A Decidable Characterization of a Graphical Pi-Calculus with Iterators

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

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

Infinity 2010 - Singapour
Plan

1. Motivations
2. Introducing the (static) $\pi$-graphs
3. A decidable characterization
4. Conclusion and future work
Motivations

Introducing the (static) $\pi$-graphs

A decidable characterization

Conclusion and future work
Motivations

Modelling with the $\pi$-calculus

Objective 1

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

Existing approaches

early attempts Milner’s $\pi$-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.)
Motivations

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 ($\pi \rightarrow \text{HDA} \rightarrow \text{FSM}$)

Claim: simpler and more efficient techniques can be developed
Plan

1. Motivations
2. Introducing the (static) $\pi$-graphs
3. A decidable characterization
4. Conclusion and future work
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
Introducing the (static) $\pi$-graphs

The modelling framework

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

A dual formalism
- A visual language inspired by Petri nets
- A (textual) variant of the $\pi$-calculus

A dual characterization
- (inductive) graph relabelling
- labelled transition systems (LTS)
The (static) $\pi$-graph language

**Principle**
A token-based interpretation of the basic $\pi$-calculus constructs

**Example**: illustrating mobility

$$\nu d(y) \parallel \overline{\nu c}\langle \nu d \rangle \parallel \nu c(x) \parallel \overline{x}\langle m \rangle \parallel 0$$
The (static) $\pi$-graph language

**Principle**

A token-based interpretation of the basic $\pi$-calculus constructs

**Example**: illustrating mobility

\[
\nu d(y) \parallel \nu c(\nu d) \parallel \nu c(x) \parallel \bar{x}\langle m\rangle 0
\]

Remark 1: the $\pi$-graphs have a static structure

Remark 2: direct correspondence with a (textual) process calculus
The (static) \( \pi \)-graph language

**Principle**

A token-based interpretation of the basic \( \pi \)-calculus constructs

**Example : illustrating mobility**

\[
\nu d(y) \parallel \nu c(\nu d) \parallel \nu c(\nu d) \parallel \nu d(\nu c(\nu d)) \parallel \nu d(\nu c(\nu d))
\]
The (static) $\pi$-graph language

**Principle**

A token-based interpretation of the basic $\pi$-calculus constructs

**Example:** illustrating mobility

$$\nu \circ 0 \parallel \nu c \langle \nu d \rangle 0 \parallel \nu c(\nu d) \overline{\nu d} \langle m \rangle 0$$
The (static) $\pi$-graph language

Principle

A token-based interpretation of the basic $\pi$-calculus constructs

Example: illustrating mobility

\[
\nu_d(m) \parallel \nu c(\nu d) \parallel \nu c(\nu d) \nu d(\nu) \nu c(\nu) \nu d(\nu)
\]
The (static) $\pi$-graph language

**Principle**
A token-based interpretation of the basic $\pi$-calculus constructs

**Example**: illustrating mobility

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

**Remark 1**: the $\pi$-graphs have a static structure
The (static) $\pi$-graph language

**Principle**

A token-based interpretation of the basic $\pi$-calculus constructs

**Example:** illustrating mobility

\[
\nu_d(m) \parallel \nu c(\nu_d) \parallel \nu c(\nu d) \parallel \nu d(\nu d) \parallel \nu c(\nu d) \parallel \nu d(\nu d)
\]

**Remark 1:** the $\pi$-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:

\[
\nu a
\]

\[
\ast \left[ \left[ \bar{c} \left\langle \nu a \right\rangle \right] 0 \right]
\]
Introducing the (static) $\pi$-graphs

$\pi$-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example: A generator of fresh names:

\[
1 : \ast \left[ \overline{c} \langle \nu a \mid 1! \rangle .0 \right]
\]
π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example: A generator of fresh names:

\[ \overline{c}(1!) \rightarrow 1 : *[\overline{c}(\nu a \mid 1!).0] \]
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
Introducing the (static) $\pi$-graphs

$\pi$-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example: A generator of fresh names:

$$\nu a \xrightarrow{*} 1 : \star [\bar{c} \langle \nu a \rangle . 0]$$
π-graphs with iterators

Iterators

Recursive behaviors as (static) graphs rewrites

Example: A generator of fresh names:

\[
\overline{c}^{(1!)} \rightarrow 2 : \ast [\overline{c}\langle\nu a | 2!\rangle].0
\]
Iterators

Recursive behaviors as (static) graphs rewrites

Example: A generator of fresh names:

\[
\begin{align*}
\overline{c}\langle 1! \rangle & \xrightarrow{\overline{c}\langle 2! \rangle} 2 : \ast [\overline{c}\langle \nu a \mid 2! \rangle . \overline{0}]
\end{align*}
\]

Remark 1: Synchronous interpretation of binders using a linear clock

Remark 2: (minimalistic) infinite system
Iterators

Recursive behaviors as (static) graphs rewrites

Example: A generator of fresh names:

\[
\begin{align*}
\overline{c}\langle 1! \rangle & \rightarrow \overline{c}\langle 2! \rangle \\
\overline{c}\langle 1! \rangle & \rightarrow 2 : \star \overline{c}\langle \nu a \mid 2! \rangle.0 & \overline{c}\langle 3! \rangle
\end{align*}
\]
Iterators

Recursive behaviors as (static) graphs rewrites

Example: A generator of fresh names:

\[
\begin{array}{c}
\pi\langle 1! \rangle \xrightarrow{c} \pi\langle 2! \rangle \\
\pi\langle 2! \rangle \xrightarrow{c} \pi\langle 3! \rangle \\
\pi\langle 3! \rangle \xrightarrow{c} \pi\langle 4! \rangle \rightarrow \text{etc.}
\end{array}
\]
Iterators

Recursive behaviors as (static) graphs rewrites

**Example**: A generator of fresh names:

\[
\begin{array}{c}
\ast \\
\bigcirc \rightarrow c \langle 1! \rangle \\
\bigcirc \rightarrow c \langle 2! \rangle \\
\bigcirc \rightarrow 2 : *_{\bigcirc} [c \langle \nu a \mid 2! \rangle \cdot \bigcirc] \\
\bigcirc \rightarrow c \langle 3! \rangle \\
\bigcirc \rightarrow c \langle 4! \rangle \\
\end{array}
\]

Remark 1: synchronous interpretation of binders using a linear clock

\[
\begin{align*}
\cbar{c} \langle 1! \rangle & \xrightarrow{\cbar{c} \langle 2! \rangle} 2 : *_{\cbar{c}} [\cbar{\nu a} \mid 2! \rangle \cdot \bigcirc] & \cbar{c} \langle 3! \rangle & \xrightarrow{\cbar{c} \langle 4! \rangle} etc.
\end{align*}
\]
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
Introducing the (static) $\pi$-graphs

$\pi$-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

Remark 2: (minimalistic) infinite system

\[
\begin{align*}
\overline{c}\langle 1! \rangle & \xrightarrow{\overline{c}\langle 2! \rangle} 2 : \ast \overline{c}\langle \nu a \mid 2! \rangle. \overline{0} \xrightarrow{\overline{c}\langle 3! \rangle} \overline{c}\langle 4! \rangle \xrightarrow{\text{etc.}}
\end{align*}
\]
Operational semantics

Framework: graph relabelling + abstraction

- **Local** in-place relabelling rules (eg: \( \kappa; \gamma \vdash \tau P \xrightarrow{\tau} \kappa; \gamma \vdash \tau P \))
- abstract from low-level rewrites:

  \( \pi \xrightarrow{\mu} \pi' \) (LTS) if \( \pi \xrightarrow{\epsilon^* \mu} \pi' \) (graphs)
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
Introducing the (static) $\pi$-graphs

Operational semantics

Framework: graph relabelling + abstraction
- **local** in-place relabelling rules (eg: $\kappa; \gamma \vdash [\tau P] \stackrel{\tau}{\longrightarrow} \kappa; \gamma \vdash [\tau P]$)
- abstract from low-level rewrites:
  $\pi \xrightarrow{\mu} \pi'$ (LTS) if $\pi \xrightarrow{\varepsilon^* \mu} \pi'$ (graphs)

Important: graph context $\kappa; \delta$ with
- $\kappa$ a **global clock**
  - a synchronous interpretation of inputs and bound outputs (escapes)
  - provides freshness “for free”
- $\delta$ 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
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
Introducing the (static) $\pi$-graphs

## Operational semantics

**Framework**: graph relabelling + abstraction

- **Local** in-place relabelling rules (eg: $\kappa; \gamma \vdash \tau P \xrightarrow{\tau} \kappa; \gamma \vdash \tau P$)
- abstract from low-level rewrites:

$$\pi \xrightarrow{\mu} \pi' \text{ (LTS)} \text{ if } \pi \xrightarrow{e^* \mu} \pi' \text{ (graphs)}$$

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

- **$\kappa$** a **global clock**
  - a *synchronous* interpretation of inputs and bound outputs (escapes)
  - provides *freshness* “for free”

- **$\delta$** 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)
Infinity and $\pi$-graphs

Objective

a finite characterization of finite-control behaviors

Sources of infinity :

Counter-measures :
Infinity and $\pi$-graphs

Objective

- a finite characterization of finite-control behaviors

Sources of infinity:

1. infinite low-level $\epsilon$ transitions
2. infinite partition $\delta$ of names
3. unbounded (linear) clock $\kappa$ (ex. generator of fresh names)

Counter-measures:
Infinity and \( \pi \)-graphs

Objective

a **finite** characterization of **finite-control** behaviors

Sources of infinity:

1. infinite low-level \( \epsilon \) transitions
2. infinite partition \( \delta \) of names
3. unbounded (linear) clock \( \kappa \) (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
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
A decidable characterization

Causal clocks

Preamble: \( \mathcal{N}_o \defeq \{ n! | n \in \mathbb{N} \} \) (resp. \( \mathcal{N}_i \defeq \{ n? | n \in \mathbb{N} \} \)) is the set of fresh output (resp. fresh input) names.

**Linear clocks** vs. **Causal clocks**

- Linear clocks:
  - \( \kappa \in \mathbb{N} \)
  - \( \text{init} \defeq 0 \)
  - \( \text{out}(\kappa) \defeq \text{next}_o(\kappa)! \)
  - \( \text{in}(\kappa) \defeq \text{next}_i(\kappa)? \)
  - \( \text{next}_o(\kappa) \defeq \kappa + 1 \)
  - \( \text{next}_i(\kappa) \defeq \kappa + 1 \)

- Causal clocks:
  - \( \kappa \in (\{\bot\} \cup \mathcal{N}_o) \rightarrow \mathcal{P}(\mathcal{N}_i) \)
  - \( \text{init} \defeq \{ \bot \mapsto \emptyset \} \)
  - \( \text{out}(\kappa) \defeq \kappa \cup \{ \text{next}_o(\kappa)! \mapsto \emptyset \} \)
  - \( \text{in}(\kappa) \defeq \left\{ \begin{array}{l}
    o \mapsto (\kappa(o) \cup \{ \text{next}_i(\kappa)? \}) \\
    | o \in \text{dom}(\kappa)
  \end{array} \right\} \)
  - \( \text{next}_o(\kappa) \defeq \min (\mathbb{N}^+ \setminus \{ n | n! \in \text{dom}(\kappa) \}) \)
  - \( \text{next}_i(\kappa) \defeq \min (\mathbb{N}^+ \setminus \{ n | n? \in \bigcup \text{cod}(\kappa) \}) \)

**Read-write causality:**

- \( n! \prec_\kappa n? \defeq n < m \)
- \( n! \prec_\kappa m? \defeq n! \in \text{dom}(\kappa) \land m? \in \kappa(n!) \)
Illustrating read/write causality

Compare:
\[ \{\bot \leftrightarrow \emptyset\} \vdash [\nu a = x] P \]

with:
\[ \{\bot \leftrightarrow \emptyset\} \vdash [\nu a = x] P \]
A Decidable Characterization of a Graphical Pi-Calculus with Iterators

Illustrating read/write causality

Compare:
\[
\begin{align*}
\{\bot \leftrightarrow \emptyset\} & \vdash \overline{c}\langle \nu a \rangle d(x)[\nu a = x]P \\
\overline{c}1! & \rightarrow \{\bot \leftrightarrow \emptyset, 1! \leftrightarrow \emptyset\} \vdash \overline{c}\langle \nu a \mid 1! \rangle \overline{d}(x)[(\nu a \mid 1!) = (x)]P
\end{align*}
\]

with:
\[
\{\bot \leftrightarrow \emptyset\} \vdash d(x)\overline{c}\langle \nu a \rangle[\nu a = x]P
\]
Illustrating read/write causality

Compare:

\[\{⊥ \mapsto \emptyset\} \vdash c\langle \nu a \rangle d(x)[\nu a = x]P\]

\[\xrightarrow{c1!} \{⊥ \mapsto \emptyset, 1! \mapsto \emptyset\} \vdash c\langle \nu a \mid 1! \rangle d(x)[(\nu a \mid 1!) = (x)]P\]

\[d2\? \left\{\begin{array}{c}
\{⊥ \mapsto \{2?\}\} \\
1! \mapsto \{2?\}
\end{array}\right\} \vdash c\langle \nu a \mid \nu \rangle d(x \mid 2?) [(\nu a \mid 1!) = (x \mid 2?)] P\]

with:

\[\{⊥ \mapsto \emptyset\} \vdash d(x) c\langle \nu a \rangle[\nu a = x]P\]
Illustrating read/write causality

Compare:
\[
\{\bot \mapsto \emptyset\} \vdash \overline{c} \langle \nu a \rangle d(x)[\nu a = x] P
\]
\[
\overline{c}1! \rightarrow \{\bot \mapsto \emptyset, 1! \mapsto \emptyset\} \vdash \overline{c} \langle \nu a \vdash 1! \rangle d(x)[(\nu a \vdash 1!) = (x)] P
\]
\[
d2?\rightarrow \left\{\begin{array}{l}
\bot \mapsto \{2?\} \\
1! \mapsto \{2?\}
\end{array}\right\} \vdash \overline{c} \langle \nu a \vdash 0 \rangle d(x \vdash 2?)[(\nu a \vdash 1!) = (x \vdash 2?)] P
\]
\[
\epsilon \rightarrow \left\{\begin{array}{l}
\bot \mapsto \{2?\} \\
1! \mapsto \{2?\}
\end{array}\right\}; 1! = 2? \vdash \overline{c} \langle \nu a \vdash 1! \rangle d(x \vdash 2?)[(\nu a \vdash 1!) = (x \vdash 2?)] P
\]

with:
\[
\{\bot \mapsto \emptyset\} \vdash d(x) \overline{c} \langle \nu a \rangle[\nu a = x] P
\]
Illustrating read/write causality

Compare:

\[ \{ \bot \mapsto \emptyset \} \vdash \overline{c} \langle \nu a \rangle d(x)[\nu a = x] P \]

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

\[ d 2? \mapsto \left\{ \begin{array}{l}
\bot \mapsto \{2?\} \\
1! \mapsto \{2?\}
\end{array} \right\} \vdash \overline{c} \langle \nu a \mid 0 \rangle d(x \mid 2?)[(\nu a \mid 1!) = (x \mid 2?)] P \]

\[ \epsilon \mapsto \left\{ \begin{array}{l}
\bot \mapsto \{2?\} \\
1! \mapsto \{2?\}
\end{array} \right\} ; 1! = 2? \vdash \overline{c} \langle \nu a \mid 1! \rangle d(x \mid 2?)[(\nu a \mid 1!) = (x \mid 2?)] P \]

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

with:

\[ \{ \bot \mapsto \emptyset \} \vdash d(x) \overline{c} \langle \nu a \rangle [\nu a = x] P \]
Illustrating read/write causality

Compare:

\[ \{ \bot \mapsto \emptyset \} \vdash \overline{c} \langle \nu a \rangle d(x)[\nu a = x]P \]

\[ \overline{c}1! \rightarrow \{ \bot \mapsto \emptyset, 1! \mapsto \emptyset \} \vdash \overline{c} \langle \nu a \cdot 1! \rangle \overline{d}(x)[(\nu a \cdot 1!) = (x)]P \]

\[ d2? \rightarrow \left\{ \begin{array}{c}
\bot \mapsto \{2?\} \\
1! \mapsto \{2?\}
\end{array} \right\} \vdash \overline{c} \langle \nu a \cdot \overline{0} \rangle \overline{d}(x \cdot 2?)[(\nu a \cdot 1!) = (x \cdot 2?)]P \]

\[ \epsilon \rightarrow \left\{ \begin{array}{c}
\bot \mapsto \{2?\} \\
1! \mapsto \{2?\}
\end{array} \right\}; 1! = 2? \vdash \overline{c} \langle \nu a \cdot 1! \rangle \overline{d}(x \cdot 2?)[(\nu a \cdot 1!) = (x \cdot 2?)]P \]

(note: \( 1! \prec_\kappa 2? \) since \( 2? \in \kappa(1!) \))

with:

\[ \{ \bot \mapsto \emptyset \} \vdash d(x) \overline{c} \langle \nu a \rangle [\nu a = x]P \]

\[ \overline{c}1? \rightarrow \{ \bot \mapsto \{1?\} \} \vdash d(x \cdot 1?) \overline{c} \langle \nu a \rangle [(\nu a) = (x \cdot 1?)]P \]
Illustrating read/write causality

Compare:

\[
\{ \bot \mapsto \emptyset \} \vdash \overline{c} \langle \nu a \rangle d(x)[\nu a = x]P
\]

\[
\overline{c}1! \quad \{ \bot \mapsto \emptyset, 1! \mapsto \emptyset \} \vdash \overline{c} \langle \nu a \mid 1! \rangle d(x)[(\nu a \mid 1!) = (x)]P
\]

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

\[
\epsilon \quad \{ \bot \mapsto \{2?\}, 1! \mapsto \{2?\} \} \ni 1! = 2? \vdash \overline{c} \langle \nu a \mid 1! \rangle d(x \mid 2?)[(\nu a \mid 1!) = (x \mid 2?)]P
\]

(note: \(1! \prec_\kappa 2?\) since \(2? \in \kappa(1!))

with:

\[
\{ \bot \mapsto \emptyset \} \vdash d(x) \overline{c} \langle \nu a \rangle[\nu a = x]P
\]

\[
c1? \quad \{ \bot \mapsto \{1?\} \} \vdash d(x \mid 1?) \overline{c} \langle \nu a \rangle[(\nu a) = (x \mid 1?)]P
\]

\[
c2! \quad \{ \bot \mapsto \{1?\}, 2! \mapsto \emptyset \} \vdash d(x \mid 1?) \overline{c} \langle \nu a \mid 2! \rangle[(\nu a \mid 2!) = (x \mid 1?)]P
\]
Illustrating read/write causality

Compare:

\[
\{ \bot \mapsto \emptyset \} \vdash \bar{c}\langle \nu a \rangle d(x)[\nu a = x]P
\]

\[
\xrightarrow{c1!} \{ \bot \mapsto \emptyset, 1! \mapsto \emptyset \} \vdash \bar{c}\langle \nu a \times 1! \rangle d(x)[(\nu a \times 1!) = (x)]P
\]

\[
\xrightarrow{d2?} \begin{cases} \bot \mapsto \{2?\} \\ 1! \mapsto \{2?\} \end{cases} \vdash \bar{c}\langle \nu a \times 0 \rangle d(x \times 2?)[(\nu a \times 1!) = (x \times 2?)]P
\]

\[
\xrightarrow{\epsilon} \begin{cases} \bot \mapsto \{2?\} \\ 1! \mapsto \{2?\} \end{cases} ; 1! = 2? \vdash \bar{c}\langle \nu a \times 1! \rangle d(x \times 2?)[(\nu a \times 1!) = (x \times 2?)]P
\]

(note : \(1! \prec_\kappa 2?\) since \(2? \in \kappa(1!)\))

with:

\[
\{ \bot \mapsto \emptyset \} \vdash d(x) \bar{c}\langle \nu a \rangle[\nu a = x]P
\]

\[
\xrightarrow{c1?} \{ \bot \mapsto \{1?\} \} \vdash d(x \times 1?) \bar{c}\langle \nu a \rangle[(\nu a) = (x \times 1?)]P
\]

\[
\xrightarrow{c2!} \{ \bot \mapsto \{1?\}, 2! \mapsto \emptyset \} \vdash d(x \times 1?)\bar{c}\langle \nu a \times 2! \rangle[(\nu a \times 2!) = (x \times 1?)]P
\]

\(\rightarrow\) because \(2! \not\prec_\kappa 1?\) (\(1? \not\in \kappa(2!)\))
Garbage collection of inactive names

**Definition : Active name**

A name $n$ is **active** in a $\pi$-graph with clock $\kappa$ and partition $\delta$ iff
- either it is instantiated in the graph
- or it is a component of $\kappa$ and $\delta$ (only for fresh outputs)
Garbage collection of inactive names

Definition : Active name

A name \( n \) is active in a \( \pi \)-graph with clock \( \kappa \) and partition \( \delta \) iff

- either it is instantiated in the graph
- or it is a component of \( \kappa \) and \( \delta \) (only for fresh outputs)

Question : how to avoid \( \kappa \) and \( \delta \) to grow infinitely?
Garbage collection of inactive names

Definition: Active name

A name $n$ is **active** in a $\pi$-graph with clock $\kappa$ and partition $\delta$ iff
- either it is instantiated in the graph
- or it is a component of $\kappa$ and $\delta$ (only for fresh outputs)

Question: how to avoid $\kappa$ and $\delta$ to grow infinitely?

Answer: Garbage collection of inactive names

Let $\kappa; \delta \vdash G$ a graph with instantiations $I$ then
$\text{gc}(\pi) \overset{\text{def}}{=} \kappa'; \delta' \vdash G$ such that
$$
\begin{align*}
\gamma' & \overset{\text{def}}{=} \left\{ E \cap (\mathcal{N}_f \cup \mathcal{N}_o \cup \text{cod}(I)) \mid E \in \gamma \right\} \setminus \{\emptyset\} \\
\kappa' & \overset{\text{def}}{=} \left\{ d \mapsto \kappa(d) \cap \text{cod}(I) \mid d \in \text{dom}(\kappa) \land \left( d = \bot \lor d \in \text{cod}(I) \lor \{d\} \notin \gamma' \right) \right\}
\end{align*}
$$
A Decidable Characterization of a Graphical Pi-Calculus with Iterators

A decidable characterization

Illustrating garbage collection

\{ \bot \rightarrow \emptyset \} : \ast \lbrack \overline{c} \langle \nu a \rangle . 0 \rbrack
Illustrating garbage collection

\{\bot \mapsto \emptyset, 1! \mapsto \emptyset\} : \ast [\overline{c}\langle \nu a \mid 1! \rangle].0]
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
A decidable characterization

Illustrating garbage collection

\[ \overline{c} \langle 1! \rangle \rightarrow \{ \bot \mapsto \emptyset, 1! \mapsto \emptyset \} : *[\overline{c} \langle \nu a \mid 1! \rangle].[0] \]
A Decidable Characterization of a Graphical Pi-Calculus with Iterators

Illustrating garbage collection

\[
\overline{c} \langle 1! \rangle \rightarrow \{ \bot \mapsto \emptyset \} : \square[\overline{c} \langle \nu a \rangle.0]
\]
A Decidable Characterization of a Graphical Pi-Calculus with Iterators

Illustrating garbage collection

\[
\overline{c}(1!) \rightarrow \{ \bot \leftrightarrow \emptyset, 1! \leftrightarrow \emptyset \} : *[\overline{c}(\nu a \mid 1!) \cdot 0]
\]
Illustrating garbage collection

\[
\begin{array}{c}
\text{\[c\langle \nu a \mid 1! \}\]} \\
\end{array}
\]
A Decidable Characterization of a Graphical Pi-Calculus with Iterators
A decidable characterization

Illustrating garbage collection

\begin{equation}
\overline{c}\langle 1! \rangle \xrightarrow{\overline{c}\langle 1! \rangle} \{\bot \leftrightarrow \emptyset, 1! \leftrightarrow \emptyset\} : *[\overline{c}\langle \nu a \mid 1! \rangle . \mathbf{0}]. \overline{c}\langle 1! \rangle
\end{equation}
A Decidable Characterization of a Graphical Pi-Calculus with Iterators

Illustrating garbage collection

\[
\begin{array}{c}
\overline{c}(1!) \quad \overline{c}(1!)
\end{array}
\rightarrow
\begin{array}{c}
\{ \bot \leftrightarrow \emptyset, 1! \leftrightarrow \emptyset \} : \ast[\overline{c} \langle \nu a \mid 1! \rangle . \overline{0}] \quad \overline{c}(1!)
\end{array}
\rightarrow
\begin{array}{c}
\overline{c}(1!)
\end{array}
\rightarrow
d \text{etc.}
\]
Decidability results

Finite systems

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

- \( \bigcup_Q \bigcup \text{cod}(\kappa_Q) \subseteq \{1?, 2?, \ldots, |B|?\} \)
  (where \( B \) is the number of “boxes” in the graph)

- \( \bigcup_Q \text{dom}(\kappa_Q) \subseteq \{\perp, 1!, 2!, \ldots, |B|!\} \)
  (the proof for this is more involved)

\[ \implies \text{the sets } Q \text{ and } T \text{ are finite} \]
Decidability results

Finite systems

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

- $\bigcup Q \cup \text{cod}(\kappa_Q) \subseteq \{1?, 2?, \ldots, |B|?\}$
  (where $B$ is the number of “boxes” in the graph)
- $\bigcup Q \cup \text{dom}(\kappa_Q) \subseteq \{\bot, 1!, 2!, \ldots, |B|!\}$
  (the proof for this is more involved)

$\implies$ the sets $Q$ and $T$ are finite

$\implies$ “The” theorem

Bisimilarity for $\pi$-graphs with causal clocks is decidable
Summary

- A visual paradigm *and* a process calculus
- Expressivity of the (finite-control) \(\pi\)-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) \(\pi\)-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 \(\pi\)-calculus
- Translation to (relatively low-level) Petri nets
- Application \(\Rightarrow\) \(\pi\) explorer tool (for now only support the dynamic \(\pi\)-graphs)