

Formally Tracing Executions

From an Analysis Tool Back to a Domain Specific Modeling
Language's Operational Semantics

Vlad Rusu and Laure Gonnord and Benoît Combemale

INRIA Lille/LIFL(Univ. Lille)

<http://laure.gonnord.org/pro/>
Laure.Gonnord@lifl.fr



Context - 1

- Model-based development.
 - Design of new specialised languages : **Domain-Specific**.
 - The syntax : meta-models (graphical grammars) :
“Modeling”
 - The (operational) semantics : methods of these objects.
- ▶ **Domain-Specific Modeling Languages** (DSML).

Context - 2

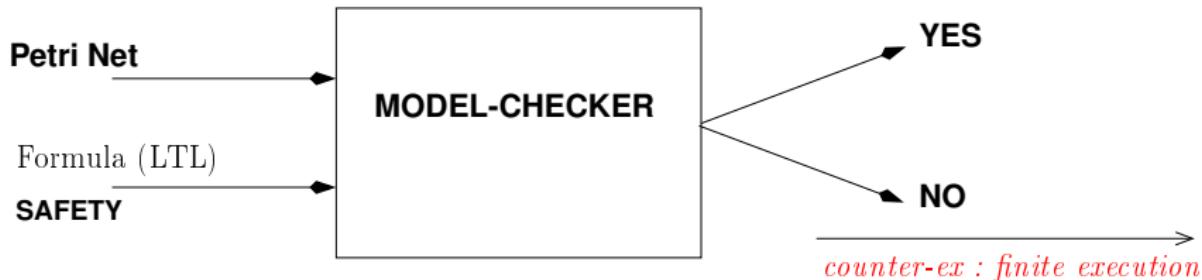
How do we deal with these DSMLs ?

- Some of them are just modeling languages.
 - Compilation through **model transformation** :
 - To other DSMLs.
 - To other “standard” languages : C, Lustre, ...
 - **Verification of** execution properties :
 - compilation into “classical” formal objects : Petri Nets, automata.
 - use an associated decision tool.
- The compilation produces a model which **semantics is well-defined**.

At the beginning - an example

[Combemale et al, Journal of Software, 2009]

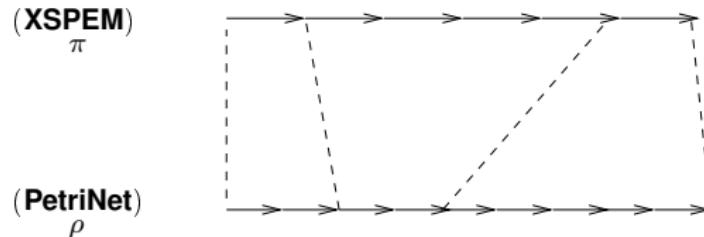
- A model transformation from **xSPEM** to **Petri Net** ;
- Both languages are designed in terms of metamodels ;
- After transformation, the resulted Petri Net is analysed.
▶ **Model Checking**



ajouter animation avec une flèche qui remonte.

General Result

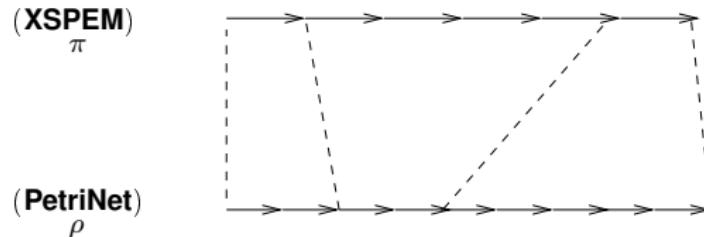
When Φ a **bisimulation** :



the result of the model checker **holds**.

General Result

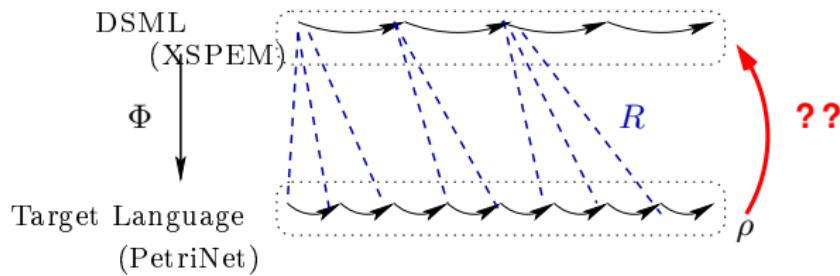
When Φ a **bisimulation** :



the result of the model checker **holds**.

- ▶ **But** how to deal with counter examples ?

The Problem



Given :

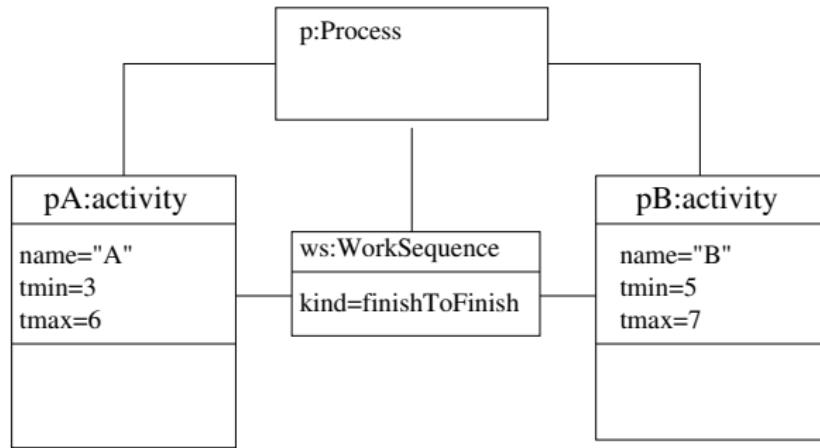
- A **relation** R between states ($|R| \leq |\Phi|$).
- An execution ρ of **the target language**,

how expressing an execution in terms of **the input language** ?

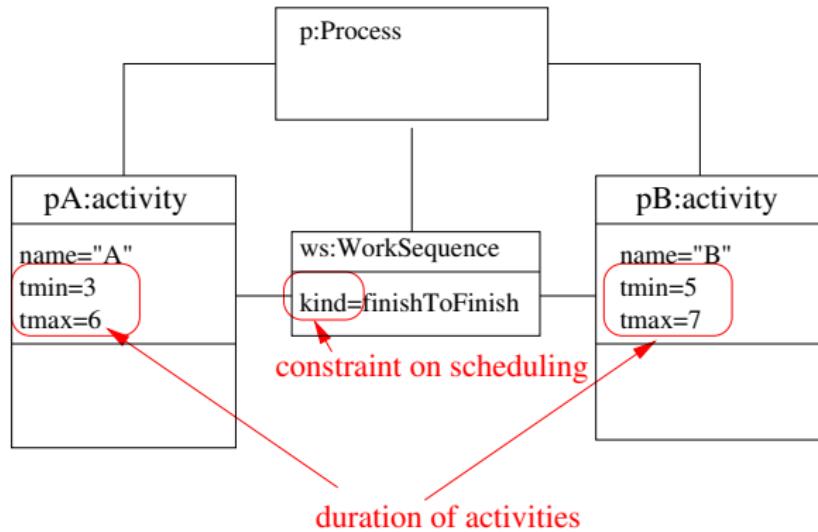
► The **backward tracing problem**

- 1 Running Example
- 2 Formalisation and algorithm
- 3 Implementation and example

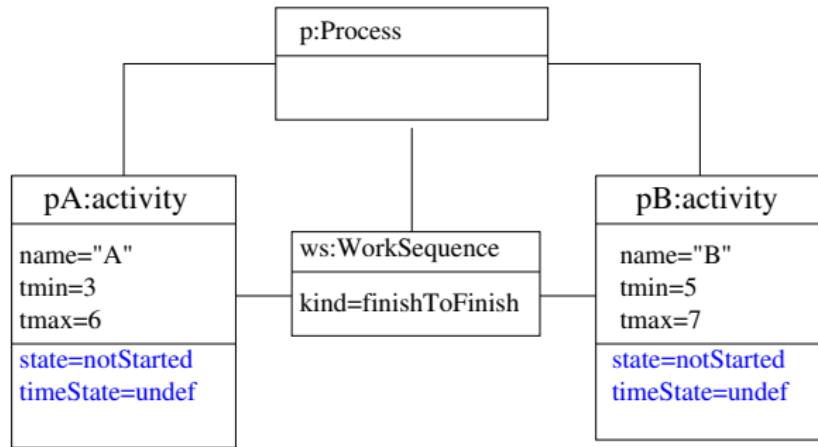
An example for Φ - Input Model (1/2)



An example for Φ - Input Model (1/2)

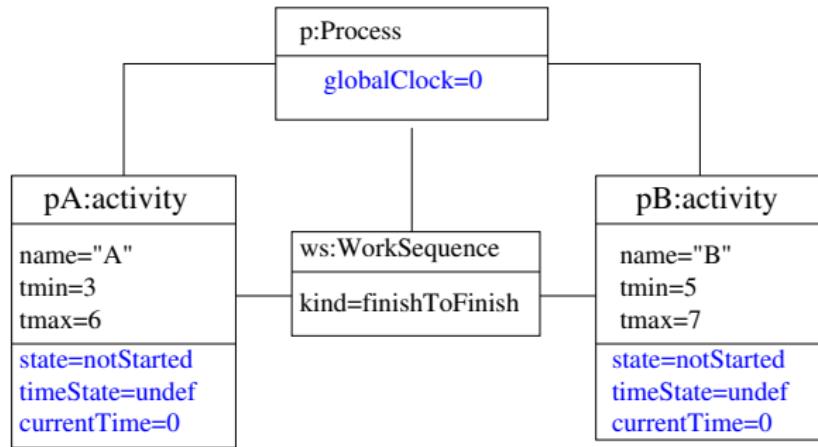


An example for Φ - Input Model (1/2)



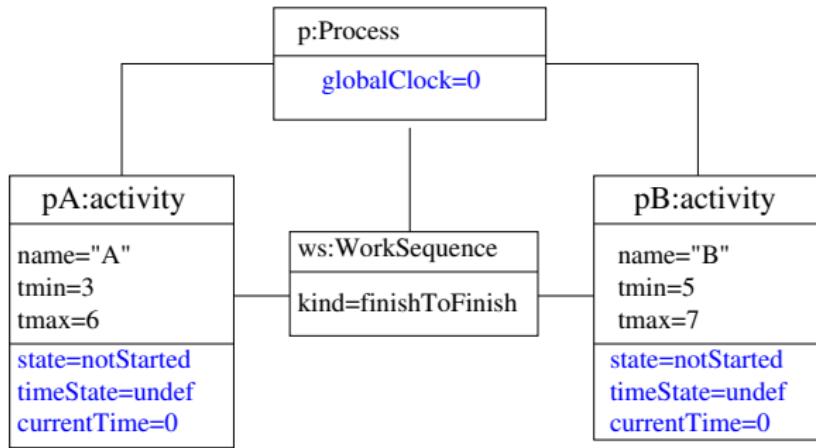
$state \in \{notStarted, started, finished\}$
 $timeState \in \{undef, ok, tooLate, tooEarly\}$

An example for Φ - Input Model (1/2)



$state \in \{notStarted, started, finished\}$
 $timeState \in \{undef, ok, tooLate, tooEarly\}$

An example for Φ - Input Model (1/2)



$state \in \{notStarted, started, finished\}$
 $timeState \in \{undef, ok, tooLate, tooEarly\}$

evolving vars ▶ **operational semantics**

An example for Φ - Input Model (2/2)

Its operational semantics is a set of rules operating on *states* :

- The **global state** is

$$\{globalTime\} \times \prod_{a \in A} (state_a, timeState_a, currentTime_a).$$

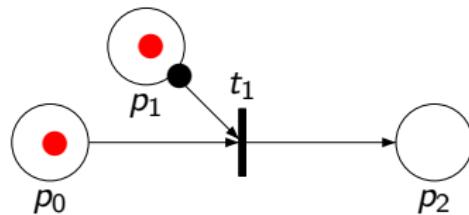
An example for Φ - Input Model (2/2)

Its operational semantics is a set of rules operating on *states* :

- The **global state** is
$$\{globalTime\} \times \prod_{a \in A} (state_a, timeState_a, currentTime_a).$$
- Given an activity a , the (non deterministic) evolution of its state is defined by **a set of rules** :
 - Evolution of *GlobalClock* and all *currentTime*.
 - Evolution of $state_a$ (notstarted,started,finish) : must respect the static constraints $tmin, tmax$.
 - Evolution of $timeState_a$ (ok,tooLate,tooEarly), according to the real execution date of a .
- ▶ Implemented in **Process.run()** and **Process.incTime()** (they appear in the metamodel).

An example for Φ - Output Model

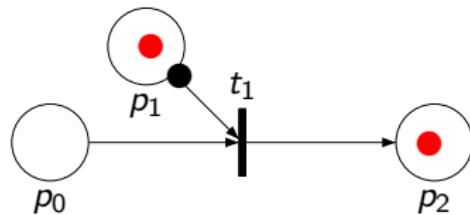
Classic PN



- ▶ A PrTPN **state** is $(marking, timestamp)$

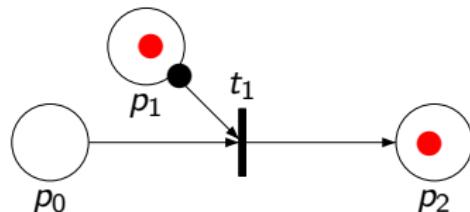
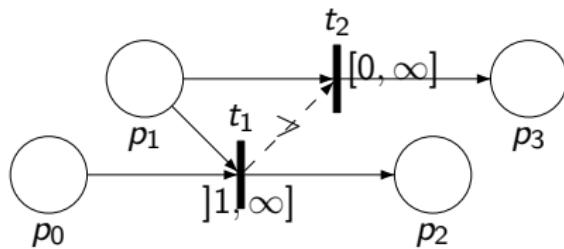
An example for Φ - Output Model

Classic PN



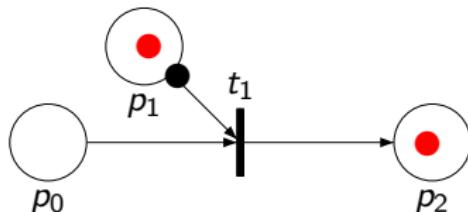
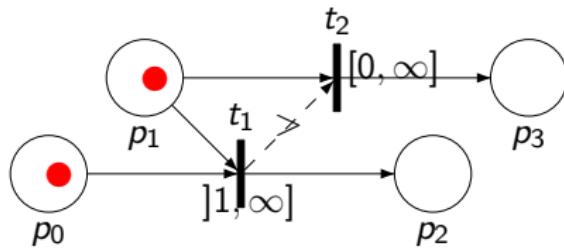
- ▶ A PrTPN **state** is $(marking, timestamp)$

An example for Φ - Output Model

Classic PN**Prioritized TPN**

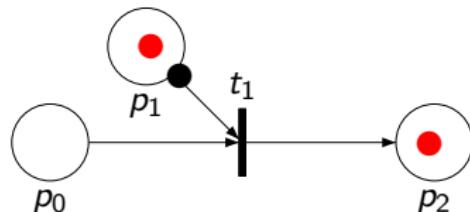
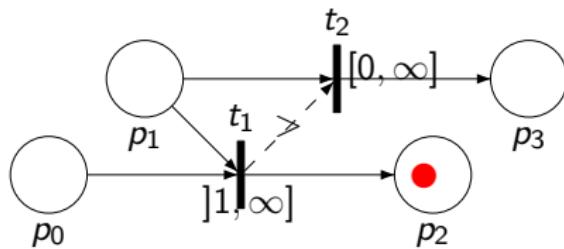
- ▶ A PrTPN **state** is $(marking, timestamp)$

An example for Φ - Output Model

Classic PN**Prioritized TPN**

- ▶ A PrTPN **state** is $(marking, timestamp)$

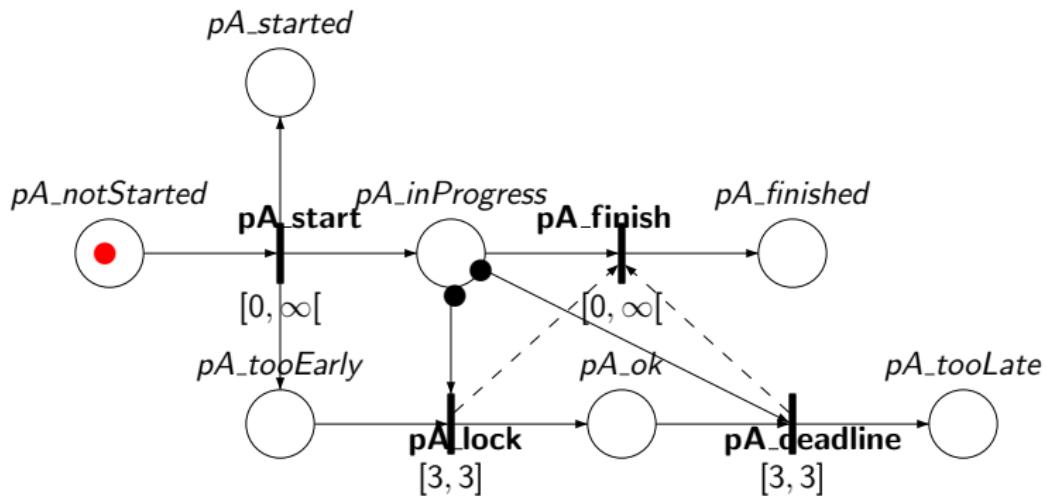
An example for Φ - Output Model

Classic PN**Prioritized TPN**

- ▶ A PrTPN **state** is $(marking, timestamp)$

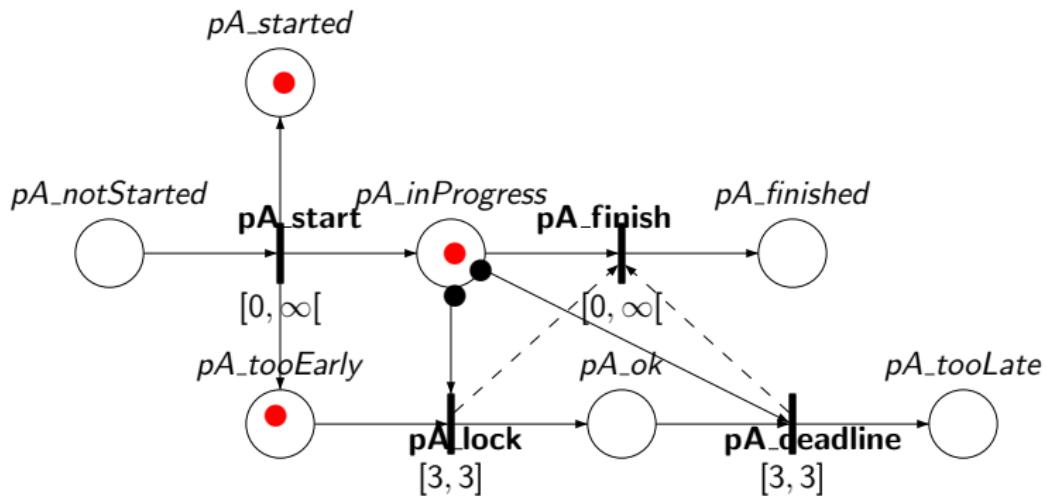
An example for Φ - Result

A Petri Net for each activity :



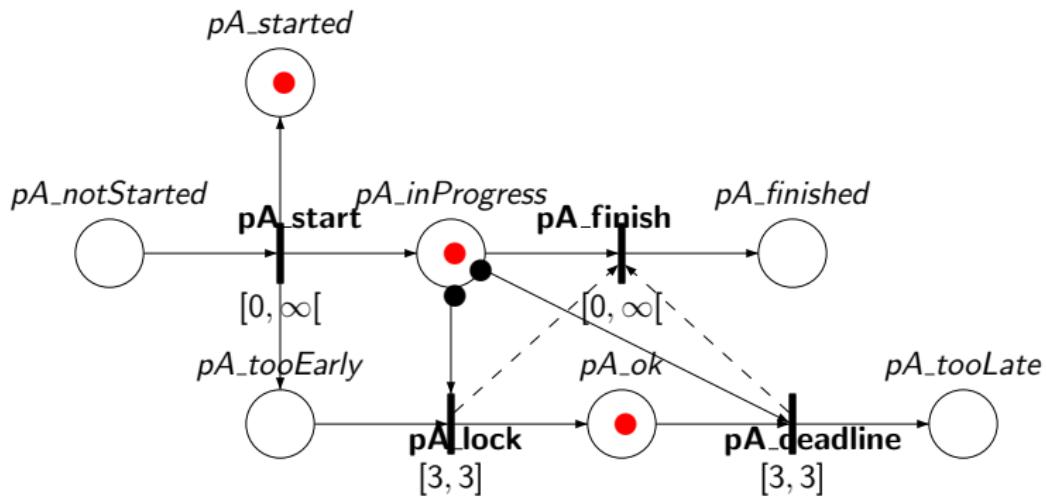
An example for Φ - Result

A Petri Net for each activity :



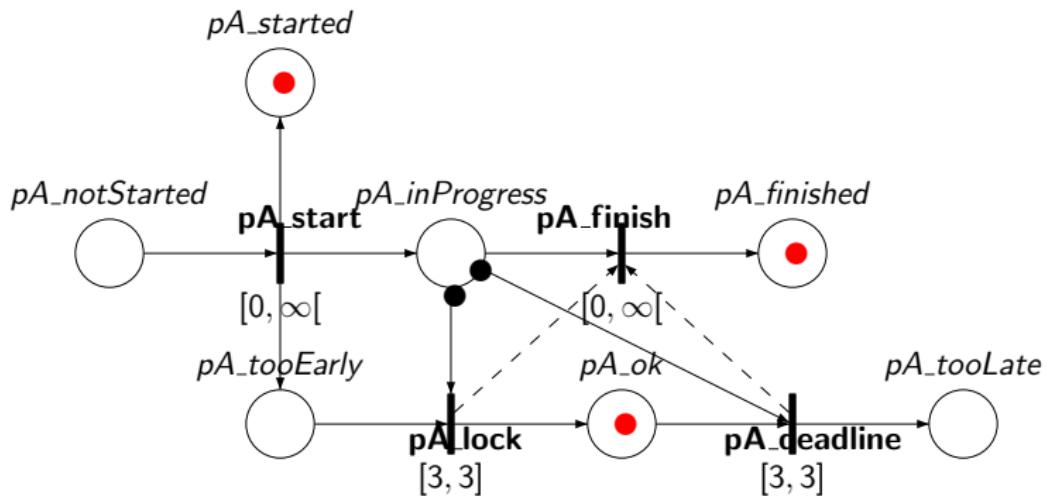
An example for Φ - Result

A Petri Net for each activity :



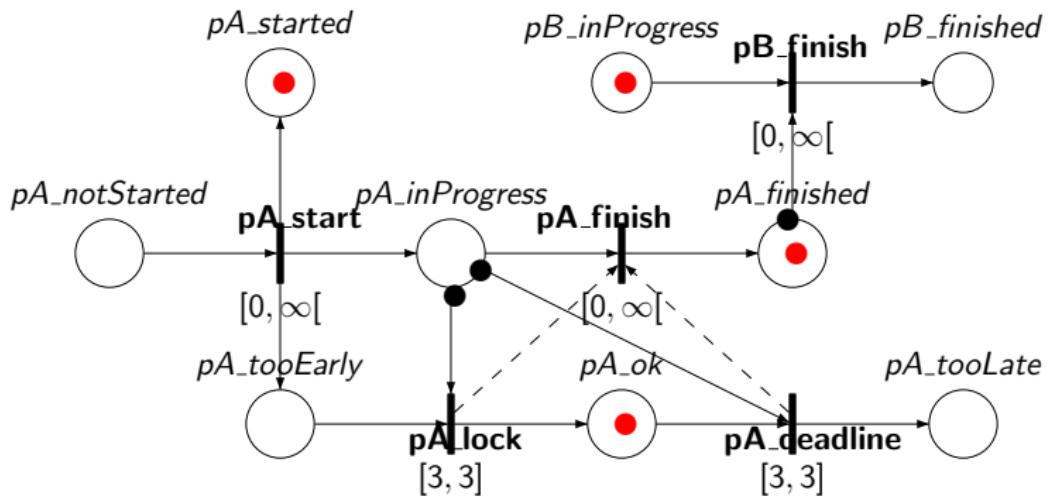
An example for Φ - Result

A Petri Net for each activity :



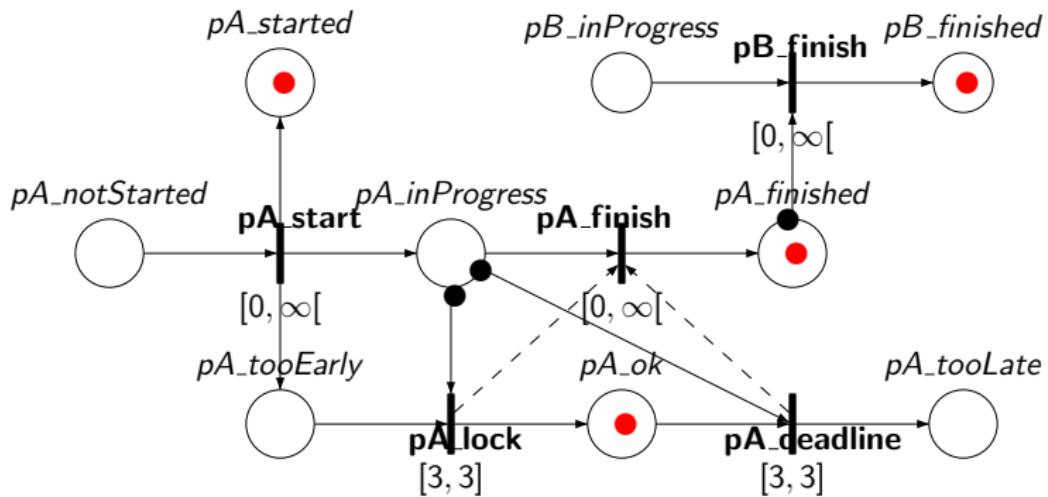
An example for Φ - Result

A Petri Net for each activity :



An example for Φ - Result

A Petri Net for each activity :



Relationship between states / executions

- A PN execution is a sequence of states :
 $(marking, timestamp)$.
- A XSPEM execution is a sequence of states
 $\{globalTime\} \times \prod_{a \in A} (state_a, timeState_a, currentTime_a)$.

Relationship between states / executions

- A PN execution is a sequence of states :
(marking, timestamp).
 - A XSPEM execution is a sequence of states
 $\{globalTime\} \times \prod_{a \in A} (state_a, timeState_a, currentTime_a)$.
- Φ induces a **relation R** between states :

	XSPEM	Petri Net
	globalTime	time stamp
for all activity a example $a = pA$	$state_a = i$ $state_{pA} = started$	token in place a_i a token in place $pAStarted$
if a is finished example	$timeState_a = j$ $timeState_{pA} = ok$	token in place a_j (status) token in place $pAok$

An example for Φ - Analysis

Analysis of the output PN, with a LTL formula :

$$\square \neg (pA_finished \wedge pA_ok \wedge pB_finished \wedge pB_ok)$$

An example for Φ - Analysis

Analysis of the output PN, with a LTL formula :

$$\square \neg (pA_finished \wedge pA_ok \wedge pB_finished \wedge pB_ok)$$

TINA gives an execution ρ where both activities end in due time :

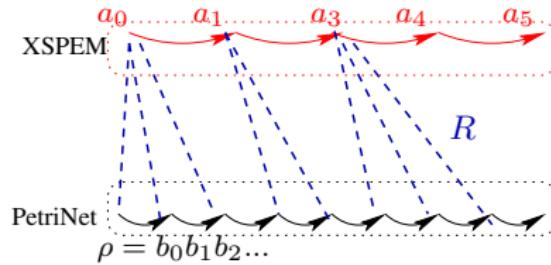
$(m_0, 0), pA_start, (m_1, 0), pA_lock, (m_2, 3), pA_finish,$
 $(m_3, 3), pB_start, (m_4, 3), pB_lock, (m_5, 8), pB_finish,$
 $(m_6, 8).$

- A begins at $t = 0$ and finishes at $t = 3$.
- A begins at $t = 3$ and finishes at $t = 8$.

Our algorithm in summary

Given :

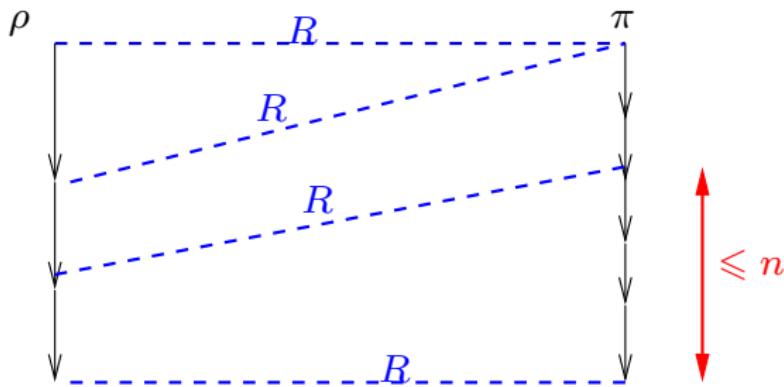
- **Input model** : syntax + implementation of the semantics.
 - **Output model** : only the syntax.
 - 3 other parameters : R (not necessarily a **simulation**), and n , and a ρ (execution of PN).
- our algorithm produces :



- 1 Running Example
- 2 Formalisation and algorithm
- 3 Implementation and example

R-matching

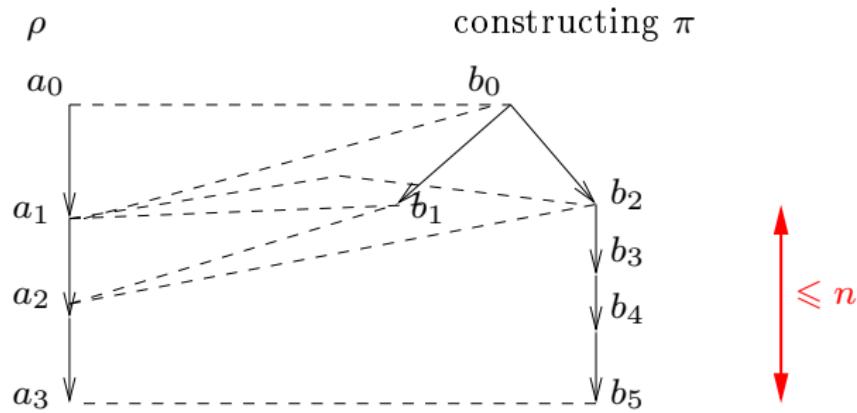
We consider **transition systems** with finite branching. Let R be a relation between states.



- ▶ The execution $\pi(n, R)$ matches the execution ρ .

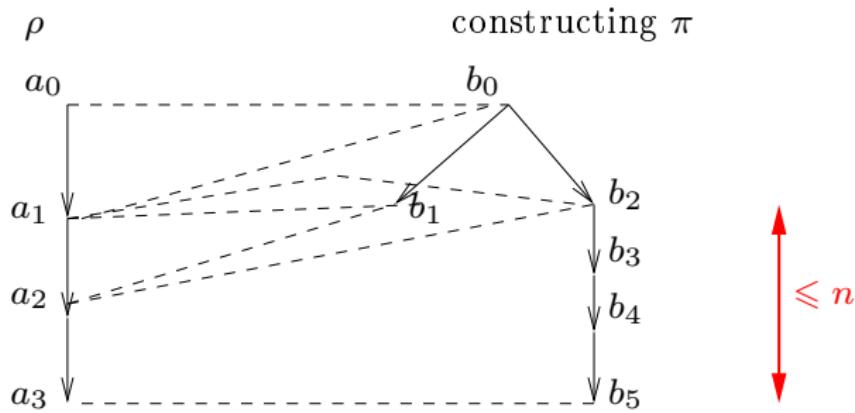
Algorithm

Trying to match ρ :



Algorithm

Trying to match ρ :



If the algorithms fails, R is **not** a simulation.

Algorithm - Result

Theorem

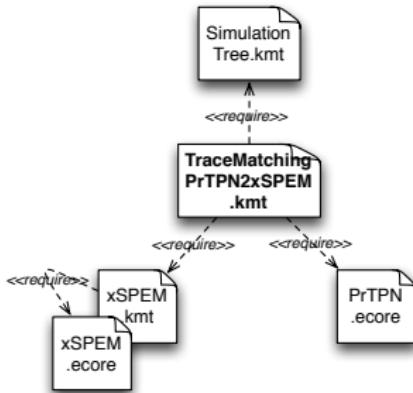
Given ρ , the algorithm produces an execution that matches the longest prefix that be (n, R) matched.

- ▶ Proof in the paper.

- 1 Running Example
- 2 Formalisation and algorithm
- 3 Implementation and example

Implementation

- A generic implementation in Kermeta (Triskell).
- Instantiation on XSPEM to PetriNet.

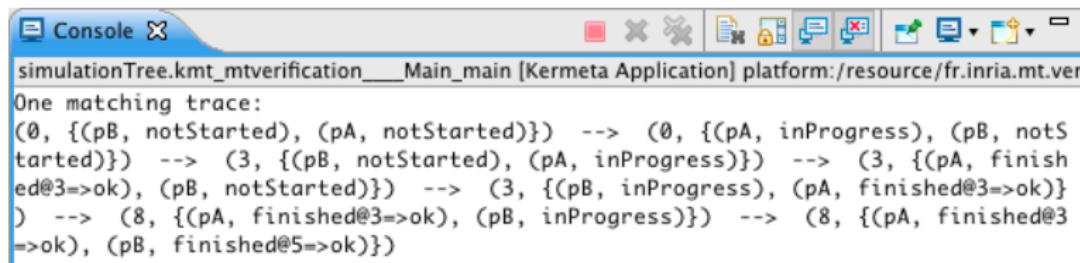


Implementation - usage

The user provides :

- the input model + semantics (+metamodel)
- an execution of the output model (here, given by TINA)
- a relation R (method), a bound n (here, 3).

and then :



A screenshot of a computer interface showing a "Console" window. The window title is "Console". The content area displays a Kermeta application trace. The text starts with "simulationTree.kmt_mtverification___Main_main [Kermeta Application] platform:/resource/fr.inria.mt.ver". It then shows a "One matching trace:" followed by a sequence of state transitions. Each transition is represented by a pair of sets of states separated by a double-headed arrow. The first set contains a state labeled 0 and a condition involving pB (notStarted) and pA (notStarted). The second set contains a state labeled 0 and a condition involving pA (inProgress) and pB (notStarted). This pattern repeats three times, with the final transition being from state 8 to state 8, both with the condition pA (finished@3=>ok) and pB (finished@5=>ok).

```
simulationTree.kmt_mtverification___Main_main [Kermeta Application] platform:/resource/fr.inria.mt.ver
One matching trace:
(0, {(pB, notStarted), (pA, notStarted)}) --> (0, {(pA, inProgress), (pB, notStarted)})
(0, {(pA, inProgress), (pB, notStarted)}) --> (3, {(pB, notStarted), (pA, inProgress)})
(3, {(pB, notStarted), (pA, inProgress)}) --> (3, {(pA, finished@3=>ok), (pB, notStarted)})
(3, {(pA, finished@3=>ok), (pB, notStarted)}) --> (3, {(pB, inProgress), (pA, finished@3=>ok)})
(3, {(pB, inProgress), (pA, finished@3=>ok)}) --> (8, {(pA, finished@3=>ok), (pB, inProgress)})
(8, {(pA, finished@3=>ok), (pB, inProgress)}) --> (8, {(pA, finished@3=>ok), (pB, finished@5=>ok)})
```

Perspectives

- Implementation : genericity
- Algorithmic : **sharing** common parts in a model execution
- Theory : Relationship between Φ and R

Questions ?