Petri Nets Tutorial, Parametric Verification (session 3)

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

- Petri Nets with Parameters
  - Parametric Petri Nets.
  - Parametric Time Petri Nets.
  - Roméo in a nutshell.
- Action synthesis
  - Model.
  - SPATULA in a nutshell.


ORTVS
AVRORA

Parametric Petri Nets

First of all. . .

You now know about:

- parametric timed automata
- synthesis of timing parameters
- interval Markov chains with parameters

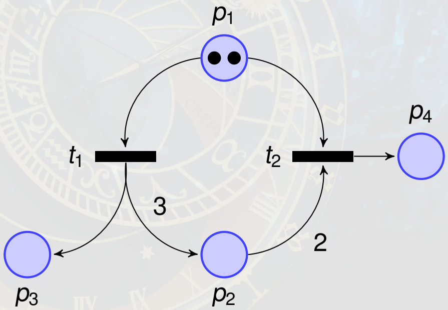
First of all…

You now know about:

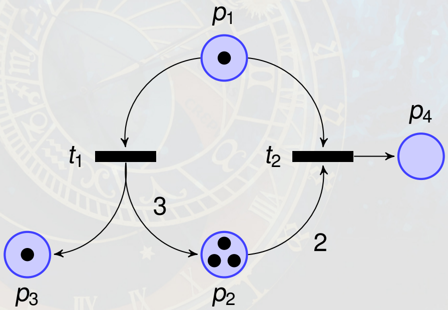
- parametric timed automata
- synthesis of timing parameters
- interval Markov chains with parameters

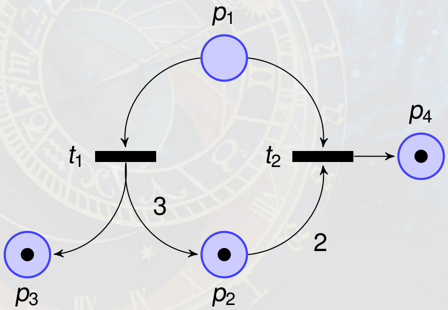
Let us now see Parametric Petri nets

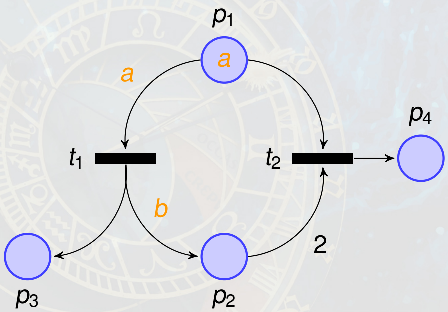
Petri nets



Petri nets




p_1
p_4
t_1
t_2
3
2
p_3
p_2



- initial marking: number of processes, initial value of a semaphore, etc.
- pre weights: number of processes to synchronise, number of items to take, etc.
- post weights: number of processes to spawn, number of items to give, etc.

Definition (Coverability)

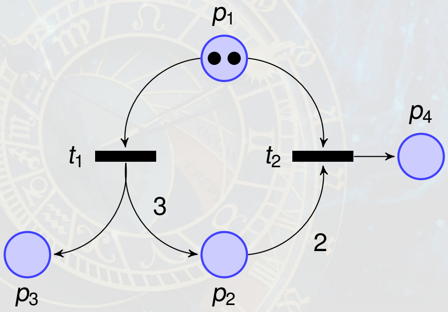
Given a marking m, does there exist a reachable marking m' such that $m' \geq m$

Definition (Coverability)

Given a marking m, does there exist a reachable marking m' such that $m' \geq m$

- Coverability is EXPSPACE-complete in Petri nets;
- It is equivalent to knowing if some transition can fire;
- This includes many safety properties.

Coverability: Example



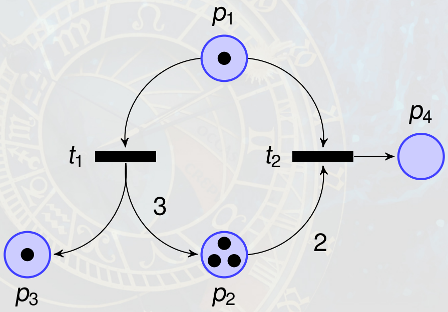
Some markings that can be covered:

$(0, 0, 0, 0)$ – $(1, 1, 1, 0)$ – $(0, 1, 1, 1)$

Some markings that cannot be covered:

$(1, 0, 0, 1)$ – $(2, 0, 1, 0)$ – $(0, 4, 0, 0)$

Coverability: Example



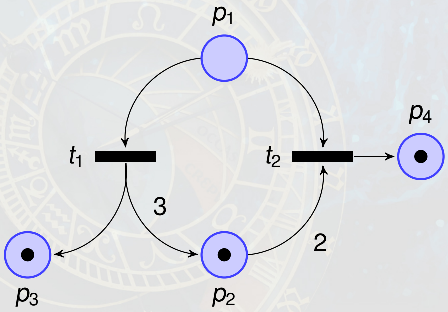
Some markings that can be covered:

$(0,0,0,0)$ – $(1,1,1,0)$ – $(0,1,1,1)$

Some markings that cannot be covered:

$(1,0,0,1)$ – $(2,0,1,0)$ – $(0,4,0,0)$

Coverability: Example



Some markings that can be covered:

$(0,0,0,0)$ – $(1,1,1,0)$ – $(0,1,1,1)$

Some markings that cannot be covered:

$(1,0,0,1)$ – $(2,0,1,0)$ – $(0,4,0,0)$

Coverability in Parametric Petri Nets

Definition (E-cov: Existential Coverability)

Is some given marking coverable for at least one parameter valuation?

Definition (U-cov: Universal Coverability)

Is some given marking coverable for all the parameter valuations?

Parametric Coverability is Undecidable

Theorem

E-cov and U-cov are undecidable for parametric Petri nets.

They can simulate 2-counter machines:

- two counters C_1, C_2,
- states $P = \{p_0, ...p_m\}$, a terminal state labelled *halt*
- finite list of instructions $I_1, ..., I_s$ among the following list:
  - increment a counter and go to I_j
  - if the counter is positive decrement it and go to I_j
  - if the counter is null go to I_i else go to I_j

Counters are always non negative.

An Example of 2-Counter Machine

$$\boxed{\begin{array}{l} \text{in } p_1 : \; C_1 := C_1 + 1; \text{goto } p_2; \\ \text{in } p_2 : \; C_2 := C_2 + 1; \text{goto } p_1; \end{array}}$$

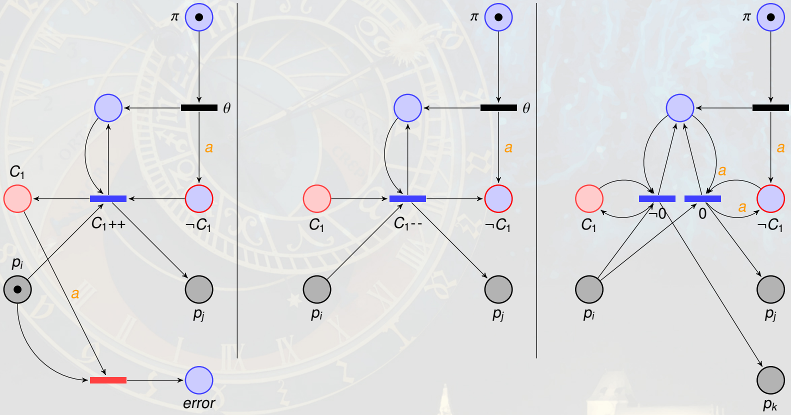
Successive configurations:

$$(p_1, C_1 = 0, C_2 = 0) \rightarrow (p_2, C_1 = 1, C_2 = 0) \rightarrow (p_1, C_1 = 1, C_2 = 1) \\ \rightarrow (p_2, C_1 = 2, C_2 = 1) \rightarrow ...$$

Simulation of a 2-Counters Machine

- The halting problem (whether some state `halt` of the machine is reachable) can be reduced to E-cov;
- The counters boundedness problem (whether the counters values stay in a finite set) can be reduced to U-cov;
- Both problems are undecidable for 2-counter machines Minsky [1967].
- From any machine $\mathcal{M}$, we build a parametric Petri net $\mathcal{N}_{\mathcal{M}}$ encoding it such that:
  - $\mathcal{M}$ halts iff there exists a parameter valuation v such that place $p_{\mathtt{halt}}$ is coverable in $v(\mathcal{N}_{\mathcal{M}})$.
  - a counter of $\mathcal{M}$ is unbounded iff for all parameter valuations v, place p_{error} is coverable in $v(\mathcal{N}_{\mathcal{M}})$.

Simulation of Instructions



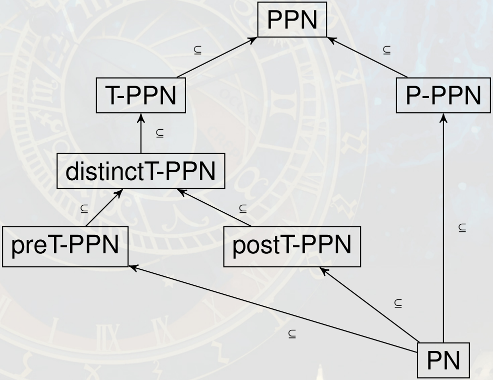
incrementation of a counter

decrementation of a counter

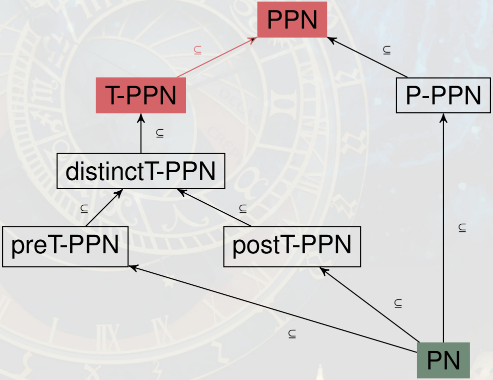
zero test of a counter

By construction, $m(C_1) + m(\neg C_1) = a$

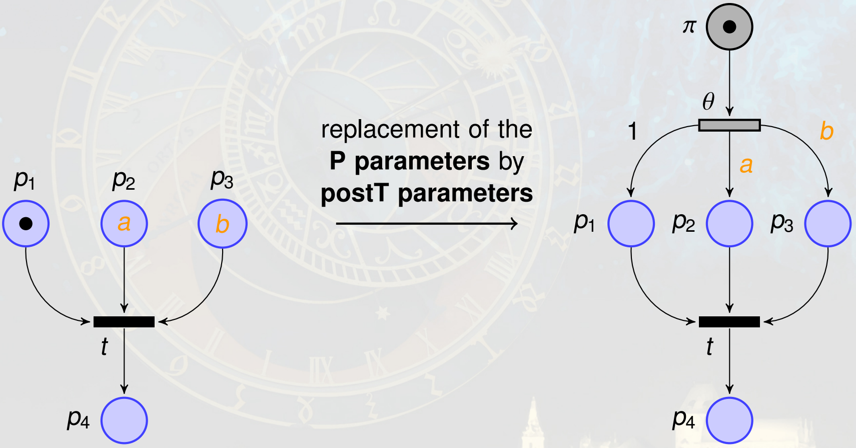
Decidable Subclasses: A Hierarchy of Parametric PNs



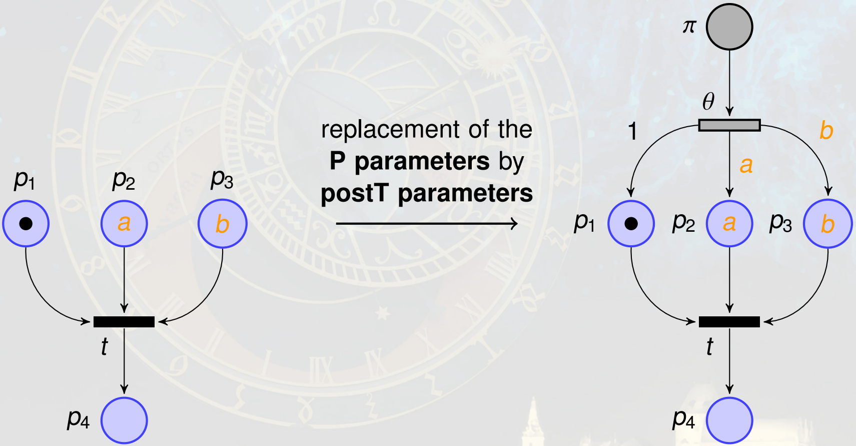
Decidable Subclasses: A Hierarchy of Parametric PNs



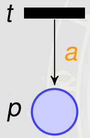
From Markings to Output Weights



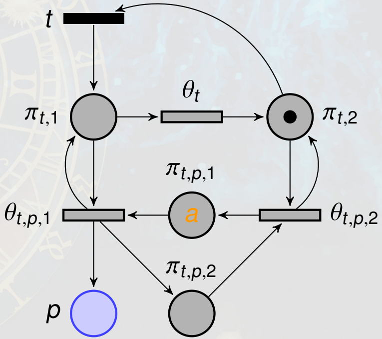
From Markings to Output Weights



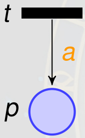
From Output Weights to Markings



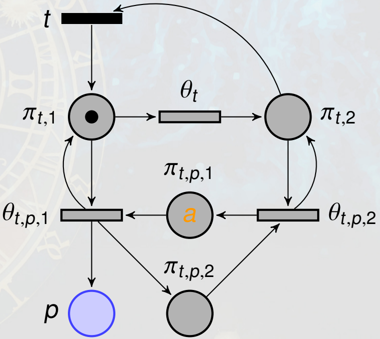
replacement of the
postT parameters
by **P parameters**



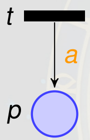
From Output Weights to Markings



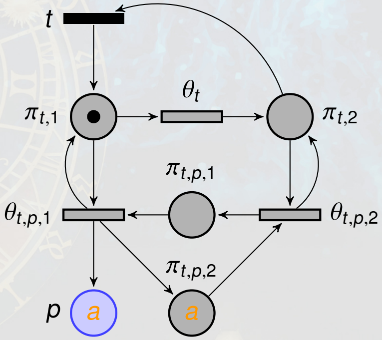
replacement of the **postT parameters** by **P parameters**



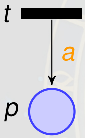
From Output Weights to Markings



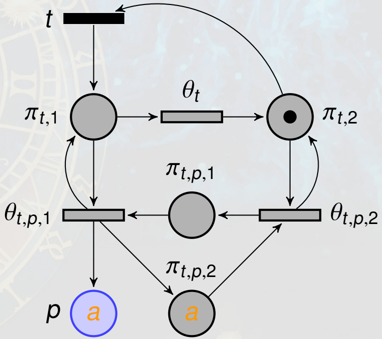
replacement of the **postT parameters** by **P parameters**



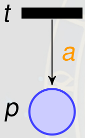
From Output Weights to Markings



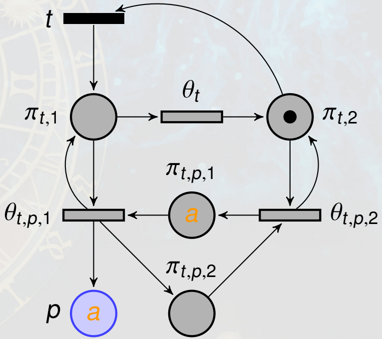
replacement of the **postT parameters** by **P parameters**



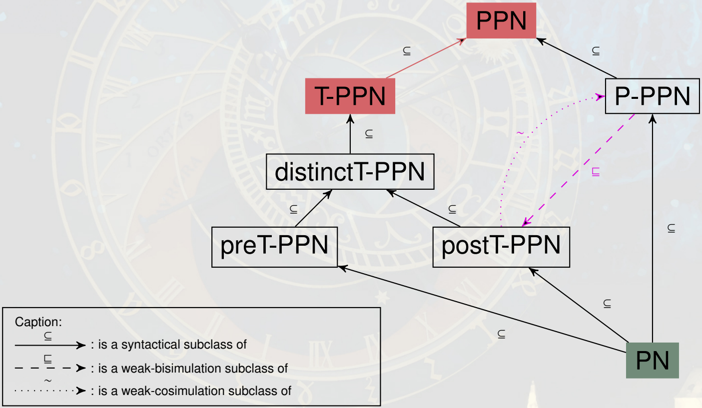
From Output Weights to Markings



replacement of the
postT parameters
by **P parameters**

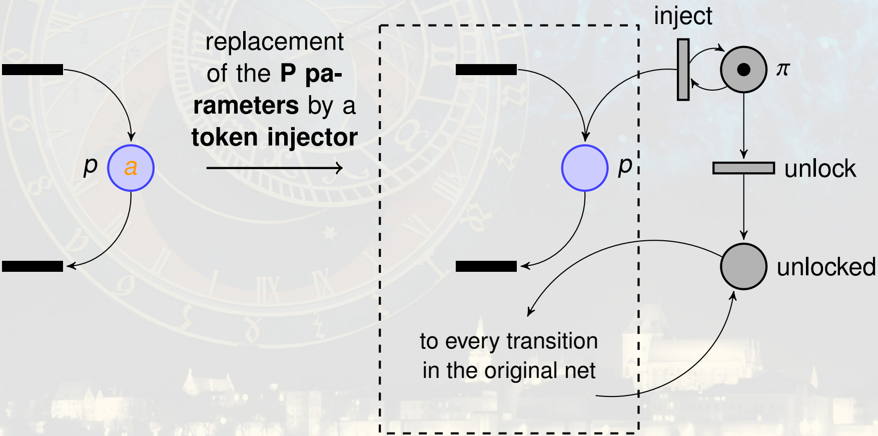


Decidable Subclasses: A Hierarchy of Parametric PNs

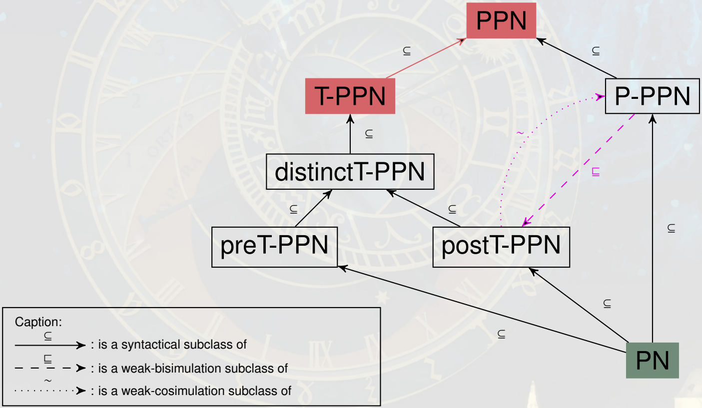


From Parametric Markings to Classic Petri Nets

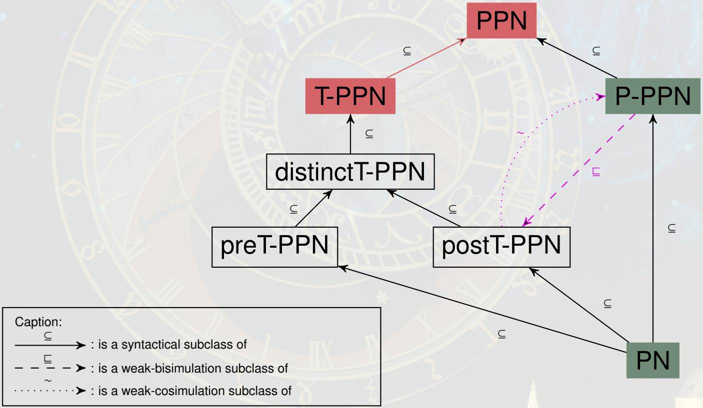
- for U-cov: all parameters to 0 is the worst case ;
- for E-cov:



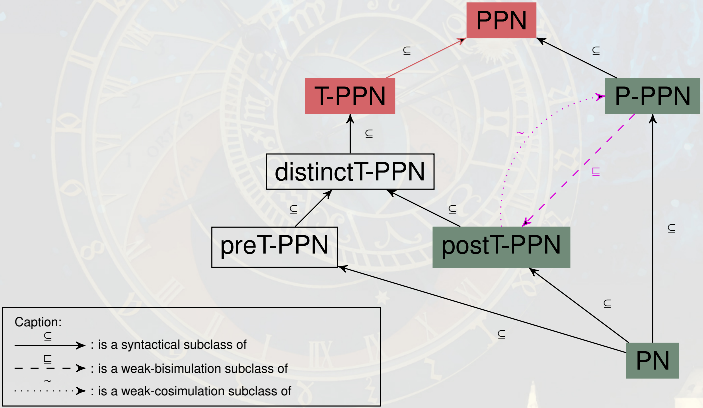
Decidable Subclasses: A Hierarchy of Parametric PNs



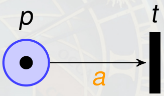
Decidable Subclasses: A Hierarchy of Parametric PNs



Decidable Subclasses: A Hierarchy of Parametric PNs

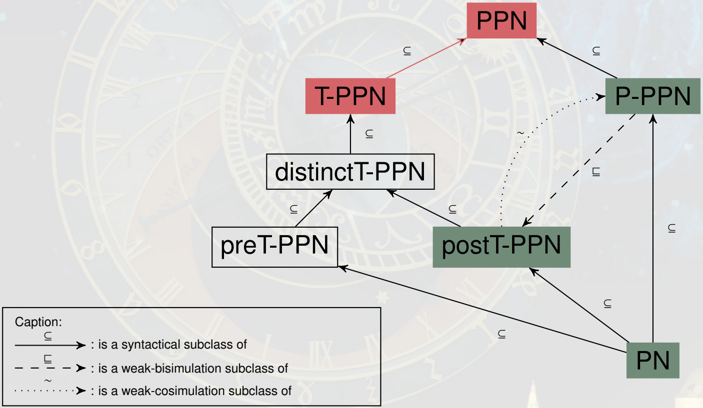


Deciding Coverability with Parametric Input Weights

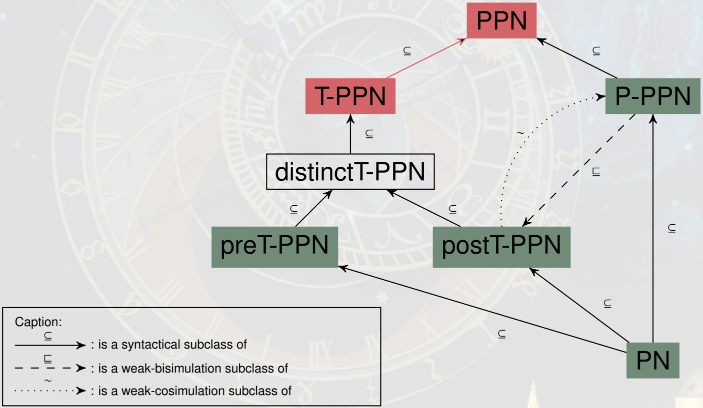


- for E-cov: all parameters to 0 is the best case;
- for U-cov:
  - extend the coverability tree construction of Karp & Miller Karp and Miller [1969]
  - consider that a transition with a parametric input weight can fire only if the corresponding place can become unbounded (i.e. has an ω marking).

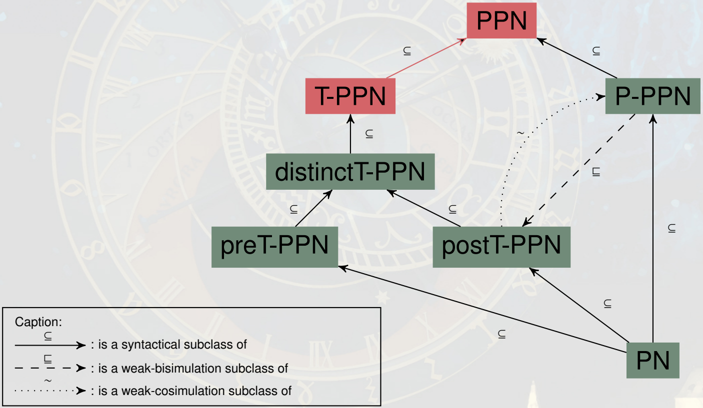
Decidable Subclasses: A Hierarchy of Parametric PNs



Decidable Subclasses: A Hierarchy of Parametric PNs



Decidable Subclasses: A Hierarchy of Parametric PNs



Conclusion

- Parametric Petri Nets are an expressive but undecidable model;
- There are interesting and still expressive decidable subclasses;
- For those subclasses, parametric coverability is EXPSPACE-complete (no upper bound for U – *cov* for input weights)
- The problem of synthesis is still open.

Conclusion

- Parametric Petri Nets are an expressive but undecidable model;
- There are interesting and still expressive decidable subclasses;
- For those subclasses, parametric coverability is EXPSPACE-complete (no upper bound for $U-cov$ for input weights)
- The problem of synthesis is still open.

Let us now see how timing parameters can be introduced in (time) Petri Nets


ORTVS
AVRORA

Parametric Time Petri Nets

First of all...

You now know about:

- Parametric Petri nets
- Decidability issues

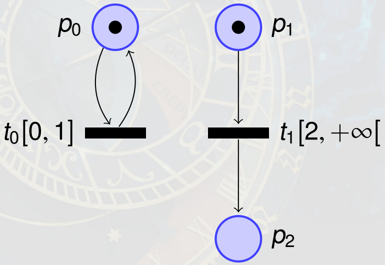
First of all...

You now know about:

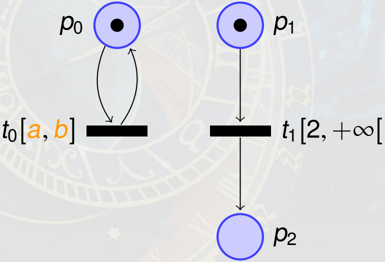
- Parametric Petri nets
- Decidability issues

Let us now review Parametric Time Petri nets

Parametric Time Petri Nets (PTPNs)



Parametric Time Petri Nets (PTPNs)



Undecidability Results for Parametric TPNs

- We have a structural translation from timed automata to bounded time Petri nets preserving timed language (implying state reachability) Bérard et al. [2013]
- Has one gadget per simple constraint in guards and timing constants appear explicitly;
- It extends trivially to parameterized guards.

Theorem

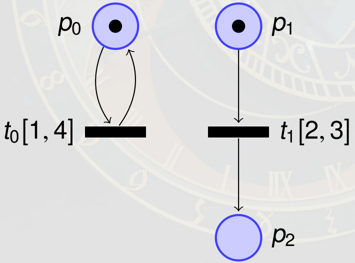
The EF-emptiness problem is undecidable for bounded parametric time Petri nets.

Decidability Results for Parametric TPNs

- We also have structural translations the other way round (preserving almost everything); Bérard et al. [2013]
- All decidability results carry over to parametric Petri nets;
- The symbolic state abstraction presented earlier can also be defined for PTPNs; Gardey et al. [2006]
- EFSynth and similar algorithms can be used as is for PTPNs!
- But TPNs enjoy a "better" symbolic abstraction: Berthomieu & Menasche's State Classes. Berthomieu and Menasche [1983]; Berthomieu and Diaz [1991]

State Classes for Time Petri Nets

- State classes also regroup states obtained with the same discrete transition sequence in a pair (I, Z) where Z is a zone;
- But states record time to firing instead of time elapsed;



Initially:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq t_1 \leq 3 \end{cases}$$

Fire t_0:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq t_1 \leq 3 \\ t_0 \leq t_1 \end{cases}$$

New times to fire:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 2 \leq t_1' + t_0 \leq 3 \\ t_0 \leq t_1' + t_0 \end{cases}$$

Disabled (incl. t_0):

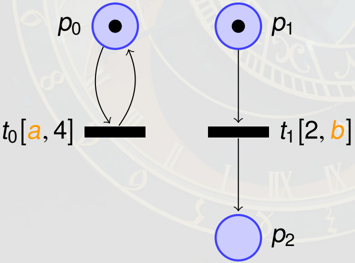
$$\begin{cases} 0 \leq t_1' \leq 2 \end{cases}$$

Newly enabled:

$$\begin{cases} 1 \leq t_0 \leq 4 \\ 0 \leq t_1 \leq 2 \end{cases}$$

State Classes for Parametric Time Petri Nets

- Successive state classes computations are done with classic polyhedral operations;
- They can be extended to account for timing parameters Traonouez et al. [2009]:



Initially:

$$\begin{cases} a \leq t_0 \leq 4 \\ 2 \leq t_1 \leq b \end{cases}$$

Fire t_0:

$$\begin{cases} a \leq t_0 \leq 4 \\ 2 \leq t_1 \leq b \\ t_0 \leq t_1 \\ (a \leq b) \end{cases}$$

New times to fire:

$$\begin{cases} a \leq t_0 \leq 4 \\ 2 \leq t_1' + t_0 \leq b \\ t_0 \leq t_1' + t_0 \end{cases}$$

Disabled (incl. t_0):

$$\begin{cases} 0 \leq t_1' \leq b - a \end{cases}$$

Newly enabled:

$$\begin{cases} a \leq t_0 \leq 4 \\ 0 \leq t_1 \leq b - a \end{cases}$$

Synthesis for Parametric TPNs

- EFSynth works the same with parametric state classes;

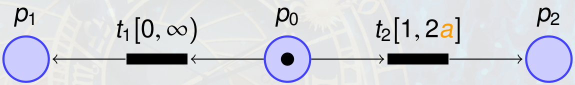
$$\mathrm{EF}_G(S,M) = \begin{cases} Z{\downarrow}_P & \text{if } I \in G \\ \emptyset & \text{if } S \in M \\ \bigcup_{\substack{t \in T \\ S' = \mathrm{Next}(S,t)}} \mathrm{EF}_G\big(S', M \cup \{S\}\big) & \text{otherwise.} \end{cases}$$

- We can also do synthesis for inevitability Jovanović et al. [2015]:

$$\mathrm{AF}_G(S,M) = \begin{cases} Z{\downarrow}_P & \text{if } I \in G \\ \emptyset & \text{if } S \in M \\ \Big(\bigcap_{\substack{t \in T \\ S' = \mathrm{Next}(S,t)}} \big(\mathrm{AF}_G\big(S', M \cup \{S\}\big) \cup (\mathbb{Q}^P \setminus S'{\downarrow}_P)\big)\Big) & \text{otherwise} \end{cases}$$

- $S = (I, Z)$;
- G a set of markings to reach;
- M is a list of visited state classes;
- $\mathrm{Next}(S,t)$ computes the state class successor of S by transition t;
- termination is not guaranteed.

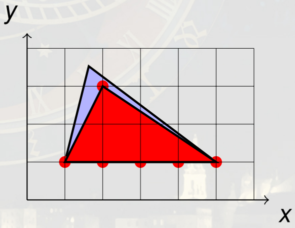
AF: Cutting for More



- Put a token in p_1: no constraint
- Put a token in p_2: $a \geq \frac{1}{2}$
- Ensuring both paths are possible (for AF ($p_1 > 0$ or $p_2 > 0$)): $a \geq \frac{1}{2}$
- Or we can cut t_2 and p_2 off with $a < \frac{1}{2}$ and the property is satisfied with no further constraint
- Finally, AF ($p_1 > 0$ or $p_2 > 0$) is satisfied for all values of a.

Symbolic Synthesis for Bounded Integers

- EF-emptiness is undecidable for integer parameters Alur et al. [1993];
- It is undecidable for bounded rational parameters Miller [2000];
- It is PSPACE-complete for bounded integer parameters Jovanović et al. [2015].
  - non-deterministically guess a parameter valuation and store it (polynomial storage size);
  - instantiate the PTA or PTPN and solve the problem (PSPACE);
  - PSPACE = NPSPACE (Savitch's theorem).
- Synthesis can be done symbolically, using integer hulls:



Symbolic Synthesis for Bounded Integer Parameters

- IEF computes polyhedra containing exactly the “good” integer parameter valuations:

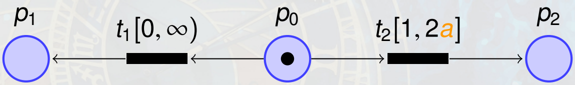
$$\mathrm{IEF}_G(S, M) = \begin{cases} Z{\downarrow_P} & \text{if } l \in G \\ \emptyset & \text{if } S \in M \\ \bigcup_{\substack{t \in T \\ S' = \mathrm{IH}(\mathrm{Next}(S,t))}} \mathrm{IEF}_G\big(S', M \cup \{S\}\big) & \text{otherwise.} \end{cases}$$

- It is guaranteed to terminate when the parameters are bounded;
- AF can be modified similarly.

Density of the Results

- The question:
  - the result of IEF or IAF is a union of convex polyhedra;
  - we know that these sets contain exactly the “good” integer valuations;
  - but what of the non-integer valuations in those polyhedra?
- The short answer:
  - they are all “good” for IEF (but we can do a bit better);
  - they are in general not all “good” for IAF (and we can do a bit better).

The Result of IAF is not Dense



- To ensure AF ($p_1 > 0$), cut t_2 and p_2, i.e., take $a < \frac{1}{2}$;
- When p_2 is marked, $Z_2 = \{1 \leq x \wedge 1 \leq 2a\}$, so $\mathrm{IH}(C_2) = \{1 \leq x \wedge 1 \leq a\}$
- So, to cut ($p_2 = 1, \mathrm{IH}(Z_2)$), we need $a < 1$.
- $\frac{1}{2} \leq a < 1$ are not “good” valuations.

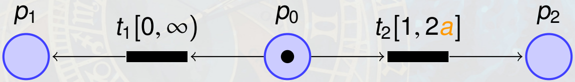
- In IAF, we cut off not enough states because $\mathrm{IH}(Z) \subseteq Z$;
- Solution: use integer hulls only for convergence André et al. [2015]:

$$\mathrm{RIEF}_G(S, M) = \begin{cases} Z{\downarrow}_P & \text{if } l \in G \\ \emptyset & \text{if } \mathrm{IH}(S) \in M \\ \bigcup_{\substack{t \in T \\ S' = \mathrm{Next}(S,t)}} \mathrm{EF}_G\big(S', M \cup \{\mathrm{IH}(S)\}\big) & \text{otherwise.} \end{cases}$$

$$\mathrm{RIAF}_G(S, M) = \begin{cases} Z{\downarrow}_P & \text{if } l \in G \\ \emptyset & \text{if } \mathrm{IH}(S) \in M \\ \Big(\bigcap_{\substack{t \in T \\ S' = \mathrm{Next}(S,t)}} \big(\mathrm{AF}_G\big(S', M \cup \{\mathrm{IH}(S)\}\big) \cup (\mathbb{Q}^P \setminus S'{\downarrow}_P)\big)\Big) & \text{otherwise} \end{cases}$$

- Gives a “dense” underapproximation containing at least all integer valuations.

Dense Integer-preserving Underapproxations



- AF l_1: $a < \frac{1}{2}$ instead of (erroneous) $a < 1$ for IAF
- EF l_2: $a \geq \frac{1}{2}$ instead of $a \geq 1$ for IEF

Conclusion

- Time Petri nets are well-suited to timing parametrization;
- Bounded PTPNs globally have the same decidability results as PTA;
- Synthesis (semi-)algorithms for PTA can be adapted for PTPN (and are sometimes a bit simpler);
- They can use state classes;
- General synthesis is hard and approximate/partial synthesis is a good way to address this problem;

- Time Petri nets are well-suited to timing parametrization;
- Bounded PTPNs globally have the same decidability results as PTA;
- Synthesis (semi-)algorithms for PTA can be adapted for PTPN (and are sometimes a bit simpler);
- They can use state classes;
- General synthesis is hard and approximate/partial synthesis is a good way to address this problem;

Roméo is a tool that supports parametric TPNs (next sequence)


ORTVS
AVRORA

Roméo in a nutshell

First of all . . .

You know that:

- Time Petri nets are well-suited to timing parametrization;
- Bounded PTPNs globally have the same decidability results as PTA;
- Synthesis (semi-)algorithms for PTA can be adapted for PTPN (and are sometimes a bit simpler);
- They can use state classes;
- General synthesis is hard and approximate/partial synthesis is a good way to address this problem;

First of all . . .

You know that:

- Time Petri nets are well-suited to timing parametrization;
- Bounded PTPNs globally have the same decidability results as PTA;
- Synthesis (semi-)algorithms for PTA can be adapted for PTPN (and are sometimes a bit simpler);
- They can use state classes;
- General synthesis is hard and approximate/partial synthesis is a good way to address this problem;

ROMÉO is a tool that supports parametric TPNs

Roméo

- An analysis tool / model-checker for time Petri nets with
  - timing parameters;
  - hybrid extensions;
  - discrete variables;
- Developed at Nantes since 2000, mostly by Olivier H. Roux and Didier Lime;
- Tool papers Gardey et al. [2005]; Lime et al. [2009]
- Free and open-source (CeCILL license)

Available at `http://romeo.rts-software.org/`

Conclusion

At this stage, you know about:

- Petri nets with discrete parameters
- time Petri nets with timing parameters

Conclusion

At this stage, you know about:

- Petri nets with discrete parameters
- time Petri nets with timing parameters

Let us address synthesis of actions (next sequence)


ORTVS
OCCAS
AVRORA
CREP

Action Synthesis

First of all...

You know about:

- Petri nets with discrete parameters
- time Petri nets with timing parameters

First of all. . .

You know about:

- Petri nets with discrete parameters
- time Petri nets with timing parameters

Let us now address synthesis of actions

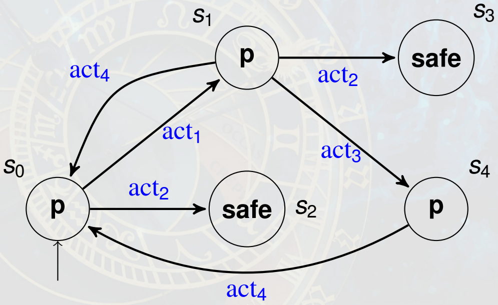
MTS: *Kripke structures with action-labelled transitions*

MTS (model) is a 5-tuple $\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{L})$, where:

- $\mathcal{S}$ – a set of states,
- $s^0 \in \mathcal{S}$ – the initial state,
- $\mathcal{A}$ – a set of actions,
- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{A} \times \mathcal{S}$ – a labelled transition relation,
- $\mathcal{PV}$ – a set of the propositional variables,
- $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{PV}}$ – a labelling function.

A path π in $\mathcal{M}$ is a maximal sequence $s_0 a_0 s_1 a_1 \ldots$ of states and actions such that $(s_i, a_i, s_{i+1}) \in \mathcal{T}$.

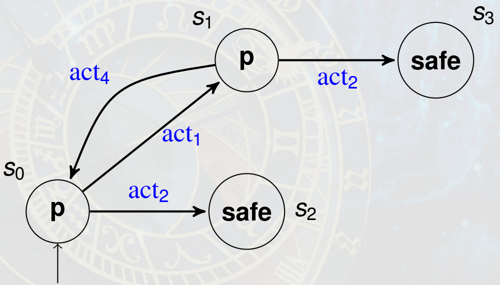
Allowed and disabled actions



$A \subseteq \mathcal{A}$ – a set of allowed actions

- $\Pi(A, s)$ – the maximal paths over A, starting from s

Allowed and disabled actions



$A \subseteq \mathcal{A}$ – a set of allowed actions

- $\Pi(A, s)$ – the maximal paths over A, starting from s

E.g., $\Pi(\{\mathrm{act}_1, \mathrm{act}_2, \mathrm{act}_4\}, s_0) =$
$\{(s_0 \mathrm{act}_1 s_1 \mathrm{act}_4)^\omega + (s_0 \mathrm{act}_1 s_1 \mathrm{act}_4)^* s_0 \mathrm{act}_1 s_1 \mathrm{act}_2 s_3 + (s_0 \mathrm{act}_1 s_1 \mathrm{act}_4)^* s_0 \mathrm{act}_2 s_2\}$

pmARCTL: CTL *with actions/variable subscripts*

ActSets – non-empty subsets of $\mathcal{A}$
ActVars – the action variables

pmARCTL: the formulae ϕ generated by the BNF grammar:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid E_\alpha X\phi \mid E_\alpha G\phi \mid E_\alpha(\phi \; U \; \phi)$$

$p \in \mathcal{PV}$, $\alpha \in$ ActSets $\cup$ ActVars

- E_α – *"there exists a maximal path over* α*"*
- X, G, U – *neXt, Globally, Until*

Parametric ARCTL

pmARCTL: CTL *with actions/variable subscripts*

ActSets – non-empty subsets of $\mathcal{A}$
ActVars – the action variables

pmARCTL: the formulae ϕ generated by the BNF grammar:

$$\phi ::= p \mid \neg\phi \mid \phi \vee \phi \mid E_\alpha X\phi \mid E_\alpha G\phi \mid E_\alpha(\phi\ U\ \phi)$$

$p \in \mathcal{PV}$, $\alpha \in$ ActSets $\cup$ ActVars

- E_α – *"there exists a maximal path over* α*"*
- X, G, U – *neXt, Globally, Until*
- (derived) A_α – *"for each maximal path over* α*"*
- (derived) F – *"in the future"*

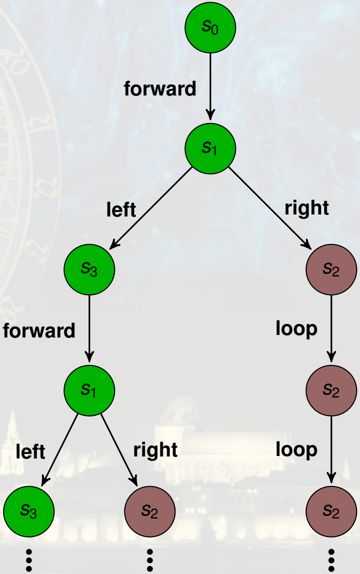
Parametric ARCTL: semantics

States:

Labelled by p

Labelled by q

Properties:



Parametric ARCTL: semantics

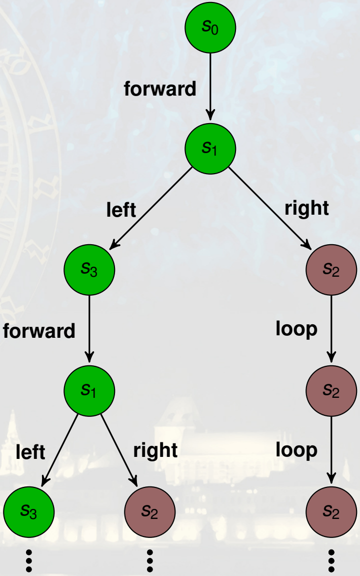
States:

Labelled by p

Labelled by q

Properties:

- $s_0 \models E_{\{\textbf{forward},\textbf{left}\}} Gp$



Parametric ARCTL: semantics

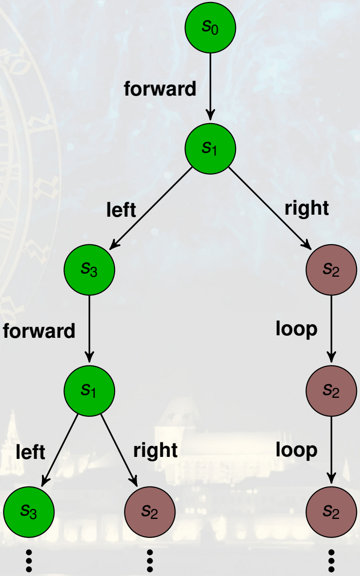
States:

Labelled by p

Labelled by q

Properties:

- $s_0 \models E_{\{\textbf{forward},\textbf{left}\}} Gp$
- $s_0 \models E_{\{\textbf{forward},\textbf{right}\}} pUq$



Parametric ARCTL: semantics

States:

Labelled by p

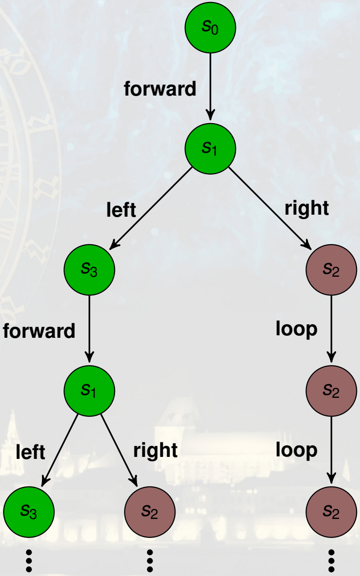
Labelled by q

Properties:

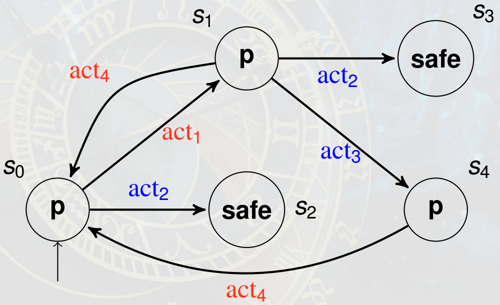
- $s_0 \models E_{\{\textbf{forward},\textbf{left}\}} Gp$
- $s_0 \models E_{\{\textbf{forward},\textbf{right}\}} pUq$

More examples:

- $E_Y G E_Y X \textbf{true}$ – infinite loops detection
- $A_Y G E_Y X \textbf{true}$ – deadlock detection
- $AG_Y(\textbf{p} \wedge EF_Z \textbf{safe})$ – using two action variables Y, Z

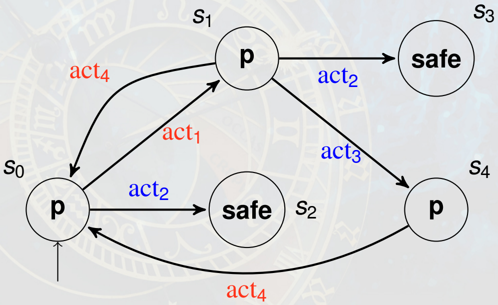


Action synthesis in a nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each Y-reachable state* **p** *holds and* **safe** *is Z-reachable*

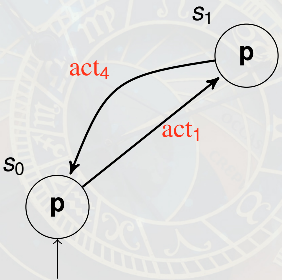
Action synthesis in a nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each Y-reachable state* **p** *holds and* **safe** *is Z-reachable*

$$s_0 \models A_{\{\mathrm{act}_1,\, \mathrm{act}_4\}} G(\mathbf{p} \wedge E_{\{\mathrm{act}_2\}} F\mathbf{safe})$$

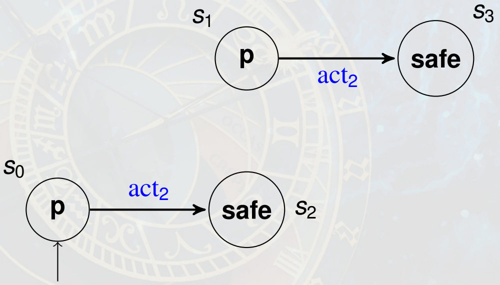
Action synthesis in a nutshell



$A_{\color{red}Y} G(\mathbf{p} \wedge E_{\color{blue}Z} F\,\mathbf{safe})$: *for each* $\color{red}Y$*-reachable state* **p** *holds and* **safe** *is* $\color{blue}Z$*-reachable*

$$s_0 \models A_{\underline{\{\mathrm{act}_1,\, \mathrm{act}_4\}}} G(\mathbf{p} \wedge E_{\{\mathrm{act}_2\}} F\,\mathbf{safe})$$

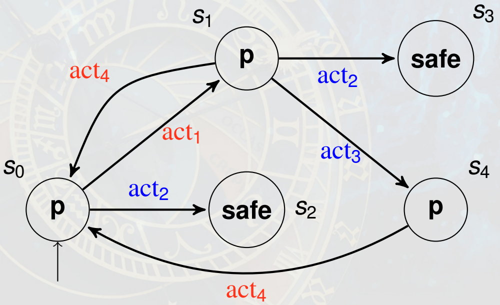
Action synthesis in a nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each Y-reachable state* **p** *holds and* **safe** *is Z-reachable*

$$s_0 \models A_{\{\mathrm{act}_1,\ \mathrm{act}_4\}} G(\mathbf{p} \wedge E_{\underline{\{\mathrm{act}_2\}}} F\mathbf{safe})$$

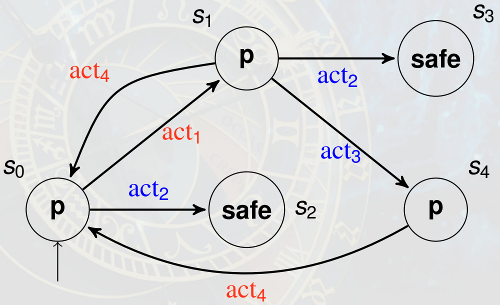
Action synthesis in a nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each Y-reachable state **p** holds and **safe** is Z-reachable*

$$s_0 \not\models A_{\{act_1,\, act_3\}} G(\mathbf{p} \wedge E_{\{act_2\}} F\mathbf{safe})$$

Action synthesis in a nutshell



$A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$: *for each Y-reachable state **p** holds and **safe** is Z-reachable*

Goal: describe all Y, Z s.t.: $s_0 \models A_Y G(\mathbf{p} \wedge E_Z F\mathbf{safe})$

Action synthesis: formal definition

$\mathcal{M} = (\mathcal{S}, s^0, \mathcal{A}, \mathcal{T}, \mathcal{L})$, $\phi \in$ pmARCTL, ActVals := ActSets$^{\text{ActVars}}$

Goal Knapik et al. [2015]

Build $f_\phi : \mathcal{S} \to 2^{\text{ActVals}}$ s.t. for all $s \in \mathcal{S}$:

$$\upsilon \in f_\phi(s) \iff s \models_\upsilon \phi$$

($f_\phi(s)$ contains all valuations that make ϕ hold in s)

THEOREM

The problem of deciding whether $f_\phi(s) \neq \emptyset$ is NP-complete.

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_{\upsilon} E_Y G\phi \iff q \models_{\upsilon} \phi \wedge \left(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\right)$

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_\upsilon E_Y G\phi \iff q \models_\upsilon \phi \wedge \left(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\right)$

  Explanation: ϕ holds along a maximal path starting at q and labelled with a Y–action iff ϕ holds in q and either there is no outgoing Y–action (deadlock) or there is a Y–action s.t. when fired it leads to a state where $E_Y G\phi$ holds

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_\upsilon E_Y G\phi \iff q \models_\upsilon \phi \wedge \left(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\right)$

  Explanation: ϕ holds along a maximal path starting at q and labelled with a Y–action iff ϕ holds in q and either there is no outgoing Y–action (deadlock) or there is a Y–action s.t. when fired it leads to a state where $E_Y G\phi$ holds

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_{\upsilon} E_Y G\phi \iff q \models_{\upsilon} \phi \wedge \left(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\right)$

  Explanation: ϕ holds along a maximal path starting at q and labelled with a Y–action iff ϕ holds in q and either there is no outgoing Y–action (deadlock) or there is a Y–action s.t. when fired it leads to a state where $E_Y G\phi$ holds

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_\upsilon E_Y G\phi \iff q \models_\upsilon \phi \wedge \left(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\right)$

  Explanation: ϕ holds along a maximal path starting at q and labelled with a Y–action iff ϕ holds in q and either there is no outgoing Y–action (deadlock) or there is a Y–action s.t. when fired it leads to a state where $E_Y G\phi$ holds

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_{\upsilon} E_Y G\phi \iff q \models_{\upsilon} \phi \wedge \left(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\right)$

  Explanation: ϕ holds along a maximal path starting at q and labelled with a Y–action iff ϕ holds in q and either there is no outgoing Y–action (deadlock) or there is a Y–action s.t. when fired it leads to a state where $E_Y G\phi$ holds

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_{\upsilon} E_Y G \phi \iff q \models_{\upsilon} \phi \wedge \left(E_Y X E_Y G \phi \vee \neg E_Y X \textbf{true} \right)$

  Explanation: ϕ holds along a maximal path starting at q and labelled with a Y–action iff ϕ holds in q and either there is no outgoing Y–action (deadlock) or there is a Y–action s.t. when fired it leads to a state where $E_Y G \phi$ holds

- $E_Y \phi U \psi \iff \psi \vee (\phi \wedge E_Y X E_Y \phi U \psi)$

(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- $q \models_{\upsilon} E_Y G\phi \iff q \models_{\upsilon} \phi \wedge \left(E_Y X E_Y G\phi \vee \neg E_Y X\mathbf{true}\right)$

  Explanation: ϕ holds along a maximal path starting at q and labelled with a Y–action iff ϕ holds in q and either there is no outgoing Y–action (deadlock) or there is a Y–action s.t. when fired it leads to a state where $E_Y G\phi$ holds

- $E_Y \phi U \psi \iff \psi \vee (\phi \wedge E_Y X E_Y \phi U \psi)$

Implementation:

- easy algorithms: implement $E_Y X$ and compute fixpoints (using BDDs)
- similar to CTL, but deal with indicator functions rather than with sets of states

Conclusion

At this stage, you know about action synthesis

Conclusion

At this stage, you know about action synthesis

Let us see some tool support (next sequence)


ORTVS
AVRORA

SPATULA in a nutshell

First of all…

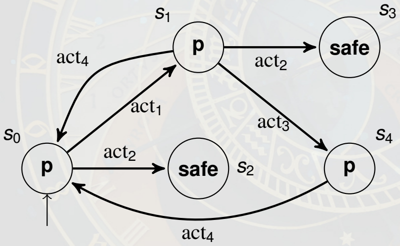
You now know about action synthesis

First of all. . .

You now know about action synthesis

Let us now see some tool support

SPATULA: example



$E_Y F$**safe**

```
module SimpleMTS:

  i = 0;
  for i in (0..5) {
      vert = "s" + i;
      bloom(vert);
   }
  mark_with("s0", "initial");

  mark_with("s0", "p");
  mark_with("s1", "p");
  mark_with("s4", "p");
  mark_with("s2", "safe");
  mark_with("s3", "safe");

  join_with("s0", "s1", "act1");
  join_with("s0", "s2", "act2" );
  join_with("s1", "s0", "act4");
  join_with("s1", "s4", "act3");
  join_with("s1", "s3", "act2");
  join_with("s4", "s0", "act4");

verify:
#EF($Y; (safe));
```

SPATULA: example

module SimpleMTS :

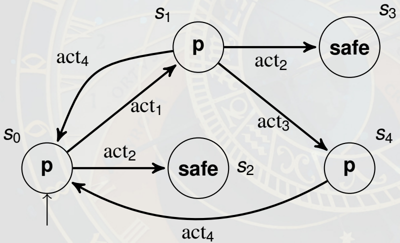
SPATULA: example


```
module SimpleMTS:

  i = 0;
  for i in (0..5) {
      vert = "s" + i;
      bloom(vert);
  }
  mark_with("s0", "initial");
```

SPATULA: example

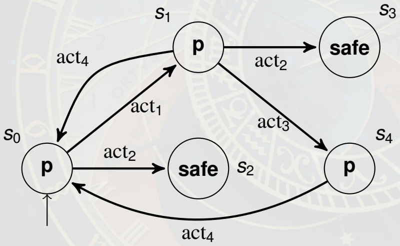

```
module SimpleMTS:

  i = 0;
  for i in (0..5) {
      vert = "s" + i;
      bloom(vert);
   }
  mark_with("s0", "initial");

  mark_with("s0", "p");
  mark_with("s1", "p");
  mark_with("s4", "p");
  mark_with("s2", "safe");
  mark_with("s3", "safe");
```

SPATULA: example




```
module SimpleMTS:

  i = 0;
  for i in (0..5) {
      vert = "s" + i;
      bloom(vert);
    }
  mark_with("s0", "initial");

  mark_with("s0", "p");
  mark_with("s1", "p");
  mark_with("s4", "p");
  mark_with("s2", "safe");
  mark_with("s3", "safe");

  join_with("s0", "s1", "act1");
  join_with("s0", "s2", "act2" );
  join_with("s1", "s0", "act4");
  join_with("s1", "s4", "act3");
  join_with("s1", "s3", "act2");
  join_with("s4", "s0", "act4");
```

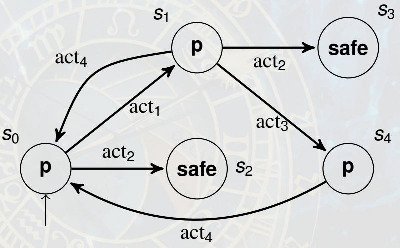
SPATULA: example



$E_Y F$**safe**

```
module SimpleMTS:

  i = 0;
  for i in (0..5) {
      vert = "s" + i;
      bloom(vert);
   }
  mark_with("s0", "initial");

  mark_with("s0", "p");
  mark_with("s1", "p");
  mark_with("s4", "p");
  mark_with("s2", "safe");
  mark_with("s3", "safe");

  join_with("s0", "s1", "act1");
  join_with("s0", "s2", "act2" );
  join_with("s1", "s0", "act4");
  join_with("s1", "s4", "act3");
  join_with("s1", "s3", "act2");
  join_with("s4", "s0", "act4");

verify:
#EF($Y; (safe));
```

SPATULA: example, ct'd



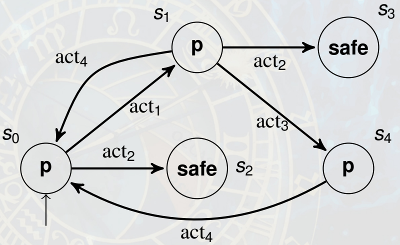
$$E_Y F \textbf{safe}$$

spatula -f SimpleMTS.txt find all Ys...

spatula -m -f SimpleMTS.txt find minimal covering of Ys...

(Easy) question: what is minimal Y here?

SPATULA: example, ct'd



$E_Y F$**safe**

spatula -f SimpleMTS.txt find all Ys...

spatula -m -f SimpleMTS.txt find minimal covering of Ys...

(Easy) question: what is minimal Y here?

A: $s_0 \models E_Y F\textbf{safe} \iff \{act_2\} \subseteq Y$

Conclusion

At this stage:

- you know basics on Petri nets with two kinds of parameters: discrete parameters and timing parameters
- you know basics of Roméo
- you know what Mixed Transition Systems are
- you understand the problem of action synthesis for Parametric Action-Restricted CTL
- you know basics of modelling and synthesis in SPATULA

Conclusion

At this stage:

- you know basics on Petri nets with two kinds of parameters: discrete parameters and timing parameters
- you know basics of Roméo
- you know what Mixed Transition Systems are
- you understand the problem of action synthesis for Parametric Action-Restricted CTL
- you know basics of modelling and synthesis in SPATULA

Let us practice with Roméo and SPATULA


ORTVS
AVRORA

Bibliography

References I

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

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

Bérard, B., Cassez, F., Haddad, S., Lime, D., and Roux, O. H. (2013). The expressive power of time Petri nets. *Theoretical Computer Science*, 474:1–20.

Berthomieu, B. and Diaz, M. (1991). Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Soft. Eng.*, 17(3):259–273.

Berthomieu, B. and Menasche, M. (1983). An enumerative approach for analyzing time Petri nets. In Mason, R. E. A., editor, *Information Processing: proceedings of the IFIP congress 1983*, volume 9 of *IFIP congress series*, pages 41–46. Elsevier Science Publishers, Amsterdam.

David, N., Jard, C., Lime, D., and Roux, O. H. (2015). Discrete parameters in Petri nets. In Devillers, R. and Valmari, A., editors, *The 36th International Conference on Application and Theory of Petri Nets and Concurrency (Petri Nets 2015)*, volume 9115 of *Lecture Notes in Computer Science*, pages 137–156, Brussels, Belgium. Springer.

Gardey, G., Lime, D., Magnin, M., and Roux, O. H. (2005). Roméo: A tool for analyzing time Petri nets. In Etessami, K. and Rajamani, S. K., editors, *17th International Conference on Computer Aided Verification (CAV 2005)*, volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK. Springer-Verlag.

Gardey, G., Roux, O. H., and Roux, O. F. (2006). State space computation and analysis of time Petri nets. *Theory and Practice of Logic Programming (TPLP). Special Issue on Specification Analysis and Verification of Reactive Systems*, 6(3):301–320.

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

Karp, R. M. and Miller, R. E. (1969). Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147 – 195.

References II

Knapik, M., Męski, A., and Penczek, W. (2015). Action synthesis for branching time logic: Theory and applications. *ACM Trans. Embedded Comput. Syst.*

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.

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.

Minsky, M. L. (1967). *Computation: finite and infinite machines*. Prentice-Hall, Inc., NJ, USA.

Traonouez, L.-M., Lime, D., and Roux, O. H. (2009). Parametric model-checking of stopwatch Petri nets. *Journal of Universal Computer Science*, 15(17):3273–3304.


ORTVS
OCCAS