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.
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

Diagram of a Petri net with places $p_1$, $p_2$, $p_3$, and $p_4$, and transitions $t_1$ and $t_2$. The diagram shows the flow of tokens between the places and transitions. There are arcs indicating the flow of tokens, with numbers 2 and 3 indicating the number of tokens transferred. The tokens are represented by black dots in the circles.
Petri nets

![Petri Net Diagram]

- **Places**: $p_1, p_2, p_3, p_4$
- **Transitions**: $t_1, t_2$
- **Arrows**:
  - $t_1$ to $p_3$: 3
  - $p_3$ to $p_2$: 2
  - $p_2$ to $p_1$: 3
  - $p_1$ to $p_4$: 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.
The problem of Coverability

Definition (Coverability)

Given a marking $m$, does there exist a \textit{reachable marking} $m'$ such that $m' \geq m$
The problem of Coverability

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.
Some markings that can be covered:

$$(0, 0, 0, 0) \rightarrow (1, 1, 1, 0) \rightarrow (0, 1, 1, 1)$$

Some markings that cannot be covered:

$$(1, 0, 0, 1) \rightarrow (2, 0, 1, 0) \rightarrow (0, 4, 0, 0)$$
Coverability: Example

Some markings that can be covered:

\[(0, 0, 0, 0) \rightarrow (1, 1, 1, 0) \rightarrow (0, 1, 1, 1)\]

Some markings that \textit{cannot} be covered:

\[(1, 0, 0, 1) \rightarrow (2, 0, 1, 0) \rightarrow (0, 4, 0, 0)\]
Coverability: Example

Some markings that can be covered:

\((0, 0, 0, 0) \rightarrow (1, 1, 1, 0) \rightarrow (0, 1, 1, 1)\)

Some markings that cannot be covered:

\((1, 0, 0, 1) \rightarrow (2, 0, 1, 0) \rightarrow (0, 4, 0, 0)\)
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 $P = \{p_0, \ldots, p_m\}$, a terminal state labelled *halt*
- finite list of instructions $l_1, \ldots, l_s$ among the following list:
  - increment a counter and go to $l_j$
  - if the counter is positive decrement it and go to $l_j$
  - if the counter is null go to $l_i$ else go to $l_j$

Counters are always non-negative.
An Example of 2-Counter Machine

\begin{center}
\begin{tabular}{|l|}
\hline
in $p_1$ : $C_1 := C_1 + 1$; goto $p_2$; \\
in $p_2$ : $C_2 := C_2 + 1$; goto $p_1$; \\
\hline
\end{tabular}
\end{center}

Successive configurations:

$$(p_1, C_1 = 0, C_2 = 0) \rightarrow (p_2, C_1 = 1, C_2 = 0) \rightarrow (p_1, C_1 = 1, C_2 = 1)$$
$$\rightarrow (p_2, C_1 = 2, C_2 = 1) \rightarrow \ldots$$
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 $M$, we build a parametric Petri net $N_M$ encoding it such that:

- $M$ halts iff there exists a parameter valuation $v$ such that place $p_{halt}$ is coverable in $v(N_M)$.
- a counter of $M$ is unbounded iff for all parameter valuations $v$, place $p_{error}$ is coverable in $v(N_M)$. 
Simulation of Instructions

By construction, $m(C_1) + m(\neg C_1) = a$
Decidable Subclasses: A Hierarchy of Parametric PNs

- PPN
- T-PPN
- distinctT-PPN
- preT-PPN
- postT-PPN
- P-PPN
- PN

⊆ ⊆ ⊆ ⊆ ⊆ ⊆ ⊆
Decidable Subclasses: A Hierarchy of Parametric PNs

- $\text{T-PPN}$
- $\text{distinctT-PPN}$
- $\text{preT-PPN}$
- $\text{postT-PPN}$
- $\text{PN}$
- $\text{PPN}$
- $\text{P-PPN}$
From Markings to Output Weights

Replacement of the P parameters by postT parameters
 replacement of the \( P \) parameters by post\( T \) parameters
replacement of the postT parameters by P parameters
replacement of the postT parameters by P parameters
From Output Weights to Markings

replacement of the postT parameters by P parameters

π\(_{t,1}\) → π\(_{t,2}\)

θ\(_{t,p,1}\) → θ\(_{t,p,2}\)

π\(_{t,p,1}\) → π\(_{t,p,2}\)

θ\(_t\)
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

![Diagram showing the hierarchy of PPN subclasses: PN ⊆ T-PPN ⊆ P-PPN ⊆ distinctT-PPN ⊆ preT-PPN ⊆ postT-PPN.](image)

Caption:
- ⊆: is a syntactical subclass of
- ⊑: is a weak-bisimulation subclass of
- ~: is a weak-cosimulation subclass of
- for U-cov: all parameters to 0 is the worst case;
- for E-cov:

replacement of the P parameters by a token injector
to every transition in the original net
Decidable Subclasses: A Hierarchy of Parametric PNs

PPN

T-PPN

distinctT-PPN

preT-PPN

postT-PPN

P-PPN

PN

Caption:

\( \subseteq \) : a syntactical subclass of

\( \sqsubseteq \) : a weak-bisimulation subclass of

\( \sim \) : a weak-cosimulation subclass of
Decidable Subclasses: A Hierarchy of Parametric PNs

Caption:
- $\subseteq$: is a syntactical subclass of
- $\sqsubseteq$: is a weak-bisimulation subclass of
- $\sim$: is a weak-cosimulation subclass of
Decidable Subclasses: A Hierarchy of Parametric PNs

Caption:
- \( \subseteq \) : is a syntactical subclass of
- \( \sim \) : is a weak-bisimulation subclass of
- \( \bowtie \) : is a weak-cosimulation subclass of
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 $\omega$ marking).
Decidable Subclasses: A Hierarchy of Parametric PNs

Caption:
- \( \subseteq \): is a syntactical subclass of
- \( \sqsubseteq \): is a weak-bisimulation subclass of
- \( \sim \): is a weak-cosimulation subclass of
Decidable Subclasses: A Hierarchy of Parametric PNs

Caption:

\[ \subseteq \] : is a syntactical subclass of

\[ \leq \] : is a weak-bisimulation subclass of

\[ \sim \] : is a weak-cosimulation subclass of

\[ \preceq \] : is a weak-modal subclass of

\[ \succeq \] : is a strong-bisimulation subclass of

\[ \simeq \] : is a strong-cosimulation subclass of

\[ \preceq \] : is a preorder subclass of

\[ \succeq \] : is a postorder subclass of
Decidable Subclasses: A Hierarchy of Parametric PNs

Caption:
- \( \subseteq \): is a syntactical subclass of
- \( \sqsubseteq \): is a weak-bisimulation subclass of
- \( \sim \): is a weak-cosimulation subclass of
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.
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
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)

\[ p_0 \rightarrow_{t_0} a, b \rightarrow_{t_1} 2, +\infty \rightarrow p_2 \]
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 also regroup states obtained with the same discrete transition sequence in a pair \( (l, Z) \) where \( Z \) is a zone;

But states record time to firing instead of time elapsed;

Initially:
\[
\begin{align*}
&1 \leq t_0 \leq 4 \\
&2 \leq t_1 \leq 3
\end{align*}
\]

Fire \( t_0 \):
\[
\begin{align*}
&1 \leq t_0 \leq 4 \\
&2 \leq t_1 \leq 3 \\
&t_0 \leq t_1
\end{align*}
\]

Disabled (incl. \( t_0 \)):
\[
\begin{align*}
&0 \leq t_1' \leq 2
\end{align*}
\]

Newly enabled:
\[
\begin{align*}
&1 \leq t_0 \leq 4 \\
&0 \leq t_1 \leq 2
\end{align*}
\]
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:
\[
\begin{align*}
 a &\leq t_0 &\leq 4 \\
 2 &\leq t_1 &\leq b
\end{align*}
\]

Fire \( t_0 \):
\[
\begin{align*}
 a &\leq t_0 &\leq 4 \\
 2 &\leq t_1 &\leq b \\
 t_0 &\leq t_1 \\
 (a &\leq b)
\end{align*}
\]

New times to fire:
\[
\begin{align*}
 a &\leq t_0 &\leq 4 \\
 2 &\leq t_1' + t_0 &\leq b \\
 t_0 &\leq t_1' + t_0
\end{align*}
\]

Disabled (incl. \( t_0 \)):
\[
\begin{align*}
 0 &\leq t_1' &\leq b - a
\end{align*}
\]

Newly enabled:
\[
\begin{align*}
 a &\leq t_0 &\leq 4 \\
 0 &\leq t_1 &\leq b - a
\end{align*}
\]
Synthesis for Parametric TPNs

- EFSynth works the same with parametric state classes:

\[
\text{EF}_G(S, M) = \begin{cases} 
Z \downarrow_P & \text{if } l \in G \\
\emptyset & \text{if } S \in M \\
\bigcup_{t \in T} \text{EF}_G(S', M \cup \{S\}) & \text{otherwise.}
\end{cases}
\]

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

\[
\text{AF}_G(S, M) = \begin{cases} 
Z \downarrow_P & \text{if } l \in G \\
\emptyset & \text{if } S \in M \\
\left(\bigcap_{t \in T} \left(\text{AF}_G(S', M \cup \{S\}) \cup (Q^P \setminus S' \downarrow_P)\right)\right) & \text{otherwise}
\end{cases}
\]

- \( S = (l, Z) \);
- \( G \) a set of markings to reach;
- \( M \) is a list of visited state classes;
- \( \text{Next}(S, t) \) 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 $a$. 

![Diagram](Image)
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);
  - \( \text{PSPACE} = \text{NPSPACE} \) (Savitch’s theorem).

- Synthesis can be done **symbolically**, using integer hulls:

\[
y
\]
IEF computes polyhedra containing exactly the “good” integer parameter valuations:

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

- It is guaranteed to terminate when the parameters are bounded;
- AF can be modified similarly.
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, $Z_2 = \{1 \leq x \land 1 \leq 2a\}$, so $\mathrm{IH}(C_2) = \{1 \leq x \land 1 \leq a\}$
- 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 \( \text{IH}(Z) \subseteq Z \);

Solution: use integer hulls only for convergence André et al. [2015]:

\[
\text{RIEF}_G(S, M) = \begin{cases} 
Z_{\downarrow P} & \text{if } l \in G \\
\emptyset & \text{if } \text{IH}(S) \in M \\
\bigcup_{t \in T} \left( \text{EF}_G(S', M \cup \{\text{IH}(S)\}) \right) & \text{otherwise} 
\end{cases}
\]

\[
\text{RIAF}_G(S, M) = \begin{cases} 
Z_{\downarrow P} & \text{if } l \in G \\
\emptyset & \text{if } \text{IH}(S) \in M \\
\left( \bigcap_{t \in T} \left( \text{AF}_G(S', M \cup \{\text{IH}(S)\}) \cup (\mathbb{Q}^P \setminus S'_{\downarrow P}) \right) \right) & \text{otherwise}
\end{cases}
\]

Gives a “dense” underapproximation containing at least all integer valuations.
Dense Integer-preserving Underapproximations

- 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;
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;

**Roméo** is a tool that supports parametric TPNs (next sequence)
Roméo in a nutshell
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;
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
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/
At this stage, you know about:

- Petri nets with discrete parameters
- Time Petri nets with timing parameters
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)
Action Synthesis
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
Mixed Transition Systems (MTS)

MTS: *Kripke structures with action-labelled transitions*

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

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

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

\[ A \subseteq \mathcal{A} \text{ – a set of allowed actions} \]

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

E.g., \( \Pi(\{\text{act}_1, \text{act}_2, \text{act}_4\}, s_0) = \)
\[ \{(s_0 \text{act}_1 s_1 \text{act}_4)\omega + (s_0 \text{act}_1 s_1 \text{act}_4)^* s_0 \text{act}_1 s_1 \text{act}_2 s_3 + (s_0 \text{act}_1 s_1 \text{act}_4)^* s_0 \text{act}_2 s_2\} \]
Parametric ARCTL

pmARCTL: CTL with actions/variable subscripts

ActSets – non-empty subsets of \( \mathcal{A} \)

ActVars – the action variables

pmARCTL: the formulae \( \phi \) generated by the BNF grammar:

\[
\phi ::= p \mid \neg \phi \mid \phi \lor \phi \mid E_\alpha X \phi \mid E_\alpha G \phi \mid E_\alpha (\phi U \phi)
\]

\( p \in \mathcal{P}V, \alpha \in \text{ActSets} \cup \text{ActVars} \)

- \( E_\alpha \) – “there exists a maximal path over \( \alpha \)”
- \( X, G, U \) – \text{next}, \text{Globally}, \text{Until}
Parametric ARCTL

pmARCTL: CTL with actions/variable subscripts

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

pmARCTL: the formulae $\phi$ generated by the BNF grammar:

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

$p \in \mathcal{P}\mathcal{V}$, $\alpha \in \text{ActSets} \cup \text{ActVars}$

- $E_\alpha$ – “there exists a maximal path over $\alpha$”
- $X, G, U$ – $\text{neXt}, \text{Globally}, \text{Until}$
- (derived) $A_\alpha$ – “for each maximal path over $\alpha$”
- (derived) $F$ – “in the future”
Parametric ARCTL: semantics

States:
- Labelled by $p$
- Labelled by $q$

Properties:
- $s_0 | s_0 = E \{ \text{forward}, \text{left} \}$
- $s_0 | s_0 = E \{ \text{forward}, \text{right} \}$
- More examples:
  - $E Y GE Y X$ true – infinite loops
  - $A Y GE Y X$ true – deadlock detection
  - $A G Y (p \land EF Z)$ safe – using two action variables $Y, Z$

Diagram:
- States $s_0, s_1, s_2, s_3, ...$
- Labels:
  - Forward
  - Left
  - Right
  - Loop
States:
- Labelled by $p$
- Labelled by $q$

Properties:
- $s_0 \models E_{\{\text{forward, left}\}} Gp$
- More examples:
  - $E_{Y} G E_{X}$: true – infinite loops
  - $A_{Y} G E_{X}$: true – deadlock detection
  - $AG_{Y} (p \land EF_{Z} \text{safe})$ – using two action variables $Y$, $Z$
Parametric ARCTL: semantics

States:
- Labelled by $p$
- Labelled by $q$

Properties:
- $s_0 \models E_{\text{forward, left}} G p$
- $s_0 \models E_{\text{forward, right}} pU q$
Parametric ARCTL: semantics

States:
- Labelled by $p$
- Labelled by $q$

Properties:
- $s_0 \models E\{\text{forward, left}\} Gp$
- $s_0 \models E\{\text{forward, right}\} pUq$

More examples:
- $E_y G E_y X \text{true}$ – infinite loops detection
- $A_y G E_y X \text{true}$ – deadlock detection
- $AG_y (p \land EF_z \text{safe})$ – using two action variables $Y$, $Z$
Action synthesis in a nutshell

\[ A_Y \Box (p \land E_Z \Diamond \text{safe}) \]: for each Y-reachable state \( p \) holds and \text{safe} is Z-reachable
Action synthesis in a nutshell

\[ A_Y G(p \land E_Z F_{\text{safe}}) \text{: for each } Y\text{-reachable state } p \text{ holds and safe is } Z\text{-reachable} \]

\[ s_0 \models A \{\text{act}_1, \text{act}_4\} G(p \land E_{\text{act}_2} F_{\text{safe}}) \]
$A_Y G(p \land E_Z F_{safe})$: for each $Y$-reachable state $p$ holds and $safe$ is $Z$-reachable

$$s_0 \models A_{\{act_1, act_4\}} G(p \land E_{\{act_2\}} F_{safe})$$
Action synthesis in a nutshell

\[ A_Y G(p \land E_Z F_{\text{safe}}) : \text{for each } Y\text{-reachable state } p \text{ holds and } \text{safe} \text{ is } Z\text{-reachable} \]

\[ s_0 \models A_{\{\text{act}_1, \text{act}_4\}} G(p \land E_{\{\text{act}_2\}} F_{\text{safe}}) \]
**Action synthesis in a nutshell**

$$\forall Y \exists Z \left( p \land E_{\{act_2\}} F_{\text{safe}} \right) : \text{for each } Y\text{-reachable state } p \text{ holds and safe is } Z\text{-reachable}$$

$$s_0 \not\models A_{\{act_1, act_3\}} G(p \land E_{\{act_2\}} F_{\text{safe}})$$
Action synthesis in a nutshell

\[
A_Y G(p \land E_Z F_{\text{safe}}): \text{for each } Y \text{-reachable state } p \text{ holds and safe is } Z \text{-reachable}
\]

**Goal:** describe all \( Y, Z \) s.t.: \( s_0 \models A_Y G(p \land E_Z F_{\text{safe}}) \)
Action synthesis: formal definition

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

**Goal** Knapik et al. [2015]

Build \( f_\phi : S \rightarrow 2^{\text{ActVals}} \) s.t. for all \( s \in S \):

\[ \forall \nu \in f_\phi(s) \iff s \models_\nu \phi \]

\((f_\phi(s) \text{ contains all valuations that make } \phi \text{ hold in } s)\)

**THEOREM**

The problem of deciding whether \( f_\phi(s) \neq \emptyset \) is NP-complete.
(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

\[ q \models_{\nu} E_Y G \phi \iff q \models_{\nu} \phi \land \left( E_Y X E_Y G \phi \lor \neg E_Y X \text{true} \right) \]
Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

\[ q \models_{v} E_{Y}G\phi \iff q \models_{v} \phi \land (E_{Y}XE_{Y}G\phi \lor \neg E_{Y}Xtrue) \]

Explanation: \( \phi \) holds along a maximal path starting at \( q \) and labelled with a \( Y \)-action iff \( \phi \) 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:

\[ q \models \nu E Y G \phi \iff q \models \phi \land (E Y X E Y G \phi \lor \neg E Y X \text{true}) \]

Explanation: \( \phi \) holds along a maximal path starting at \( q \) and labelled with a \( Y \)–action iff \( \phi \) 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:

\[ q \models_u E_Y G \phi \iff q \models_u \phi \land (E_Y X E_Y G \phi \lor \neg E_Y X \text{true}) \]

Explanation: \( \phi \) holds along a maximal path starting at \( q \) and labelled with a \( Y \)-action iff \( \phi \) 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:

\[ q \models _\upsilon E_Y G \phi \iff q \models _\upsilon \phi \land (E_Y X E_Y G \phi \lor \neg E_Y X \text{true}) \]

Explanation: \( \phi \) holds along a maximal path starting at \( q \) and labelled with a \( Y \)–action iff \( \phi \) 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:

\[ q \models \nu E Y G \phi \iff q \models \nu \phi \land (E Y X E Y G \phi \lor \neg E Y X \text{true}) \]

Explanation: \( \phi \) holds along a maximal path starting at \( q \) and labelled with a \( Y \)-action iff \( \phi \) 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:

\[ q \models \phi \iff q \models \phi \land (E Y X E Y G \phi \lor \neg E Y X \text{true}) \]

Explanation: \( \phi \) holds along a maximal path starting at \( q \) and labelled with a \( Y \)-action iff \( \phi \) 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

\[ E Y \phi U \psi \iff \psi \lor (\phi \land E Y X E Y \phi U \psi) \]
(Some) fixed-points for pmARCTL

Recursive equivalences in pmARCTL:

- \[ q \models_{\nu} E_Y G \phi \iff q \models_{\nu} \phi \land (E_Y X E_Y G \phi \lor \neg E_Y X \text{true}) \]

Explanation: \( \phi \) holds along a maximal path starting at \( q \) and labelled with a \( Y \)-action iff \( \phi \) 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.

- \[ E_Y \phi U \psi \iff \psi \lor (\phi \land E_Y X E_Y \phi U \psi) \]

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
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)
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
module SimpleMTS:

\( i = 0; \)
\[
\text{for } i \text{ in } (0..5) \{ \\
\quad \text{vert} = "s" + i; \\
\quad \text{bloom(vert);} \\
\}
\]
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");

\[
E_Y F_{\text{safe}}
\]

verify:
\[
\#EF(Y; (\text{safe}));
\]
module SimpleMTS:
module SimpleMTS:

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

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

(Easy) question: what is minimal Y here?

\[ E_Y F_{safe} \]
**SPATULA: example, ct’d**

```
(spatula -f SimpleMTS.txt) find all Ys...
(spatula -m -f SimpleMTS.txt) find minimal covering of Ys...
```

(Easy) question: what is minimal Y here?

\[ A: s_0 \models E_Y F\text{safe} \iff \{\text{act}_2\} \subseteq Y \]
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


