Diagrams of Mobile Interactions

Frédéric Peschanski
University Pierre & Marie Curie - Paris 6
LIP6 - APR

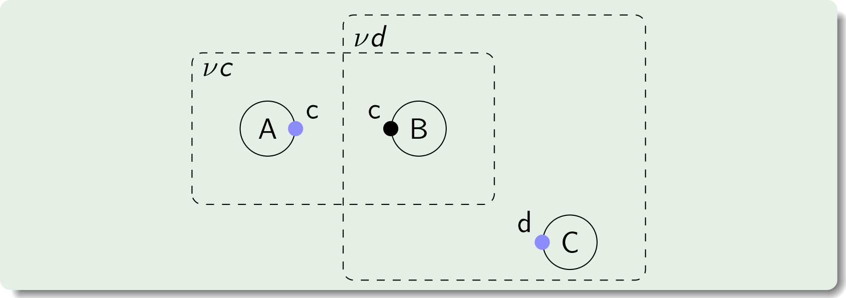
joint work with Hanna Klaudel (IBisc/Evry)
and Raymond Devillers (Univ. Libre de Bruxelles)

Mefosyloma–seminar - Nov. 5th 2010 - Paris

Plan

Plan

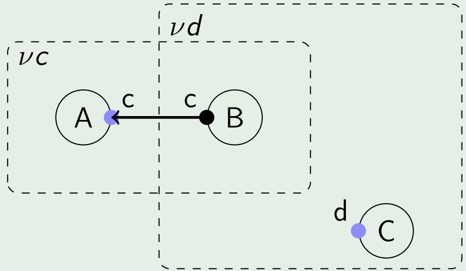
The π-calculus in a nutshell



$$\nu(c)[A(c,m) \mid \nu(d)[B(c,d) \mid C(d)]]$$

$$\left[\begin{array}{lcl} A(c,m) & = & c(x).\overline{x}\langle m\rangle.P \\ B(c,d) & = & \overline{c}\langle d\rangle.Q \\ C(d) & = & d(y).R(y) \end{array}\right.$$

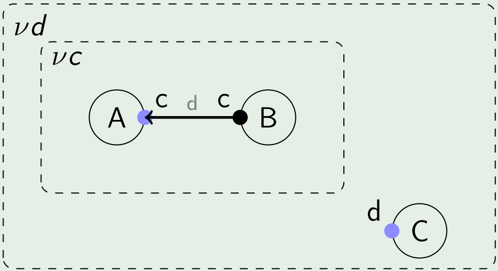
The π-calculus in a nutshell



$$\nu(c)[c(x).\overline{x}\langle m\rangle.P \mid \nu(d)[\overline{c}\langle d\rangle.Q \mid C(d)]]$$

$$\left[\begin{array}{lcl} A(c,m) & = & c(x).\overline{x}\langle m\rangle.P \\ B(c,d) & = & \overline{c}\langle d\rangle.Q \\ C(d) & = & d(y).R(y) \end{array}\right.$$

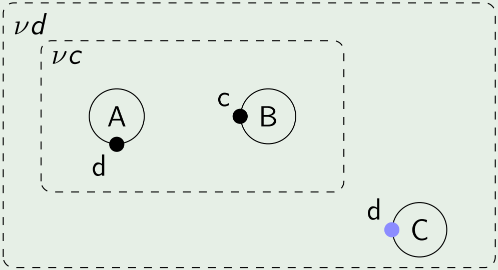
The π-calculus in a nutshell



$$\nu(c)\nu(d)[c(x).\overline{x}\langle m\rangle.P \mid \overline{c}\langle d\rangle.Q \mid C(d)]$$

$$\left[\begin{array}{lcl} A(c,m) & = & c(x).\overline{x}\langle m\rangle.P \\ B(c,d) & = & \overline{c}\langle d\rangle.Q \\ C(d) & = & d(y).R(y) \end{array}\right.$$

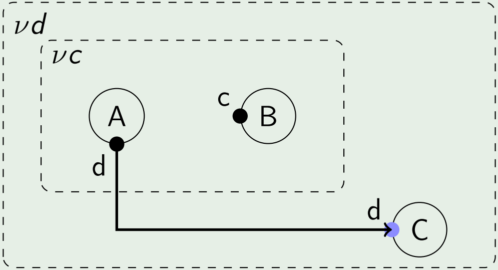
The π-calculus in a nutshell



$$\nu(c)\nu(d)[\overline{d}\langle m\rangle.P \mid Q \mid d(y).R(y)]$$

$$\left[\begin{array}{lcl} A(c,m) & = & c(x).\overline{x}\langle m\rangle.P \\ B(c,d) & = & \overline{c}\langle d\rangle.Q \\ C(d) & = & d(y).R(y) \end{array}\right.$$

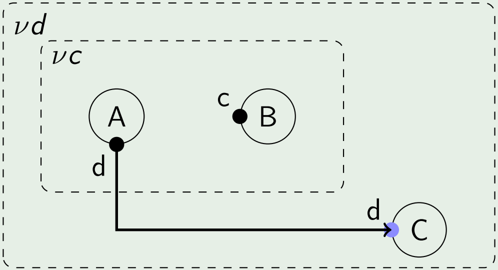
The π-calculus in a nutshell



$$\nu(c)\nu(d)[\overline{d}\langle m\rangle.P \mid Q \mid d(y).R(y)]$$

$$\left[\begin{array}{lcl} A(c,m) & = & c(x).\overline{x}\langle m\rangle.P \\ B(c,d) & = & \overline{c}\langle d\rangle.Q \\ C(d) & = & d(y).R(y) \end{array}\right.$$

The π-calculus in a nutshell



$$\nu(c)\nu(d)[P \mid Q \mid R(m)]$$

$$\left[\begin{array}{lcl} A(c,m) & = & c(x).\overline{x}\langle m\rangle.P \\ B(c,d) & = & \overline{c}\langle d\rangle.Q \\ C(d) & = & d(y).R(y) \end{array}\right.$$

Splendor and misery of the π-calculus

Splendor

- A **minimal** language to characterize concurrent and dynamic (a.k.a. "mobile") systems
- A very **expressive** language
- A (too ?) large **body of theoretical works**

Splendor and misery of the π-calculus

Splendor

- A **minimal** language to characterize concurrent and dynamic (a.k.a. "mobile") systems
- A very **expressive** language
- A (too ?) large **body of theoretical works**

Misery

- A somewhat unsettled theory with many semantic variants (early, late, open, barbed, etc.)
- A lack of modelling and verification tools
- A lack of implementations (cf. the π-threads project)

Why π ?

Why studying the π-calculus ?

Why π ?

Why studying the π-calculus ?

- Initially, to study dynamically reconfigurable systems (DRS)
- Then, to study mobile agents (which are DRS)
- Now, because it is a path towards program verification

Why π ?

Why studying the π-calculus ?

- Initially, to study dynamically reconfigurable systems (DRS)
- Then, to study mobile agents (which are DRS)
- Now, because it is a path towards program verification

More technically,

- to study the (finite) verification of infinite systems
- to study graph rewriting, especially graph relabelling

Plan

Modelling with the π-calculus

Objective 1

Design a visual language with expressive power comparable to the π-calculus and suitable for modelling purpose

Existing approaches

early attempts Milner's π-nets and Parrow's interaction diagrams

+ pedagogical tools
- informal, discontinued

Graph encodings in the DPO framework

+ formal approaches
- low-level, partial support, not suitable for verification

Claim : (control) graphs should be static (cf. UML, Petri-nets, etc.)

Verification techniques

Objective 2

Decidable characterization + efficient verification techniques

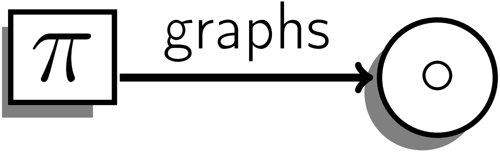
Drawbacks of existing approaches

- Mobility workbench based on open bisimilarity
  - complex ad-hoc algorithm for partition refinement
  - non-trivial detection of inactive names
  - costly because of name substitutions
- HAL based on HD-Automata
  - fine-grained interpretation of freshness
  - non-trivial detection of inactive names
  - indirect transformation ($\pi \rightarrow$ HDA $\rightarrow$ FSM)

Claim : simpler and more efficient techniques can be developed

Plan

The modelling framework



http://lip6.fr/Frederic.Peschanski/pigraphs

A dual formalism

- A visual language inspired by Petri nets
- A (textual) variant of the π-calculus

A dual characterization

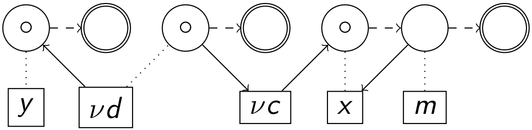
- graph relabelling
- labelled transition systems (LTS)

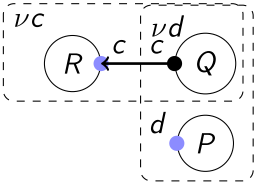
The (static) π-graph language

Principle

A "token-game" interpretation of the π-calculus constructs

Example : illustrating mobility





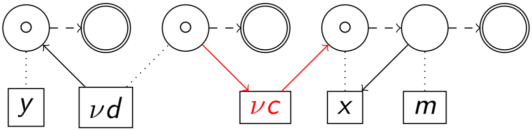
$$\boxed{\nu d(y)}\ 0 \parallel \boxed{\overline{\nu c}\langle \nu d\rangle}\ 0 \parallel \boxed{\nu c(x)}\ \overline{x}\langle m\rangle\ 0$$

The (static) π-graph language

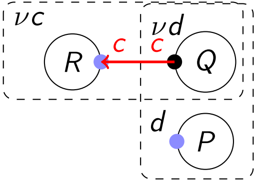
Principle

A “token-game” interpretation of the π-calculus constructs

Example : illustrating mobility



$$\boxed{\nu d(y)}\ 0 \parallel \boxed{\overline{\nu c}\langle \nu d\rangle}\ 0 \parallel \boxed{\nu c(x)}\ \overline{x}\langle m\rangle\ 0$$

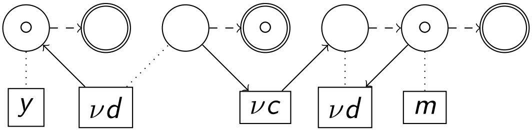


The (static) π-graph language

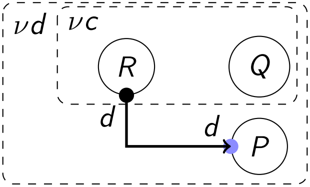
Principle

A "token-game" interpretation of the π-calculus constructs

Example : illustrating mobility



$$\boxed{\nu d(y)}\ 0 \parallel \overline{\nu c}\langle \nu d\rangle\ \boxed{0} \parallel \nu c(\nu d)\ \boxed{\overline{\nu d}\langle m\rangle}\ 0$$

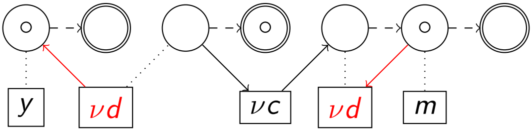


The (static) π-graph language

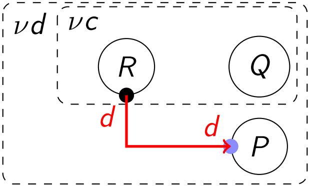
Principle

A “token-game” interpretation of the π-calculus constructs

Example : illustrating mobility



$$\boxed{\nu d(y)}\ 0 \parallel \overline{\nu c}\langle \nu d\rangle\ \boxed{0} \parallel \nu c(\nu d)\ \boxed{\overline{\nu d}\langle m\rangle}\ 0$$

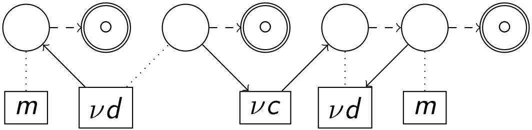


The (static) π-graph language

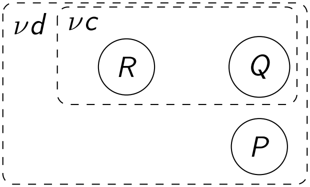
Principle

A “token-game” interpretation of the π-calculus constructs

Example : illustrating mobility



$$\nu d(m)\ \boxed{0} \parallel \overline{\nu c}\langle \nu d\rangle\ \boxed{0} \parallel \nu c(\nu d)\ \overline{\nu d}\langle m\rangle\ \boxed{0}$$

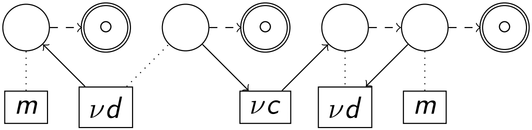


The (static) π-graph language

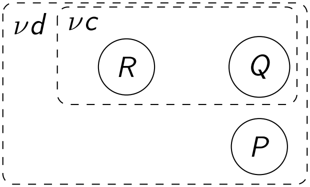
Principle

A "token-game" interpretation of the π-calculus constructs

Example : illustrating mobility



$$\nu d(m)\ \boxed{0} \parallel \overline{\nu c}\langle \nu d\rangle\ \boxed{0} \parallel \nu c(\nu d)\ \overline{\nu d}\langle m\rangle\ \boxed{0}$$



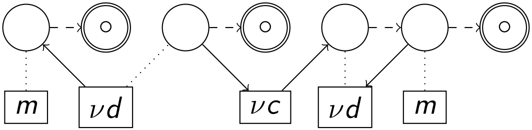
Remark 1 : the π-graphs have a static structure

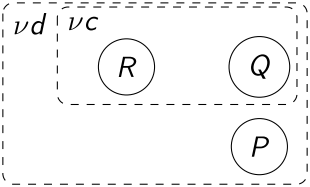
The (static) π-graph language

Principle

A "token-game" interpretation of the π-calculus constructs

Example : illustrating mobility





$$\nu d(m)\ \boxed{0} \parallel \overline{\nu c}\langle \nu d\rangle\ \boxed{0} \parallel \nu c(\nu d)\ \overline{\nu d}\langle m\rangle\ \boxed{0}$$

Remark 1 : the π-graphs have a static structure

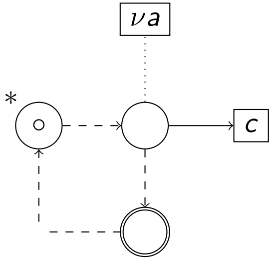
Remark 2 : direct correspondence with a (textual) process calculus

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



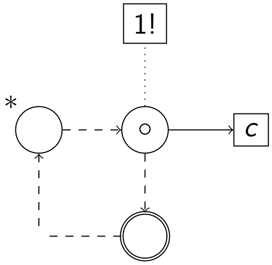
0 : $\boxed{*}[\overline{c}\langle\nu a\rangle.0]$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



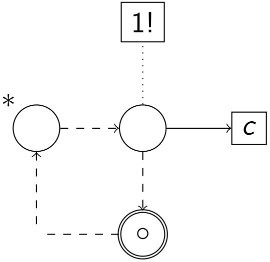
$$1 : *[\boxed{\overline{c}\langle\nu a \mid 1!\rangle}.0]$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



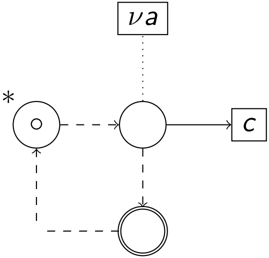
$$\xrightarrow{\overline{c}\langle 1!\rangle} 1 : *[\overline{c}\langle \nu a \mid 1!\rangle.\boxed{0}]$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



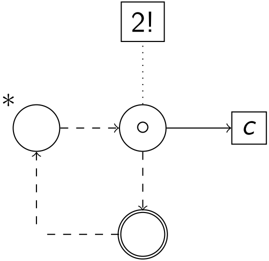
$$\xrightarrow{\overline{c}\langle 1! \rangle} 1 : \boxed{*}[\overline{c}\langle \nu a \rangle.0]$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



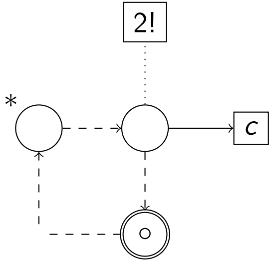
$$\xrightarrow{\overline{c}\langle 1!\rangle} 2 : *[\boxed{\overline{c}\langle \nu a \mid 2! \rangle}.0]$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



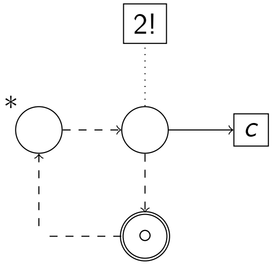
$$\xrightarrow{\overline{c}\langle 1! \rangle} \xrightarrow{\overline{c}\langle 2! \rangle} 2 : *[\overline{c}\langle \nu a \mid 2! \rangle.\boxed{0}]$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



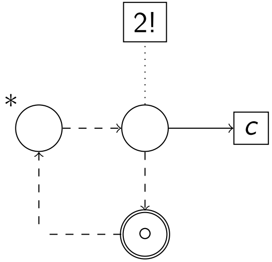
$$\xrightarrow{\overline{c}\langle 1!\rangle}\xrightarrow{\overline{c}\langle 2!\rangle} 2 : *[\overline{c}\langle \nu a \mid 2!\rangle.\boxed{0}] \xrightarrow{\overline{c}\langle 3!\rangle}$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



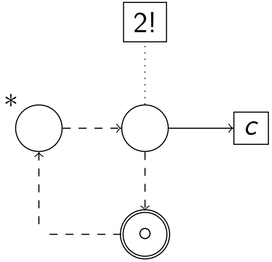
$$\xrightarrow{\overline{c}\langle 1!\rangle}\xrightarrow{\overline{c}\langle 2!\rangle} 2 : *[\overline{c}\langle \nu a \mid 2!\rangle.\boxed{0}] \xrightarrow{\overline{c}\langle 3!\rangle} \xrightarrow{\overline{c}\langle 4!\rangle} etc.$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



Remark 1 : synchronous
interpretation of binders using a linear clock (cf. [Sofsem'09]LNCS 5404)

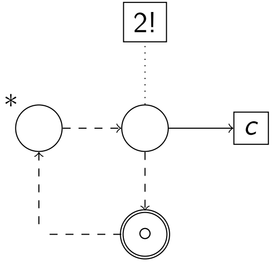
$$\xrightarrow{\overline{c}\langle 1!\rangle}\xrightarrow{\overline{c}\langle 2!\rangle} 2 : *[\overline{c}\langle \nu a \mid 2!\rangle.\boxed{0}] \xrightarrow{\overline{c}\langle 3!\rangle} \xrightarrow{\overline{c}\langle 4!\rangle} \textit{etc}.$$

π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example : A generator of fresh names :



Remark 1 : synchronous interpretation of binders using a linear clock (cf. [Sofsem'09]LNCS 5404)

Remark 2 : (minimalistic) infinite system

$$\xrightarrow{\overline{c}\langle 1!\rangle}\xrightarrow{\overline{c}\langle 2!\rangle} 2 : *[\overline{c}\langle \nu a \mid 2!\rangle.\boxed{0}] \xrightarrow{\overline{c}\langle 3!\rangle} \xrightarrow{\overline{c}\langle 4!\rangle} etc.$$

Operational semantics

Framework : graph relabelling + abstraction

- **local** in-place relabelling rules (eg : $\kappa; \gamma \vdash \boxed{\tau} P \xrightarrow{\tau} \kappa; \gamma \vdash \tau \boxed{P}$)
- abstract from low-level rewrites :
  $\pi \xrightarrow{\mu} \pi'$ (LTS) if $\pi \xrightarrow{\epsilon^* \mu} \pi'$ (graphs)

Operational semantics

Framework : graph relabelling + abstraction

- **local** in-place relabelling rules (eg : $\kappa; \gamma \vdash \boxed{\tau} P \xrightarrow{\tau} \kappa; \gamma \vdash \tau \boxed{P}$)
- abstract from low-level rewrites :
  $\pi \xrightarrow{\mu} \pi'$ (LTS) if $\pi \xrightarrow{\epsilon^* \mu} \pi'$ (graphs)

Important : graph context $\kappa; \delta$ with

κ a global clock

- a synchronous interpretation of inputs and bound outputs
- provides freshness "for free"

δ is a dynamic partition of names wrt. equality

- a unified interpretation of synchronizations and match
- allow names to be equated "on-the-fly"
- integrates read-write causality

Operational semantics

Framework : graph relabelling + abstraction

- **local** in-place relabelling rules (eg : $\kappa;\gamma \vdash \boxed{\tau}P \xrightarrow{\tau} \kappa;\gamma \vdash \tau\boxed{P}$)
- abstract from low-level rewrites :
  $\pi \xrightarrow{\mu} \pi'$ (LTS) if $\pi \xrightarrow{\epsilon^*\mu} \pi'$ (graphs)

Important : graph context $\kappa;\delta$ with

κ a global clock

- a synchronous interpretation of inputs and bound outputs
- provides freshness "for free"

δ is a dynamic partition of names wrt. equality

- a unified interpretation of synchronizations and match
- allow names to be equated "on-the-fly"
- integrates read-write causality

$\Rightarrow$ Ground transitions (+ bisimulation)

Plan

Infinity and π-graphs (cf. [Infinity 2010] EPTCS vol. 39)

Objective

a finite characterization of finite-control behaviors

Sources of infinity :

Counter-measures :

Infinity and π-graphs (cf. [Infinity 2010] EPTCS vol. 39)

Objective

a finite characterization of finite-control behaviors

Sources of infinity :

1. infinite low-level ϵ transitions
2. infinite partition δ of names
3. unbounded (linear) clock κ (ex. generator of fresh names)

Counter-measures :

Infinity and π-graphs (cf. [Infinity 2010] EPTCS vol. 39)

Objective

a finite characterization of finite-control behaviors

Sources of infinity :

1. infinite low-level ϵ transitions
2. infinite partition δ of names
3. unbounded (linear) clock κ (ex. generator of fresh names)

Counter-measures :

1. syntactic constraints (no match-only paths)
2. compact representations (implicit singletons)
3. structured clock model : causal clocks
4. garbage collection of inactive names

Causal clocks

Preamble : $\mathcal{N}_o \stackrel{\text{def}}{=} \{n! \mid n \in \mathbb{N}\}$ (resp. $\mathcal{N}_i \stackrel{\text{def}}{=} \{n? \mid n \in \mathbb{N}\}$) is the set of fresh output (resp. fresh input) names

Linear clocks	vs.	**Causal clocks**
$\kappa \in \mathbb{N}$		$\kappa \in \mathcal{N}_o \to \mathbb{P}(\mathcal{N}_i)$
$\text{init} \stackrel{\text{def}}{=} 0$		$\text{init} \stackrel{\text{def}}{=} \{\}$
$\text{out}(\kappa) \stackrel{\text{def}}{=} \text{next}_o(\kappa)!$		$\text{out}(\kappa) \stackrel{\text{def}}{=} \kappa \cup \{\text{next}_o(\kappa)! \mapsto \emptyset\}$
$\text{in}(\kappa) \stackrel{\text{def}}{=} \text{next}_i(\kappa)?$		$\text{in}(\kappa) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} o \mapsto (\kappa(o) \cup \{\text{next}_i(\kappa)?\}) \\ \mid o \in \text{dom}(\kappa) \end{array} \right\}$
$\text{next}_o(\kappa) \stackrel{\text{def}}{=} \kappa + 1$		$\text{next}_o(\kappa) \stackrel{\text{def}}{=} \min\left(\mathbb{N}^+ \setminus \{n \mid n! \in \text{dom}(\kappa)\}\right)$
$\text{next}_i(\kappa) \stackrel{\text{def}}{=} \kappa + 1$		$\text{next}_i(\kappa) \stackrel{\text{def}}{=} \min\left(\mathbb{N}^+ \setminus \{n \mid n? \in \bigcup \text{cod}(\kappa)\}\right)$

read-write causality :

$n! \prec_\kappa m? \stackrel{\text{def}}{=} n < m$		$n! \prec_\kappa m? \stackrel{\text{def}}{=} n! \in \text{dom}(\kappa) \wedge m? \in \kappa(n!)$

Illustrating read/write causality

Compare :

$$\{\} \vdash \boxed{\overline{c}\langle \nu a \rangle} d(x)[\nu a = x]P$$

with :

$$\{\} \vdash \boxed{d(x)} \overline{c}\langle \nu a \rangle [\nu a = x]P$$

Illustrating read/write causality

Compare :

$$\{\} \vdash \boxed{\overline{c}\langle \nu a\rangle} d(x)[\nu a = x]P$$

$$\xrightarrow{\overline{c}1!} \{1! \mapsto \emptyset\} \vdash \overline{c}\langle \nu a \mid 1!\rangle \boxed{d(x)} [(\nu a \mid 1!) = (x)]P$$

with :

$$\{\} \vdash \boxed{d(x)} \overline{c}\langle \nu a\rangle [\nu a = x]P$$

Illustrating read/write causality

Compare :

$$\{\} \vdash \boxed{\overline{c}\langle \nu a\rangle} d(x)[\nu a = x]P$$
$$\xrightarrow{\overline{c}1!} \{1! \mapsto \emptyset\} \vdash \overline{c}\langle \nu a \shortmid 1!\rangle \boxed{d(x)} [(\nu a \shortmid 1!) = (x)]P$$
$$\xrightarrow{d2?} \{1! \mapsto \{2?\}\} \vdash \overline{c}\langle \nu a \shortmid \overline{0}\rangle d(x \shortmid 2?) \boxed{[(\nu a \shortmid 1!) = (x \shortmid 2?)]} P$$

with :

$$\{\} \vdash \boxed{d(x)} \overline{c}\langle \nu a\rangle [\nu a = x]P$$

Illustrating read/write causality

Compare :

$$\{\} \vdash \boxed{\overline{c}\langle\nu a\rangle} d(x)[\nu a = x]P$$

$$\xrightarrow{\overline{c}1!} \{1! \mapsto \emptyset\} \vdash \overline{c}\langle\nu a \mid 1!\rangle \boxed{d(x)} [(\nu a \mid 1!) = (x)]P$$

$$\xrightarrow{d2?} \{1! \mapsto \{2?\}\} \vdash \overline{c}\langle\nu a \mid \overline{0}\rangle d(x \mid 2?) \boxed{[(\nu a \mid 1!) = (x \mid 2?)]} P$$

$$\xrightarrow{\epsilon} \{1! \mapsto \{2?\}\}; 1! = 2? \vdash \overline{c}\langle\nu a \mid 1!\rangle d(x \mid 2?)[(\nu a \mid 1!) = (x \mid 2?)] \boxed{P}$$

with :

$$\{\} \vdash \boxed{d(x)} \overline{c}\langle\nu a\rangle[\nu a = x]P$$

Illustrating read/write causality

Compare :

$$\{\} \vdash \boxed{\overline{c}\langle \nu a\rangle} d(x)[\nu a = x]P$$

$$\xrightarrow{\overline{c}1!} \{1! \mapsto \emptyset\} \vdash \overline{c}\langle \nu a \mid 1!\rangle \boxed{d(x)} [(\nu a \mid 1!) = (x)]P$$

$$\xrightarrow{d2?} \{1! \mapsto \{2?\}\} \vdash \overline{c}\langle \nu a \mid \overline{0}\rangle d(x \mid 2?) \boxed{[(\nu a \mid 1!) = (x \mid 2?)]} P$$

$$\xrightarrow{\epsilon} \{1! \mapsto \{2?\}\}; 1! = 2? \vdash \overline{c}\langle \nu a \mid 1!\rangle d(x \mid 2?)[(\nu a \mid 1!) = (x \mid 2?)] \boxed{P}$$

$$(\text{note} : 1! \prec_\kappa 2? \text{ since } 2? \in \kappa(1!))$$

with :

$$\{\} \vdash \boxed{d(x)} \overline{c}\langle \nu a\rangle[\nu a = x]P$$

Illustrating read/write causality

Compare :

$$\{\} \vdash \boxed{\overline{c}\langle \nu a\rangle} d(x)[\nu a = x]P$$
$$\xrightarrow{\overline{c}1!} \{1! \mapsto \emptyset\} \vdash \overline{c}\langle \nu a \shortmid 1!\rangle \boxed{d(x)} [(\nu a \shortmid 1!) = (x)]P$$
$$\xrightarrow{d2?} \{1! \mapsto \{2?\}\} \vdash \overline{c}\langle \nu a \shortmid \overline{0}\rangle d(x \shortmid 2?) \boxed{[(\nu a \shortmid 1!) = (x \shortmid 2?)]} P$$
$$\xrightarrow{\epsilon} \{1! \mapsto \{2?\}\}; 1! = 2? \vdash \overline{c}\langle \nu a \shortmid 1!\rangle d(x \shortmid 2?)[(\nu a \shortmid 1!) = (x \shortmid 2?)] \boxed{P}$$

$(\text{note} : 1! \prec_\kappa 2? \text{ since } 2? \in \kappa(1!))$

with :

$$\{\} \vdash \boxed{d(x)} \overline{c}\langle \nu a\rangle[\nu a = x]P$$
$$\xrightarrow{c1?} \{\} \vdash d(x \shortmid 1?) \boxed{\overline{c}\langle \nu a\rangle} [(\nu a) = (x \shortmid 1?)]P$$

Illustrating read/write causality

Compare :

$\{\} \vdash \boxed{\overline{c}\langle \nu a\rangle} d(x)[\nu a = x]P$

$\xrightarrow{\overline{c}1!} \{1! \mapsto \emptyset\} \vdash \overline{c}\langle \nu a \mid 1!\rangle \boxed{d(x)} [(\nu a \mid 1!) = (x)]P$

$\xrightarrow{d2?} \{1! \mapsto \{2?\}\} \vdash \overline{c}\langle \nu a \mid \overline{0}\rangle d(x \mid 2?) \boxed{[(\nu a \mid 1!) = (x \mid 2?)]} P$

$\xrightarrow{\epsilon} \{1! \mapsto \{2?\}\}; 1! = 2? \vdash \overline{c}\langle \nu a \mid 1!\rangle d(x \mid 2?)[(\nu a \mid 1!) = (x \mid 2?)] \boxed{P}$

$(\text{note} : 1! \prec_\kappa 2? \text{ since } 2? \in \kappa(1!))$

with :

$\{\} \vdash \boxed{d(x)} \overline{c}\langle \nu a\rangle[\nu a = x]P$

$\xrightarrow{c1?} \{\} \vdash d(x \mid 1?) \boxed{\overline{c}\langle \nu a\rangle} [(\nu a) = (x \mid 1?)]P$

$\xrightarrow{\overline{c}2!} \{2! \mapsto \emptyset\} \vdash d(x \mid 1?)\overline{c}\langle \nu a \mid 2!\rangle \boxed{[(\nu a \mid 2!) = (x \mid 1?)]} P$

Illustrating read/write causality

Compare :

$$\{\} \vdash \boxed{\overline{c}\langle\nu a\rangle} d(x)[\nu a = x]P$$

$$\xrightarrow{\overline{c}1!} \{1! \mapsto \emptyset\} \vdash \overline{c}\langle \nu a \mid 1!\rangle \boxed{d(x)} [(\nu a \mid 1!) = (x)]P$$

$$\xrightarrow{d2?} \{1! \mapsto \{2?\}\} \vdash \overline{c}\langle \nu a \mid \overline{0}\rangle d(x \mid 2?) \boxed{[(\nu a \mid 1!) = (x \mid 2?)]} P$$

$$\xrightarrow{\epsilon} \{1! \mapsto \{2?\}\}; 1! = 2? \vdash \overline{c}\langle \nu a \mid 1!\rangle d(x \mid 2?)[(\nu a \mid 1!) = (x \mid 2?)]\boxed{P}$$

$(\text{note} : 1! \prec_\kappa 2? \text{ since } 2? \in \kappa(1!))$

with :

$$\{\} \vdash \boxed{d(x)} \overline{c}\langle\nu a\rangle[\nu a = x]P$$

$$\xrightarrow{c1?} \{\} \vdash d(x \mid 1?)\boxed{\overline{c}\langle\nu a\rangle}[(\nu a) = (x \mid 1?)]P$$

$$\xrightarrow{\overline{c}2!} \{2! \mapsto \emptyset\} \vdash d(x \mid 1?)\overline{c}\langle \nu a \mid 2!\rangle \boxed{[(\nu a \mid 2!) = (x \mid 1?)]} P$$

$\not\rightarrow$ because $2! \not\prec_\kappa 1?$ $(1? \notin \kappa(2!))$

Garbage collection of inactive names

Definition : Active name

A name *n* is **active** in a π-graph with clock κ and partition δ iff

- either it is instantiated in the graph
- or it is a component of κ and δ (only for fresh outputs)

Garbage collection of inactive names

Definition : Active name

A name n is **active** in a π-graph with clock κ and partition δ iff

- either it is instantiated in the graph
- or it is a component of κ and δ (only for fresh outputs)

Question : how to avoid κ and δ to grow infinitely ?

Garbage collection of inactive names

Definition : **Active name**

A name n is **active** in a π-graph with clock κ and partition δ iff

- either it is instantiated in the graph
- or it is a component of κ and δ (only for fresh outputs)

Question : how to avoid κ and δ to grow infinitely ?

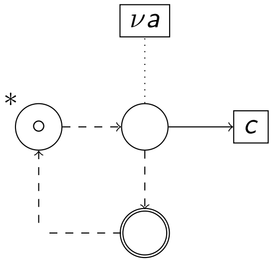
Answer : **Garbage collection of inactive names**

Let $\kappa; \delta \vdash G$ a graph with instantiations I then

$\mathrm{gc}(\pi) \stackrel{\text{def}}{=} \kappa'; \delta' \vdash G$ such that

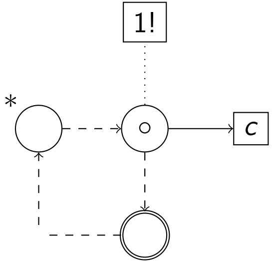
$$\begin{cases} \gamma' \stackrel{\text{def}}{=} \{E \cap (\mathcal{N}_f \cup \mathcal{N}_o \cup \mathrm{cod}(I)) \mid E \in \gamma\} \setminus \{\emptyset\} \\ \kappa' \stackrel{\text{def}}{=} \{d \mapsto \kappa(d) \cap \mathrm{cod}(I) \mid d \in \mathrm{dom}(\kappa) \wedge \left(\begin{array}{l} d \in \mathrm{cod}(I) \\ \vee (\{d\} \notin \gamma') \end{array} \right)\} \end{cases}$$

Illustrating garbage collection



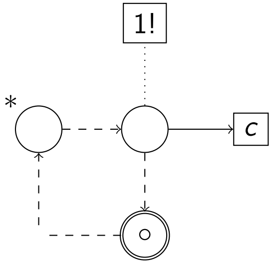
$$\{\} : \boxed{*}[\overline{c}\langle \nu a \rangle.0]$$

Illustrating garbage collection



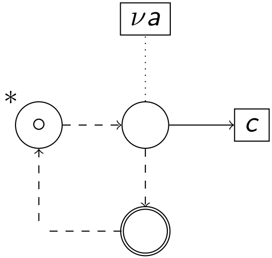
$$\{1! \mapsto \emptyset\} : *[\boxed{\overline{c}\langle \nu a \mid 1! \rangle}.0]$$

Illustrating garbage collection



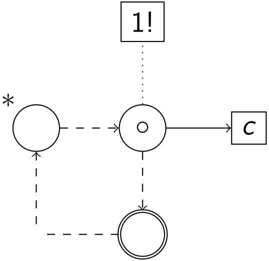
$$\xrightarrow{\overline{c}\langle 1!\rangle} \{1! \mapsto \emptyset\} : *[\overline{c}\langle \nu a \mid 1!\rangle.\boxed{0}]$$

Illustrating garbage collection



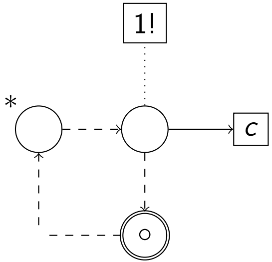
$$\xrightarrow{\overline{c}\langle 1! \rangle} \{\} : \boxed{*}[\overline{c}\langle \nu a \rangle.0]$$

Illustrating garbage collection



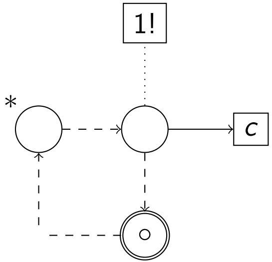
$$\xrightarrow{\overline{c}\langle 1! \rangle} \{1! \mapsto \emptyset\} : *[\boxed{\overline{c}\langle \nu a \mid 1! \rangle}.0]$$

Illustrating garbage collection



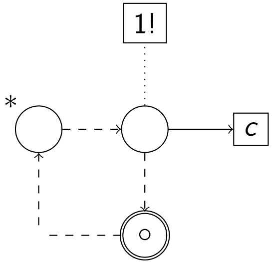
$$\xrightarrow{\overline{c}\langle 1!\rangle}\xrightarrow{\overline{c}\langle 1!\rangle} \{1! \mapsto \emptyset\} : *[\overline{c}\langle \nu a \mid 1!\rangle.\boxed{0}]$$

Illustrating garbage collection



$$\xrightarrow{\overline{c}\langle 1! \rangle} \xrightarrow{\overline{c}\langle 1! \rangle} \{1! \mapsto \emptyset\} : *[\overline{c}\langle \nu a \mid 1! \rangle.\boxed{0}] \xrightarrow{\overline{c}\langle 1! \rangle}$$

Illustrating garbage collection



$$\xrightarrow{\overline{c}\langle 1! \rangle}\xrightarrow{\overline{c}\langle 1! \rangle} \{1! \mapsto \emptyset\} : *[\overline{c}\langle \nu a \mid 1! \rangle.\boxed{0}] \xrightarrow{\overline{c}\langle 1! \rangle} \xrightarrow{\overline{c}\langle 1! \rangle} \mathit{etc}.$$

Decidability results

Finite systems

Let a transition system $\mathrm{lts}(\pi) = \langle Q, T \rangle$ with causal clock κ_Q of each state Q, then there are static bounds for fresh names :

- $\bigcup_Q \bigcup \mathrm{cod}(\kappa_Q) \subseteq \{1?, 2?, \ldots, |B|?\}$
  (where $|B|$ is the number of "boxes" in the graph)
- $\bigcup_Q \mathrm{dom}(\kappa_Q) \subseteq \{1!, 2!, \ldots, |B|!\}$
  (the proof for this is more involved)

$\Longrightarrow$ the sets Q and T are finite

Decidability results

Finite systems

Let a transition system $\mathrm{lts}(\pi) = \langle Q, T \rangle$ with causal clock κ_Q of each state Q, then there are static bounds for fresh names :

- $\bigcup_Q \bigcup \mathrm{cod}(\kappa_Q) \subseteq \{1?, 2?, \ldots, |B|?\}$
(where $|B|$ is the number of "boxes" in the graph)
- $\bigcup_Q \mathrm{dom}(\kappa_Q) \subseteq \{1!, 2!, \ldots, |B|!\}$
(the proof for this is more involved)

$\Longrightarrow$ the sets Q and T are finite

$\Longrightarrow$ "The" theorem

Bisimilarity for π-graphs with causal clocks is decidable

Plan

Petri nets translations of the π-calculus

Motivations

- A (Petri nets-based) verification framework "for free"
- An exercise in expressivity
- A study of the finite/infinite frontier

Petri nets translations of the π-calculus

Motivations

- A (Petri nets-based) verification framework "for free"
- An exercise in expressivity
- A study of the finite/infinite frontier

Semantic translations

1. Reachability analysis of a π-calculus term
2. Transition system encoded as a (generally low-level) Petri Net

Petri nets translations of the π-calculus

Motivations

- A (Petri nets-based) verification framework "for free"
- An exercise in expressivity
- A study of the finite/infinite frontier

Semantic translations

1. Reachability analysis of a π-calculus term
2. Transition system encoded as a (generally low-level) Petri Net

Syntactic translations

1. Syntax-driven translation of a π-calculus term
2. Construction of a (generally high-level) Petri Net

Petri nets translations of the π-calculus

Motivations

- A (Petri nets-based) verification framework "for free"
- An exercise in expressivity
- A study of the finite/infinite frontier

Semantic translations

1. Reachability analysis of a π-calculus term
2. Transition system encoded as a (generally low-level) Petri Net

Syntactic translations

1. Syntax-driven translation of a π-calculus term
2. Construction of a (generally high-level) Petri Net

Comparison : size of the translation, low/high level nets.

Existing (recent) translations

Semantic translation by Meyer and Gorrieri

- Translation of a π-calculus without match
- Produces a Place/Transition net characterizing the reduction semantics of the terms
- Finite characterization of finite control processes (FCP)

Existing (recent) translations

Semantic translation by Meyer and Gorrieri

- Translation of a π-calculus without match
- Produces a Place/Transition net characterizing the reduction semantics of the terms
- Finite characterization of finite control processes (FCP)

Syntactic translation by Devillers, Klaudel and Koutny

- Modular translation of a π-calculus with match
- Construction of a high-level Petri net with read arcs
- The net semantics correspond to the transition semantics
- Infinite characterization of FCP

Existing (recent) translations

Semantic translation by Meyer and Gorrieri

- Translation of a π-calculus without match
- Produces a Place/Transition net characterizing the reduction semantics of the terms
- Finite characterization of finite control processes (FCP)

Syntactic translation by Devillers, Klaudel and Koutny

- Modular translation of a π-calculus with match
- Construction of a high-level Petri net with read arcs
- The net semantics correspond to the transition semantics
- Infinite characterization of FCP

Goals

- a simpler syntactic translation
- a finite characterization of FCP

Towards a (much) simpler syntactic translation

Remark : The π-graphs already provide a "token-game" interpretation of π-calculus behaviors

Towards a (much) simpler syntactic translation

Remark : The π-graphs already provide a “token-game” interpretation of π-calculus behaviors

Idea

Consider π-graphs as the result of the first step of a syntactic translation $\Rightarrow$ intermediate language

Towards a (much) simpler syntactic translation

Remark : The π-graphs already provide a "token-game" interpretation of π-calculus behaviors

Idea

Consider π-graphs as the result of the first step of a syntactic translation $\Rightarrow$ intermediate language

Second step :

$\Rightarrow$ Translating the π-graphs to (not so) high-level Petri nets

1. Inductive translation of a π-graph to a structure net
2. Connection of the structural net to a global context net

The structure net

Principle : inductive decomposition

1. Translation of each prefix as an elementary Petri net with prev/next transitions.
2. Fusion of prev/next transitions to form sequential compositions
3. Embedding of the translated process within an Iterator Petri nets

The structure net

Principle : inductive decomposition

1. Translation of each prefix as an elementary Petri net with prev/next transitions.
2. Fusion of prev/next transitions to form sequential compositions
3. Embedding of the translated process within an Iterator Petri nets

$\Rightarrow$ each place of the structure (always) contains a single token

Control part of the token :

inactive : color $\emptyset$

active : color $\circ$

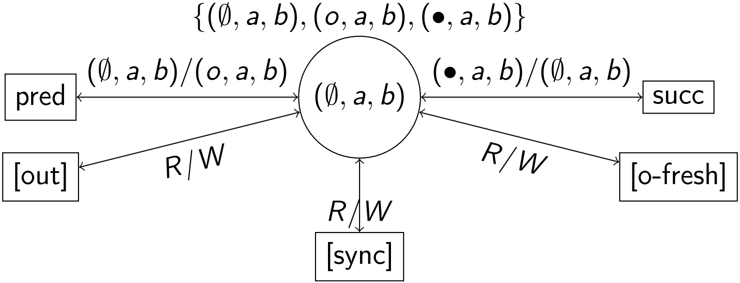
continuation : color $\bullet$

Elementary structure nets (1)

Example 1 : translating an output $\overline{a}\langle b\rangle$

Elementary structure nets (1)

Example 1 : translating an output $\overline{a}\langle b\rangle$

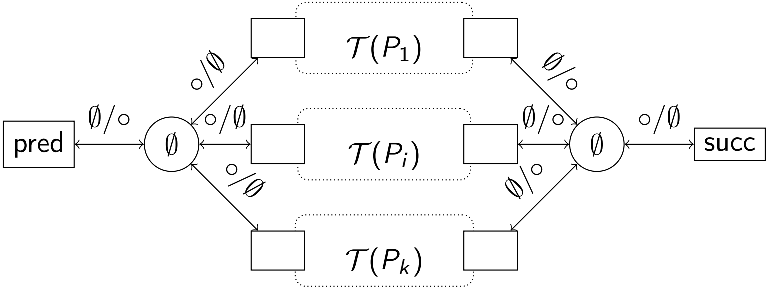


Elementary structure nets (2)

Example 2 : translating a choice $\sum[P_1 + \ldots + P_i + \ldots + P_n]$

Elementary structure nets (2)

Example 2 : translating a choice $\sum[P_1 + \ldots + P_i + \ldots + P_n]$

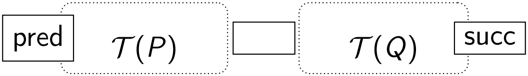


Structure nets (3) : sequence and iteration

Example 3 : translating a sequence PQ

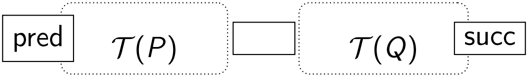
Structure nets (3) : sequence and iteration

Example 3 : translating a sequence PQ



Structure nets (3) : sequence and iteration

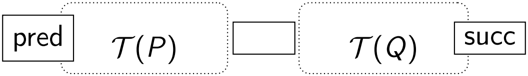
Example 3 : translating a sequence PQ



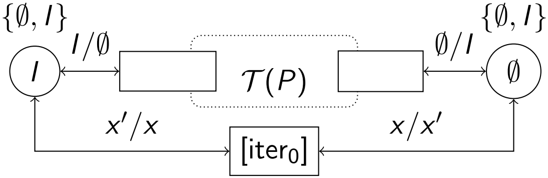
Example 4 : translating an iterator $I : *P$

Structure nets (3) : sequence and iteration

Example 3 : translating a sequence PQ



Example 4 : translating an iterator $I : *P$



The context net

Principles

Connect each transition [*trans*] (7 in total) to a global context :

- a place with the name instantiations, clock and name partition
- a place with the current observations attached to an *obs* transition

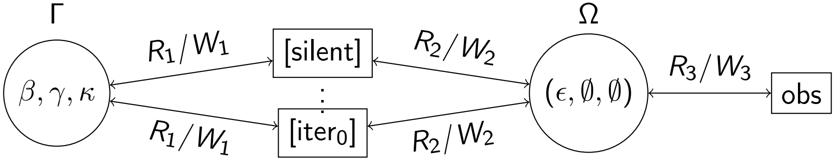
The context net

Principles

Connect each transition [*trans*] (7 in total) to a global context :

- a place with the name instantiations, clock and name partition
- a place with the current observations attached to an *obs* transition

Structure of the context net :



Properties of the translation

Ongoing proofs

- the size of the translated Petri Net is linear in the size of the π-graphs
- generates "one-token-everywhere" nets (stronger than 1-safe)
- no dynamic resource creation : everything is pre-allocated (e.g. size of the clock and the partition)
- **faithfulness** : the π-graph and its translated Petri net generate (by abstraction) the <u>same</u> LTS
- **Consequence** : bisimilarity is decidable for the translated nets

Properties of the translation

Ongoing proofs

- the size of the translated Petri Net is linear in the size of the π-graphs
- generates "one-token-everywhere" nets (stronger than 1-safe)
- no dynamic resource creation : everything is pre-allocated (e.g. size of the clock and the partition)
- **faithfulness** : the π-graph and its translated Petri net generate (by abstraction) the same LTS
- **Consequence** : bisimilarity is decidable for the translated nets

Conjectures

- the semantics (of both π-graphs and translated Petri nets) are compositional
- various equivalent firing strategies (e.g. standard vs. synchronous)
- well-structured low-level unfolding

About faithfulness

The design of the translation eases the faithfulness proof :

- each abstracted transition $\Gamma \vdash \pi \xrightarrow{\alpha} \pi'$ in the π-graphs corresponds to an activation of the *obs* transition in the Petri net.
- the observation place Ω contains the label α
- the token in the context place is Γ

About faithfulness

The design of the translation eases the faithfulness proof :

- each abstracted transition $\Gamma \vdash \pi \xrightarrow{\alpha} \pi'$ in the π-graphs corresponds to an activation of the *obs* transition in the Petri net.
- the observation place Ω contains the label α
- the token in the context place is Γ

Remark : we do not need to match the low-level transitions

Plan

Summary

- A visual paradigm *and* a process calculus
- Expressivity of the (finite-control) π-calculus : mobility, etc.
- Ground transitions and bisimulations
  $\Rightarrow$ standard techniques for verification
- Decidable characterization (with causal clocks)

Summary

- A visual paradigm *and* a process calculus
- Expressivity of the (finite-control) π-calculus : mobility, etc.
- Ground transitions and bisimulations
  $\Rightarrow$ standard techniques for verification
- Decidable characterization (with causal clocks)

Ongoing works

- Compositionality ? (conjecture : yes, but non-trivial proof)
- Develop the meta-theory by abstract interpretation using a (new) variant of the π-calculus
  (+ encoding in the Coq proof assistant)
- Translation to (high-level) Petri nets ("almost" done !)
- From iterators to replicators (infinite control)
- Application : $\Rightarrow$ π explorer tool