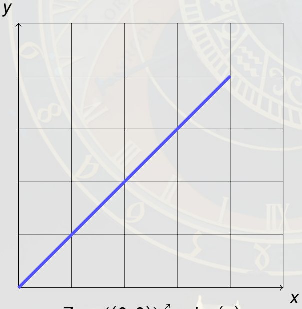
$Z_0 = \{(0,0)\}^{\nearrow} \cap Inv(\bullet)$



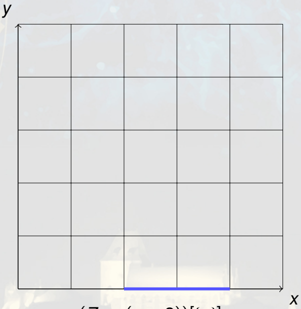
$Z_0 \cap (x \geq 2)$

Symbolic states for timed automata: Example



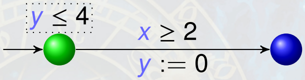


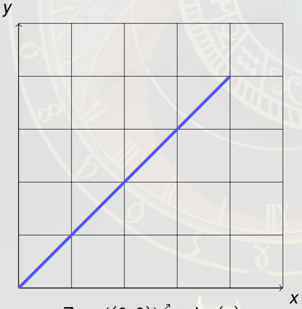
$Z_0 = \{(0,0)\}^{\nearrow} \cap Inv(\bullet)$



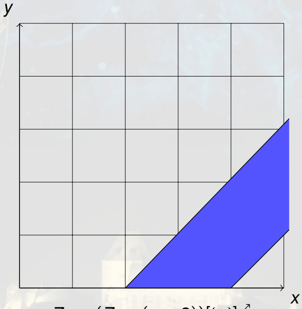
$(Z_0 \cap (x \geq 2))[\{y\}]$

Symbolic states for timed automata: Example





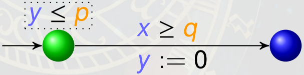
$Z_0 = \{(0,0)\}^{\nearrow} \cap Inv(\bullet)$



$Z_1 = (Z_0 \cap (x \geq 2))[\{y\}]^{\nearrow}$

Symbolic states for parametric TA

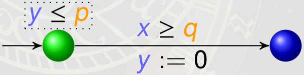
- Symbolic state (l, Z): location + convex polyhedron constraining both clocks and parameters;
- Straightforward extension of reset and future that act only on the clock variables;
- Convex polyhedra obtained have a special form called parametric zone Hune et al. [2002].



$$Z_0 = \begin{cases} x = y \\ 0 \leq y \leq p \\ p, q \geq 0 \end{cases} \qquad Z_1 = \begin{cases} q \leq x - y \leq p \\ (q \leq p) \\ x, y, p, q \geq 0 \end{cases}$$

Symbolic states for parametric TA

- Symbolic state (l, Z): location + convex polyhedron constraining both clocks and parameters;
- Straightforward extension of reset and future that act only on the clock variables;
- Convex polyhedra obtained have a special form called parametric zone Hune et al. [2002].



$$Z_0 = \begin{cases} x = y \\ 0 \leq y \leq p \\ p, q \geq 0 \end{cases} \qquad Z_1 = \begin{cases} q \leq x - y \leq p \\ (q \leq p) \\ x, y, p, q \geq 0 \end{cases}$$

- There exists in general an infinite number of such symbolic states in a PTA

A semi-algorithm for parametric reachability

$$\mathrm{EF}_G(S, M) = \begin{cases} Z\downarrow_P & \text{if } l \in G \\ \emptyset & \text{if } S \in M \\ \bigcup_{\substack{e \in E \\ S' = \mathrm{Succ}(S,e)}} \mathrm{EF}_G\big(S', M \cup \{S\}\big) & \text{otherwise.} \end{cases}$$

- $S = (l, Z)$;
- G a set of locations to reach;
- M is a list of visited symbolic states;
- $\mathrm{Succ}(S, e)$ computes the symbolic successor of S by edge e;
- EF collects the parametric reachability condition of all symbolic states with a goal location; Jovanović et al. [2015]
- correctness and completeness guaranteed if the algorithm terminates, but...

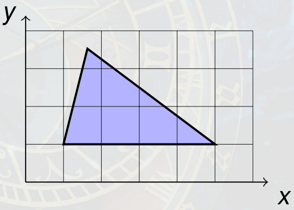
A semi-algorithm for parametric reachability

$$\mathrm{EF}_G(S, M) = \begin{cases} Z{\downarrow}_P & \text{if } l \in G \\ \emptyset & \text{if } S \in M \\ \bigcup_{\substack{e \in E \\ S' = \mathrm{Succ}(S,e)}} \mathrm{EF}_G\big(S', M \cup \{S\}\big) & \text{otherwise.} \end{cases}$$

- $S = (l, Z)$;
- G a set of locations to reach;
- M is a list of visited symbolic states;
- $\mathrm{Succ}(S, e)$ computes the symbolic successor of S by edge e;
- EF collects the parametric reachability condition of all symbolic states with a goal location; Jovanović et al. [2015]
- correctness and completeness guaranteed if the algorithm terminates, but…
- termination is not guaranteed (because the underlying problem is undecidable)

Beyond EFSynth

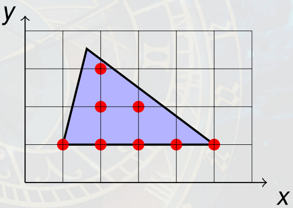
- EFSynth is the most basic synthesis semi-algorithm for PTA;
- Termination can be ensured, using the notion of integer hull Jovanović et al. [2015]; André et al. [2015b]:



  - at the cost of completeness;
  - for bounded parameters;
  - but preserves all integer points.
- Similar (semi-)algorithms are also available for more complex properties (e.g. invevitability Jovanović et al. [2015]);
- EFSynth is implemented in IMITATOR and Roméo.

Beyond EFSynth

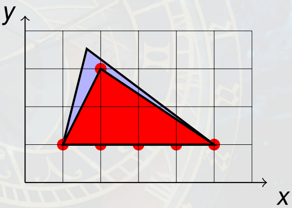
- EFSynth is the most basic synthesis semi-algorithm for PTA;
- Termination can be ensured, using the notion of integer hull Jovanović et al. [2015]; André et al. [2015b]:



  - at the cost of completeness;
  - for bounded parameters;
  - but preserves all integer points.
- Similar (semi-)algorithms are also available for more complex properties (e.g. invevitability Jovanović et al. [2015]);
- EFSynth is implemented in IMITATOR and Roméo.

Beyond EFSynth

- EFSynth is the most basic synthesis semi-algorithm for PTA;
- Termination can be ensured, using the notion of integer hull Jovanović et al. [2015]; André et al. [2015b]:



  - at the cost of completeness;
  - for bounded parameters;
  - but preserves all integer points.
- Similar (semi-)algorithms are also available for more complex properties (e.g. invevitability Jovanović et al. [2015]);
- EFSynth is implemented in IMITATOR and Roméo.

TPsynth: preserving the untimed behaviour

The trace preservation problem

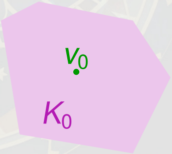
Given a PTA A and a parameter valuation v_0, synthesize other valuations yielding the same time-abstract behaviour (trace set). André et al. [2009]; André and Markey [2015]

v_0

TPsynth: preserving the untimed behaviour

The trace preservation problem

Given a PTA A and a parameter valuation v_0, synthesize other valuations yielding the same time-abstract behaviour (trace set). André et al. [2009]; André and Markey [2015]



TPsynth ("inverse method"): Simplified algorithm

Two parts:

1. Forbid all v_0-incompatible behaviours
2. Require all v_0-compatible behaviours

Algorithm TPsynth(A, v_0):
Start with $K_0 = \texttt{true}$

REPEAT

1. Compute a set S of reachable symbolic states under K_0
2. Refine K_0 by removing a v_0-incompatible state from S
   - Select a v_0-incompatible state (l, C) within S (i.e. $v_0 \not\models C$)
   - Add $\neg C{\downarrow_P}$ to K_0

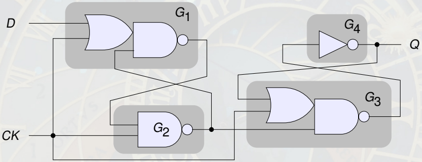
UNTIL no more v_0-incompatible state in S

RETURN the intersection of all states

An example of flip-flop circuit

- An asynchronous circuit

Clarisó and Cortadella [2007]

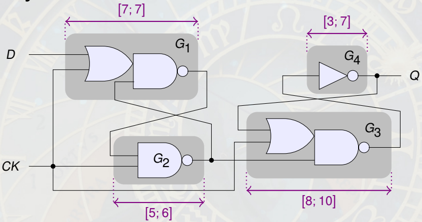


- Concurrent behaviour
  - 4 elements: G_1, G_2, G_3, G_4
  - 2 input signals (D and CK), 1 output signal (Q)

An example of flip-flop circuit

- An asynchronous circuit

Clarisó and Cortadella [2007]

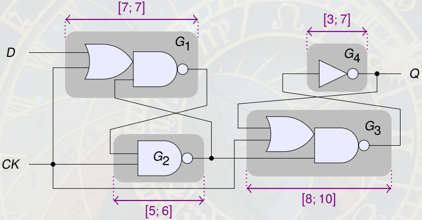


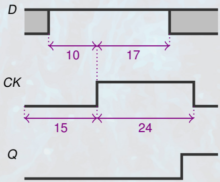
- Concurrent behaviour
  - 4 elements: G_1, G_2, G_3, G_4
  - 2 input signals (D and CK), 1 output signal (Q)
- Timing delays
  - Traversal delays of the gates: one interval per gate

An example of flip-flop circuit

- An asynchronous circuit

Clarisó and Cortadella [2007]



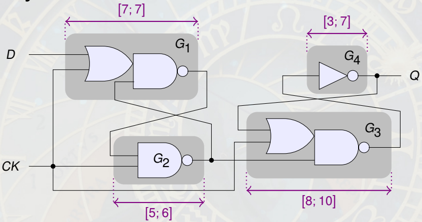


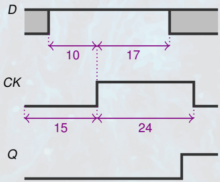
- Concurrent behaviour
  - 4 elements: G_1, G_2, G_3, G_4
  - 2 input signals (D and CK), 1 output signal (Q)
- Timing delays
  - Traversal delays of the gates: one interval per gate
  - Environment timing constants

An example of flip-flop circuit

- An asynchronous circuit

Clarisó and Cortadella [2007]





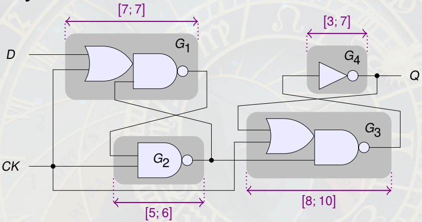
- Concurrent behaviour
  - 4 elements: G_1, G_2, G_3, G_4
  - 2 input signals (D and CK), 1 output signal (Q)
- Timing delays
  - Traversal delays of the gates: one interval per gate
  - Environment timing constants

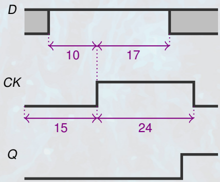
- Question
  - For these timing delays, does the rise of Q always occur before the fall of CK?

An example of flip-flop circuit

- An asynchronous circuit

Clarisó and Cortadella [2007]

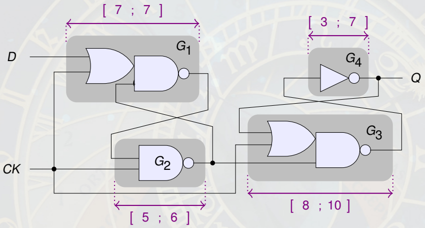


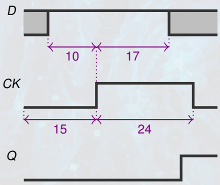


- Concurrent behaviour
  - 4 elements: G_1, G_2, G_3, G_4
  - 2 input signals (D and CK), 1 output signal (Q)
- Timing delays
  - Traversal delays of the gates: one interval per gate
  - Environment timing constants

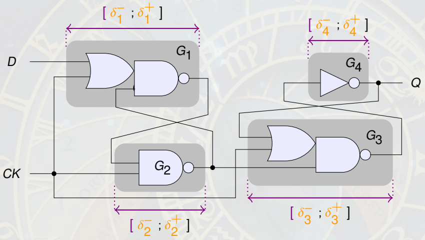
- Question
  - For these timing delays, does the rise of Q always occur before the fall of CK?
  - Timed model checking gives the answer: yes

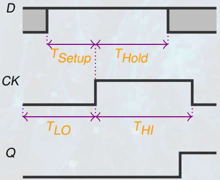
Flip-flop circuit: Timing parameters





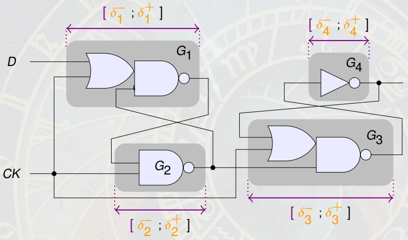
Flip-flop circuit: Timing parameters

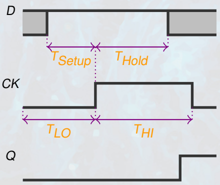




- Timing parameters
  - Traversal delays of the gates: one interval per gate
  - 4 environment parameters: T_{LO}, T_{HI}, T_{Setup} and T_{Hold}

Flip-flop circuit: Timing parameters





- Timing parameters
  - Traversal delays of the gates: one interval per gate
  - 4 environment parameters: T_{LO}, T_{HI}, T_{Setup} and T_{Hold}

- Question: which values of the parameters yield the same untimed behavior as the reference valuation (and hence for which the rise of Q always occur before the fall of CK)?

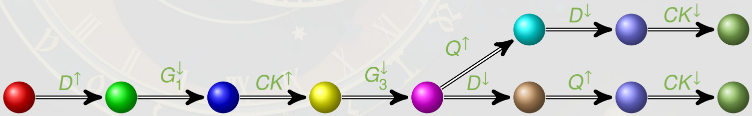
Trace set

- Trace set: set of all traces of a PTA
- Graphical representation under the form of a tree
  - (Does not give any information on the branching behavior though)

Trace set

- Trace set: set of all traces of a PTA
- Graphical representation under the form of a tree
  - (Does not give any information on the branching behavior though)

Example: trace set of the flip-flop circuit for the original valuation v_0



Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K_0 = \texttt{true}$

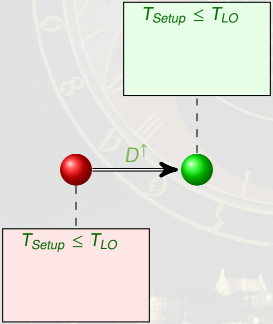
$T_{Setup} \leq T_{LO}$

Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K_0 = \texttt{true}$

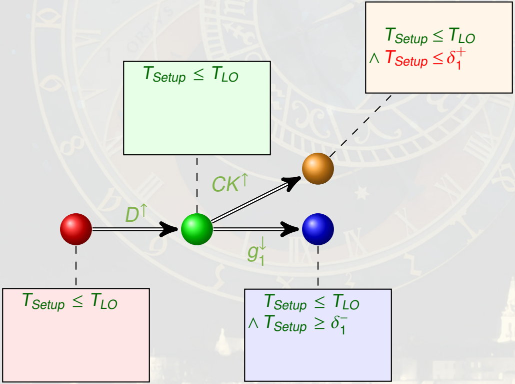


Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K_0 = \texttt{true}$



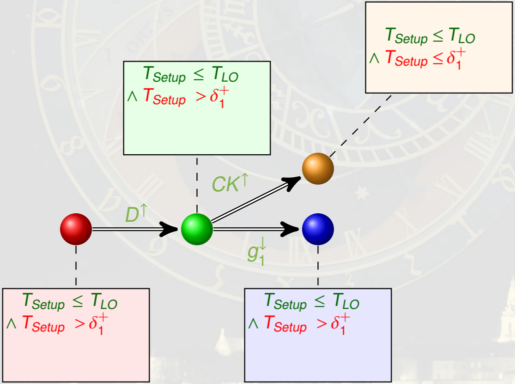
Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K_0 =$

$T_{Setup} > \delta_1^+$

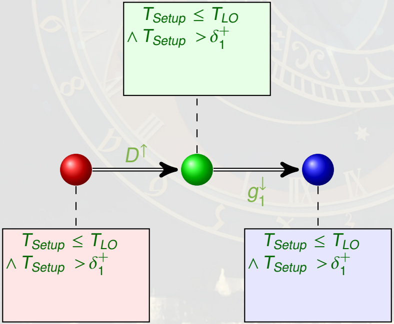


Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K_0 =$
$T_{Setup} > \delta_1^+$

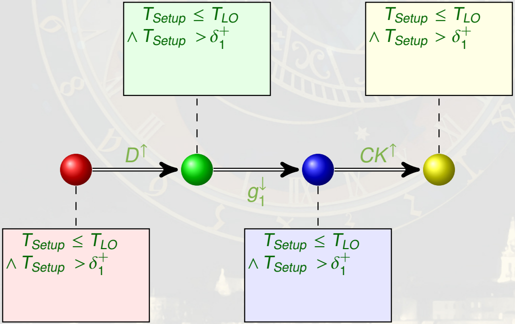


Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K_0 =$
$T_{Setup} > \delta_1^+$



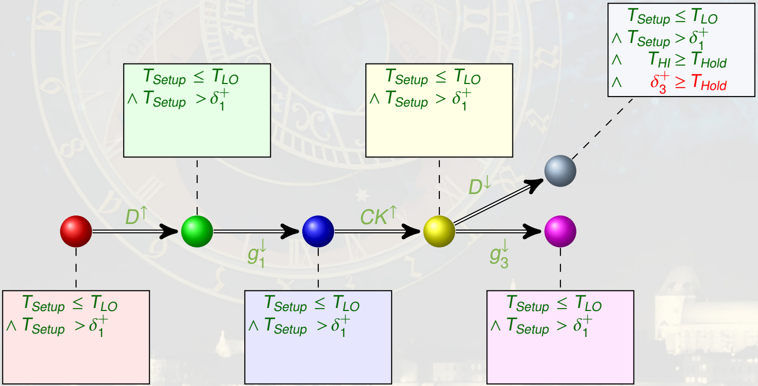
Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$K_0 =$

$T_{Setup} > \delta_1^+$



Application of TPsynth to the flip-flop circuit

v_0 :

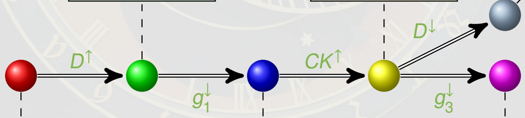
$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$$K_0 = T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{HI} \geq T_{Hold} \wedge \delta_3^+ \geq T_{Hold}$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$



$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

Application of TPsynth to the flip-flop circuit

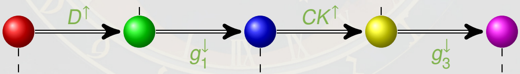
v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$$K_0 = T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$



$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

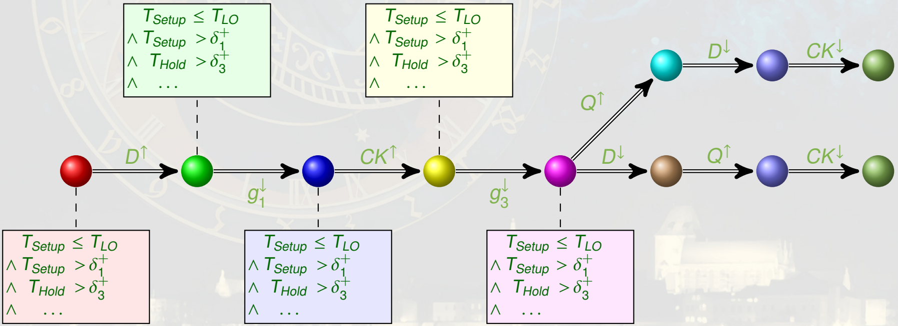
$$T_{Setup} \leq T_{LO} \wedge T_{Setup} > \delta_1^+ \wedge T_{Hold} > \delta_3^+$$

Application of TPsynth to the flip-flop circuit

v_0 :

$\delta_1^- = 7$	$\delta_1^+ = 7$	$T_{HI} = 24$
$\delta_2^- = 5$	$\delta_2^+ = 6$	$T_{LO} = 15$
$\delta_3^- = 8$	$\delta_3^+ = 10$	$T_{Setup} = 10$
$\delta_4^- = 3$	$\delta_4^+ = 7$	$T_{Hold} = 17$

$$K_0 = \begin{array}{ll} T_{Setup} > \delta_1^+ & \wedge\ \delta_3^+ + \delta_4^+ \geq T_{Hold} \\ \wedge\ T_{Hold} > \delta_3^+ & \wedge\ \delta_3^+ + \delta_4^+ < T_{HI} \\ \wedge\ T_{Setup} \leq T_{LO} & \wedge\ \delta_3^- + \delta_4^- \leq T_{Hold} \\ \wedge\ \delta_1^- > 0 & \end{array}$$



Software supporting parametric timed automata

Specification and verification of parametric models using parametric timed automata are supported by several software tools

- HyTech (also hybrid automata) Henzinger et al. [1997]
- PHAVer (also hybrid systems) Frehse [2005]
- Roméo (based on parametric time Petri nets) Lime et al. [2009]
- IMITATOR André et al. [2012]

Conclusion

Two algorithms:

- EFsynth: parametric reachability
- TPsynth: parametric trace preservation, with a measure of robustness Markey [2011]

Other algorithms (not presented):

- AFsynth: unavoidability synthesis (implemented in Roméo)
- Behavioural cartography (implemented in IMITATOR)

...but all these algorithms are costly.

Conclusion

Two algorithms:

- EFsynth: parametric reachability
- TPsynth: parametric trace preservation, with a measure of robustness Markey [2011]

Other algorithms (not presented):

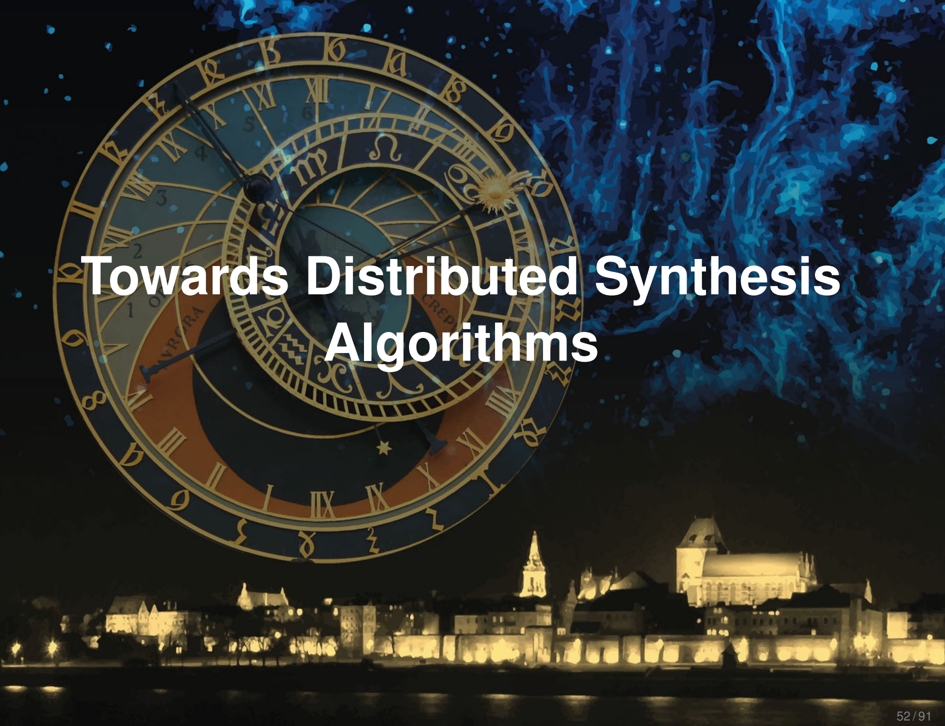
- AFsynth: unavoidability synthesis (implemented in Roméo)
- Behavioural cartography (implemented in IMITATOR)

. . . but all these algorithms are costly.

Let us see how to improve performances with distributed algorithms (next sequence)


ORTVS
AVRORA

Towards Distributed Synthesis Algorithms



First of all…

You have seen some synthesis algorithms for PTA addressing:

- parametric reachability (EFsynth)
- parametric trace preservation (TPsynth)

…but all these algorithms are costly.

First of all…

You have seen some synthesis algorithms for PTA addressing:

- parametric reachability (EFsynth)
- parametric trace preservation (TPsynth)

…but all these algorithms are costly.

Let us now see how to improve performances with distributed algorithms

Why distributed algorithms?

Algorithms for parameter synthesis for PTA are very costly

- time
- memory

Some reasons:

- expensive operations on polyhedra
- 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
- 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

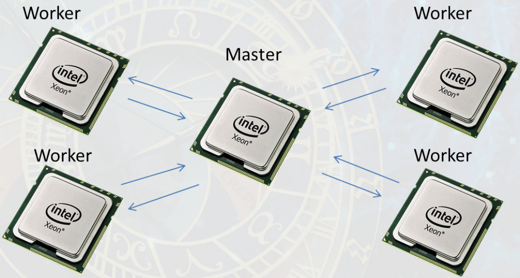
A more elaborate master-worker approach

Workers: run a “hybrid” algorithm

- PRP: parametric reachability preservation
- 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)
- based on integer points: 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

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
  - Note: not trivial at all

André et al. [2014, 2015a]

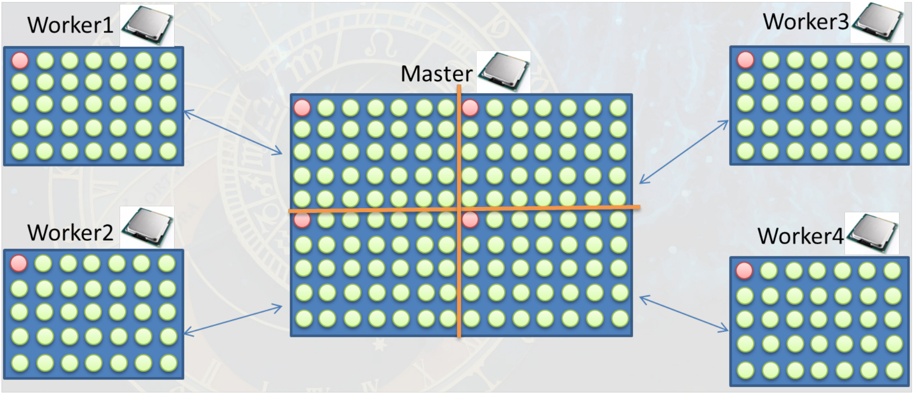
Dynamic domain decomposition

Most efficient distributed algorithm (so far!):
"Domain decomposition" scheme

- 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

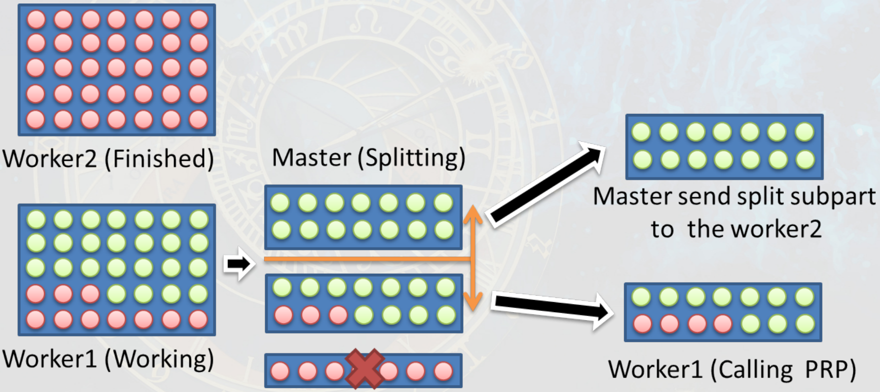
- Workers
  1. receive the subdomain from the master
  2. call PRP on the points of this subdomain
  3. send the results (list of constraints) back to the master
  4. ask for more work

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]

Conclusion

First version of distributed algorithms for PTA

What remains to be done…?

- Large space for improvement (44 faster with 128 nodes leaves much space for speedup)
- Multi-core parameter synthesis (on a single machine with several processors)

Conclusion

First version of distributed algorithms for PTA

What remains to be done. . . ?

- Large space for improvement (44 faster with 128 nodes leaves much space for speedup)
- Multi-core parameter synthesis (on a single machine with several processors)

Let us see some tool support (next sequence)


ORTVS
OCCAS
AVRORA

IMITATOR in a nutshell

First of all. . .

You now know about:

- Parametric timed automata
- parameter synthesis algorithms

First of all...

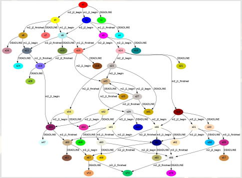
You now know about:

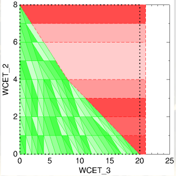
- Parametric timed automata
- parameter synthesis algorithms

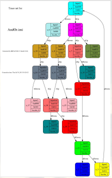
Let us now see some tool support

IMITATOR

- A tool for modelling and verifying real-time systems with unknown constants modelled with Parametric Timed Automata
  - Communication through (strong) broadcast synchronisation
  - 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
  - Behavioural 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

- Modelled 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]
- Solution to a challenge related to a distributed video processing system by Thales
- Formal timing analysis of music scores Fanchon and Jacquemard [2013]

Conclusion

At this stage, you know:

- Parametric timed automata
- synthesis algorithms for timing parameters

Conclusion

At this stage, you know:

- Parametric timed automata
- synthesis algorithms for timing parameters

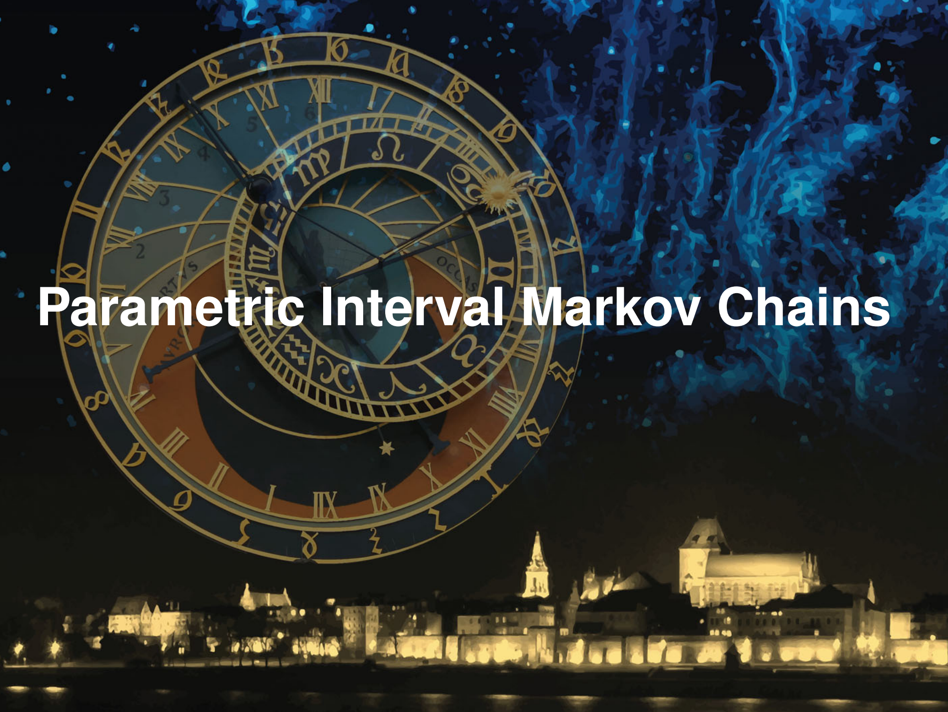
but need for parametric probabilities to capture:

- imprecisions
- robustness
- dimensioning

Let us address Markov chains with parameters (next sequence)


ORTVS
AVRORA

Parametric Interval Markov Chains



First of all. . .

You know about:

- parametric timed automata

First of all. . .

You know about:

- parametric timed automata

Need for parametric probabilities to capture:

- imprecisions
- robustness
- dimensioning

First of all…

You know about:

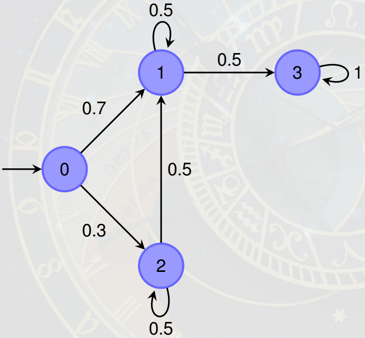
- parametric timed automata

Need for parametric probabilities to capture:

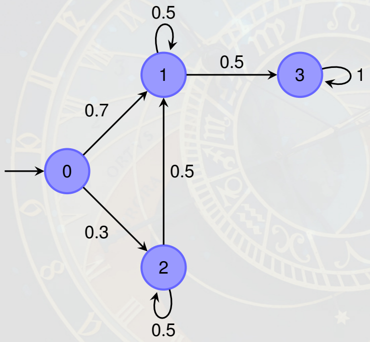
- imprecisions
- robustness
- dimensioning

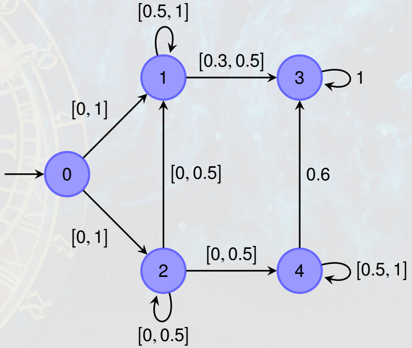
Let us now introduce Parametric Interval Markov Chains

Markov Chains (MCs)



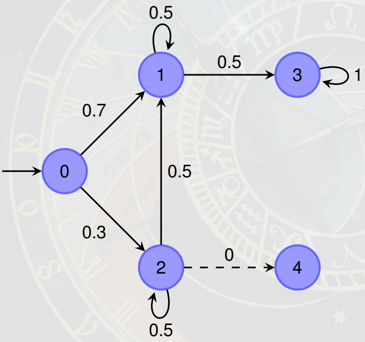
Interval Markov Chains (IMCs)



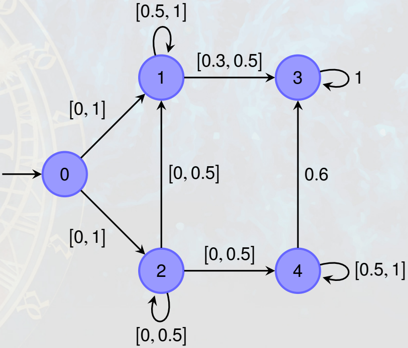


Specification (IMC)

Interval Markov Chains (IMCs)

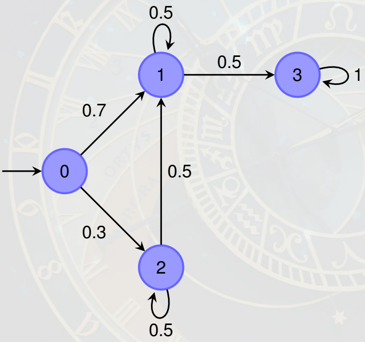


Implementation (MC)

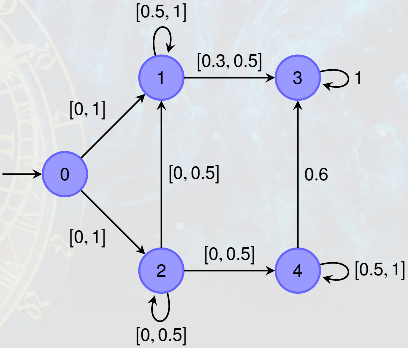


Specification (IMC)

Interval Markov Chains (IMCs)



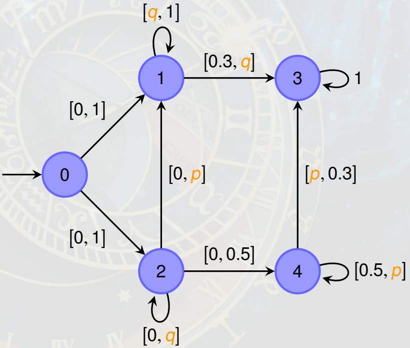
Implementation (MC)



Specification (IMC)

An IMC is consistent if it admits at least one implementation.

Parametric Interval Markov Chains (pIMCs)



Valuating the parameters of $\mathcal{I}$ with valuation v gives an IMC $v(\mathcal{I})$

n-consistency for IMCs

Definition

- State s in an IMC is 0-consistent if there exists a probability distribution over the successors of s that matches the intervals;
- State s in an IMC is n-consistent ($n \geq 1$) if:
  1. there exists a probability distribution ρ over the successors of s that matches the intervals and
  2. the successors s' such that $\rho(s') > 0$ are $(n-1)$-consistent.

n-consistency for IMCs

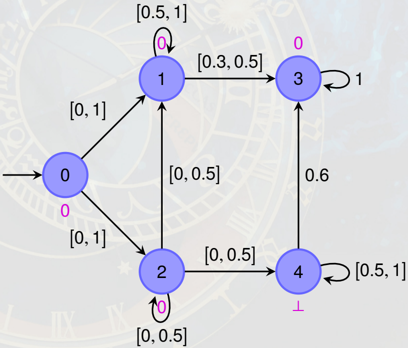
Definition

- State s in an IMC is 0-consistent if there exists a probability distribution over the successors of s that matches the intervals;
- State s in an IMC is n-consistent ($n \geq 1$) if:
  1. there exists a probability distribution ρ over the successors of s that matches the intervals and
  2. the successors s' such that $\rho(s') > 0$ are $(n-1)$-consistent.

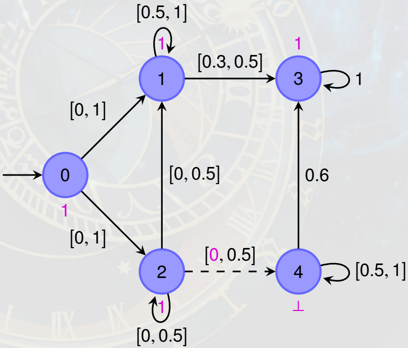
Theorem

An IMC with N states is consistent iff its initial state is N-consistent.

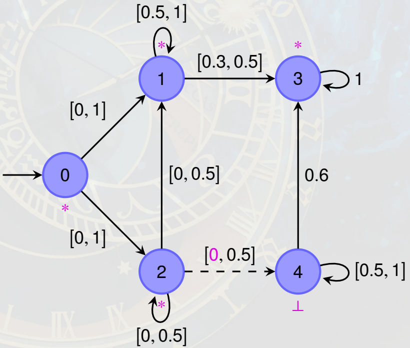
n-consistency for IMCs: first example



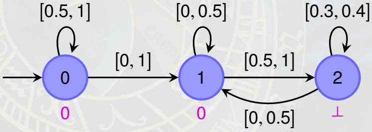
n-consistency for IMCs: first example



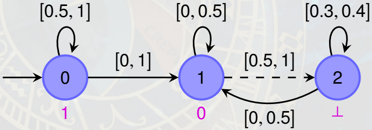
n-consistency for IMCs: first example



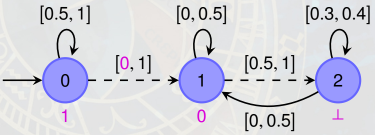
n-consistency for IMCs: second example



n-consistency for IMCs: second example



n-consistency for IMCs: second example



Conclusion

At this stage:

- you have an idea on Parametric Interval Markov Chains . . .
- you know how to check consistency for IMCs

Conclusion

At this stage:

- you have an idea on Parametric Interval Markov Chains . . .
- you know how to check consistency for IMCs

Let us see how to check consistency in PIMCs (next sequence)


ORTVS
AVRORA
OCCAS

Checking Consistency in Parametric Interval Markov Chains

First of all…

You know about:

- the Parametric Interval Markov Chains model
- checking consistency for IMCs

First of all…

You know about:

- the Parametric Interval Markov Chains model
- checking consistency for IMCs

Consistency problem for PIMCs:

- Does there exists a parameter valuation v such that IMC $v(\mathcal{I})$ is consistent?
- Is IMC $v(\mathcal{I})$ consistent for all parameter valuations v?
- Compute all parameter valuations v such that IMC $v(\mathcal{I})$ is consistent

First of all...

You know about:

- the Parametric Interval Markov Chains model
- checking consistency for IMCs

Consistency problem for PIMCs:

- Does there exists a parameter valuation v such that IMC $v(\mathcal{I})$ is consistent?
- Is IMC $v(\mathcal{I})$ consistent for all parameter valuations v?
- Compute all parameter valuations v such that IMC $v(\mathcal{I})$ is consistent

Let us now see how to check consistency in PIMCs

n-consistency constraints for pIMCs

- **Local consistency constraint for state** s wrt. some subset S' of its successors:

$$LC(s, S') = \left[\sum_{s' \in S'} \mathtt{Up}(s, s') \geq 1\right] \cap \left[\sum_{s' \in S'} \mathtt{Low}(s, s') \leq 1\right] \cap \left[\bigcap_{s' \in S'} \mathtt{Low}(s, s') \leq \mathtt{Up}(s, s')\right]$$

n-consistency constraints for pIMCs

- **n-consistency constraint for s given some cut-off successors**:
  $\mathtt{Cons}_0^X(s) = LC(s, \mathtt{Succ}(s) \setminus X) \cap [\bigcap_{s' \in X} \mathtt{Low}(s,s') = 0]$

  and for $n \geq 1$,

$$\mathtt{Cons}_n^X(s) = \left[\bigcap_{s' \in \mathtt{Succ}(s) \setminus X} \mathtt{Cons}_{n-1}(s') \right] \cap [LC(s, \mathtt{Succ}(s) \setminus X)]$$

$$\cap \left[\bigcap_{s' \in X} \mathtt{Low}(s,s') = 0 \right]$$

- **n-consistency constraint for s**:

$$\mathtt{Cons}_n(s) = \bigcup_{X \subseteq Z(s)} \mathtt{Cons}_n^X(s)$$

$Z(s)$ contains the successors of s for which $\mathtt{Low}$ is either 0 or a parameter

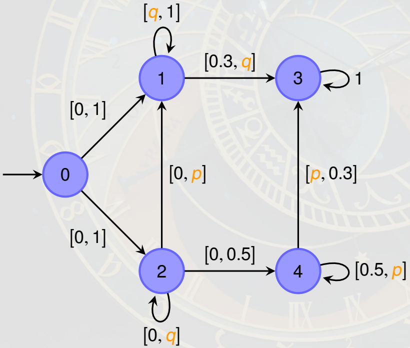
Consistency for pIMCs

Theorem (Delahaye et al. [2016])

Given a pIMC $\mathcal{I}$ with N states and initial state s_0, and a parameter valuation v:

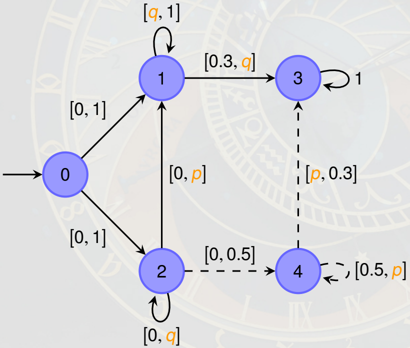
$$v(\mathcal{I}) \text{ is consistent iff } v \in \mathit{Cons}_N(s_0)$$

Consistency for PIMCs: a detailed example



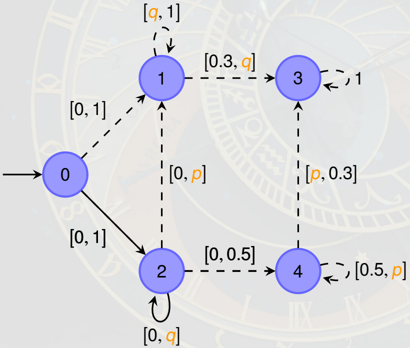
$$[(q \leq 0.7) \cap (q \geq 0.3)] \cup (q = 1)$$

Consistency for PIMCs: a detailed example



$$[(q \leq 0.7) \cap (q \geq 0.3)] \cup (q = 1)$$

Consistency for PIMCs: a detailed example



$$[(q \leq 0.7) \cap (q \geq 0.3)] \cup (q = 1)$$

Conclusion

At this stage:

- you know about parametric timed automata, their problems and algorithms
- you know about interval Markov chains with parametric probabilities

Conclusion

At this stage:

- you know about parametric timed automata, their problems and algorithms
- you know about interval Markov chains with parametric probabilities

Let us practice with IMITATOR


ORTVS
AVRORA

Bibliography

References I

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

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

André, É. (2015). What's decidable about parametric timed automata? In *Formal Techniques for Safety-Critical Systems - Fourth International Workshop, FTSCS 2015, Paris, France*, pages 52–68.

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

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

André, É., Coti, C., and Nguyen, H. G. (2015a). Enhanced distributed behavioral cartography of parametric timed automata. In Butler, M., Conchon, S., and Zaïdi, F., editors, *Proceedings of the 17th International Conference on Formal Engineering Methods (ICFEM'15)*, Lecture Notes in Computer Science. Springer.

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

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

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.

André, É., Lime, D., and Roux, O. H. (2016). Decision problems for parametric timed automata. Technical report.

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

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

References II

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

Bundala, D. and Ouaknine, J. (2014). Advances in parametric real-time reasoning. In *MFCS*, volume 8634 of *Lecture Notes in Computer Science*, pages 123–134. Springer.

Clarisó, R. and Cortadella, J. (2007). The octahedron abstract domain. *Science of Computer Programming*, 64(1):115–139.

Delahaye, B., Lime, D., and Petrucci, L. (2016). Parameter synthesis for parametric interval Markov chains. In *Proc. of the 17th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'16), St. Petersburg, Florida, USA*, volume 9583, pages 372–390. Springer.

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

Frehse, G. (2005). Phaver: Algorithmic verification of hybrid systems past HyTech. In *Hybrid Systems: Computation and Control, 8th International Workshop, HSCC 2005, Zurich, Switzerland*, pages 258–273.

Fribourg, L., Lesens, D., Moro, P., and Soulat, R. (2012). Robustness analysis for scheduling problems using the inverse method. In *TIME'12*, pages 73–80. IEEE Computer Society Press.

Henzinger, T. A., Ho, P.-H., and Wong-Toi, H. (1997). HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122.

Hune, T., Romijn, J., Stoelinga, M., and Vaandrager, F. W. (2002). Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220.

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

Lime, D., Roux, O. H., Seidner, C., and Traonouez, L.-M. (2009). Romeo: A parametric model-checker for Petri nets with stopwatches. In *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57. Springer.

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

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


ORTVS
AVRORA