A Denotational Study of Mobility

Joël-Alexis Bialkiewicz and Frédéric Peschanski

November 2, 2009

Introduction

Two Main Points of View on Modelling Processes

- Operational POV (π-calculus. . .)
  - Low level, double-edged: easy mobility but difficult to abstract
  - unsettled theory so many variants
  - issues with compositionality: bound prefixes and guards
  - denotations exist but not practical
- Denotational POV (CSP)
  - denotational (tr, fail, div) and compositional by design
  - supports refinement
  - but no easy way to account for mobility

Our Approach: Mobility in a Denotational Way

- Heavily inspired by CSP but integrated model (decorated traces)
- π-like mobility but compositional $\Longrightarrow$ fully denotational model
- Support for refinement

Introduction

Two Main Points of View on Modelling Processes

- Operational POV (π-calculus...)
  - Low level, double-edged: easy mobility but difficult to abstract
  - unsettled theory so many variants
  - issues with compositionality: bound prefixes and guards
  - denotations exist but not practical
- Denotational POV (CSP)
  - denotational (tr, fail, div) and compositional by design
  - supports refinement
  - but no easy way to account for mobility

Our Approach: Mobility in a Denotational Way

- Heavily inspired by CSP but integrated model (decorated traces)
- π-like mobility but compositional $\Longrightarrow$ fully denotational model
- Support for refinement

Outline

Outline

Representing Behaviours

The problem

- Full representation of behaviour? branching structure (LTS)
- Set of process traces: information lost
- Traces + failures,divergences: hard to introduce mobility

What we wanted

- Traces but with as much information as the LTS
- How: link observations to *where* and *when* in LTS
  $\Longrightarrow$ *locations* !
- LTS can be rebuilt from decorated traces: no information lost

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{number\ of\ branches}_{branch\ number}$, weak variants $\tilde{\triangleright}$ and $\tilde{\diamond}^{j}_{i}$

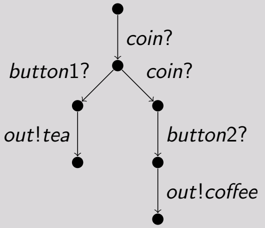
Process (*not* mobile)

$coin?.(button1?.out!tea + coin?.button2?.out!coffee)$

What does this process do?

Behaviour

LTS



Traces

$\{$

$\langle coin?{::}\triangleright, button1?{::}\diamond^2_1, out!tea{::}\triangleright\rangle,$

$\langle coin?{::}\triangleright, coin?{::}\diamond^2_2, button2?{::}\triangleright, out!coffee{::}\triangleright\rangle$

$\}$

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{\textit{number of branches}}_{\textit{branch number}}$, weak variants $\widetilde{\triangleright}$ and $\widetilde{\diamond}^{j}_{i}$

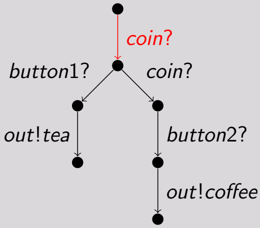
Process (*not* mobile)

$coin?.(button1?.out!tea + coin?.button2?.out!coffee)$

What does this process do?

Behaviour

LTS



Traces

$\{$
$\langle coin?{::}\triangleright, button1?{::}\diamond^2_1, out!tea{::}\triangleright\rangle,$
$\langle coin?{::}\triangleright, coin?{::}\diamond^2_2, button2?{::}\triangleright, out!coffee{::}\triangleright\rangle$
$\}$

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{\textit{number of branches}}_{\textit{branch number}}$, weak variants $\tilde{\triangleright}$ and $\tilde{\diamond}^j_i$

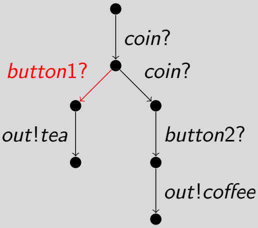
Process (*not* mobile)

$coin?.(button1?.out!tea + coin?.button2?.out!coffee)$

What does this process do?

Behaviour

LTS



Traces

$\{$
$\langle coin?::\triangleright, button1?::\diamond^2_1, out!tea::\triangleright\rangle,$
$\langle coin?::\triangleright, coin?::\diamond^2_2, button2?::\triangleright, out!coffee::\triangleright\rangle$
$\}$

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{number\ of\ branches}_{branch\ number}$, weak variants $\tilde{\triangleright}$ and $\tilde{\diamond}^{j}_{i}$

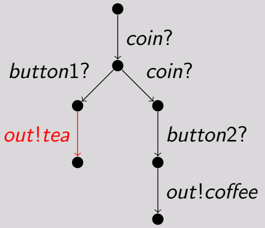
Process (*not* mobile)

$coin?.(button1?.out!tea + coin?.button2?.out!coffee)$

What does this process do?

Behaviour

LTS



Traces

$\{$

$\langle coin?{::}\triangleright, button1?{::}\diamond^2_1, out!tea{::}\triangleright\rangle,$

$\langle coin?{::}\triangleright, coin?{::}\diamond^2_2, button2?{::}\triangleright, out!coffee{::}\triangleright\rangle$

$\}$

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{number\ of\ branches}_{branch\ number}$, weak variants $\tilde{\triangleright}$ and $\tilde{\diamond}^j_i$

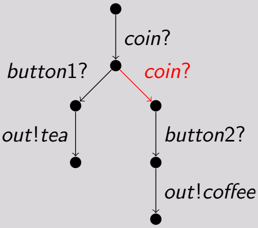
Process (*not* mobile)

$coin?.(button1?.out!tea + coin?.button2?.out!coffee)$

What does this process do?

Behaviour

LTS



Traces

$$\{$$
$$\langle coin?::\triangleright, button1?::\diamond^2_1, out!tea::\triangleright\rangle,$$
$$\langle coin?::\triangleright, coin?::\diamond^2_2, button2?::\triangleright, out!coffee::\triangleright\rangle$$
$$\}$$

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{number\ of\ branches}_{branch\ number}$, weak variants $\tilde{\triangleright}$ and $\tilde{\diamond}^j_i$

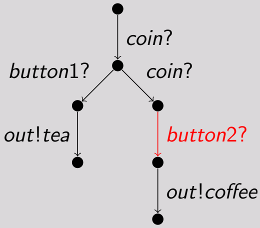
Process (*not* mobile)

coin?.(*button*1?.*out*!*tea* + *coin*?.*button*2?.*out*!*coffee*)

What does this process do?

Behaviour

LTS



Traces

$\{$
$\langle coin?::\triangleright, button1?::\diamond^2_1, out!tea::\triangleright\rangle,$
$\langle coin?::\triangleright, coin?::\diamond^2_2, button2?::\triangleright, out!coffee::\triangleright\rangle$
$\}$

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{\textit{number of branches}}_{\textit{branch number}}$, weak variants $\tilde{\triangleright}$ and $\tilde{\diamond}^{j}_{i}$

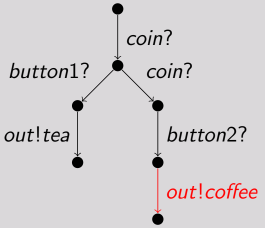
Process (*not* mobile)

$coin?.(button1?.out!tea + coin?.button2?.out!coffee)$

What does this process do?

Behaviour

LTS



Traces

$$
\{ \\
\langle coin?{::}\triangleright, button1?{::}\diamond^2_1, out!tea{::}\triangleright\rangle, \\
\langle coin?{::}\triangleright, coin?{::}\diamond^2_2, button2?{::}\triangleright, out!coffee{::}\triangleright\rangle \\
\}
$$

Basic Example

The basics

Observation (::location): input *channel*? output *channel*!*value*, or ✓

Location: origin: ϵ, next: $\triangleright$ and choice: $\diamond^{\textit{number of branches}}_{\textit{branch number}}$, weak variants $\tilde{\triangleright}$ and $\tilde{\diamond}^j_i$

Process (*not* mobile)

$coin?.(button1?.out!tea + coin?.button2?.out!coffee)$

Which locations why? What is an absolute location?

Behaviour

LTS

$$\begin{tikzcd}
 & \epsilon \arrow[d, "coin?"] & \\
 & \triangleright \arrow[dl, "button1?"'] \arrow[dr, "coin?"] & \\
\diamond^2_1 \arrow[d, "out!tea"'] & & \diamond^2_2 \arrow[d, "button2?"] \\
\triangleright & & \triangleright \arrow[d, "out!coffee"] \\
 & & \triangleright
\end{tikzcd}$$

Traces

$$\{$$
$$\langle coin?::\triangleright, button1?::\diamond^2_1, out!tea::\triangleright\rangle,$$
$$\langle coin?::\triangleright, coin?::\diamond^2_2, button2?::\triangleright, out!coffee::\triangleright\rangle$$
$$\}$$

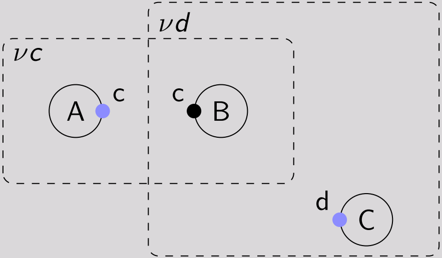
Outline

About Mobility

Physical vs Logical Mobility

A process is mobile if it changes neighbours

How Can a Process Change Neighbours?

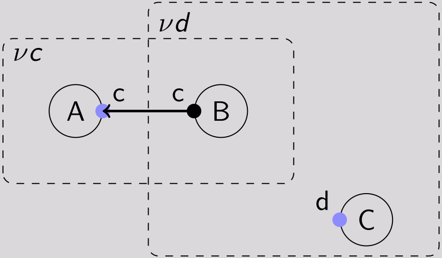


About Mobility

Physical vs Logical Mobility

A process is mobile if it changes neighbours

How Can a Process Change Neighbours?

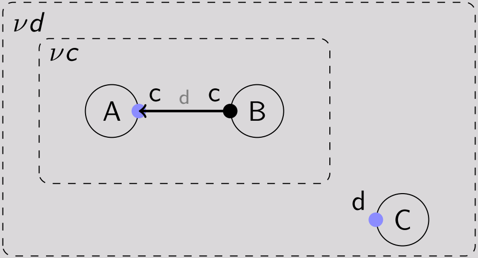


About Mobility

Physical vs Logical Mobility

A process is mobile if it changes neighbours

How Can a Process Change Neighbours?

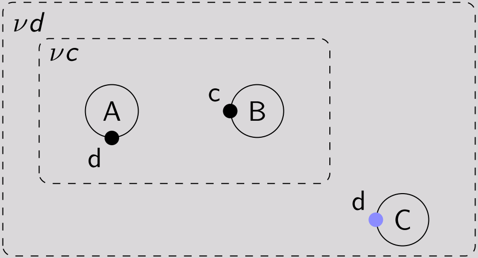


About Mobility

Physical vs Logical Mobility

A process is mobile if it changes neighbours

How Can a Process Change Neighbours?

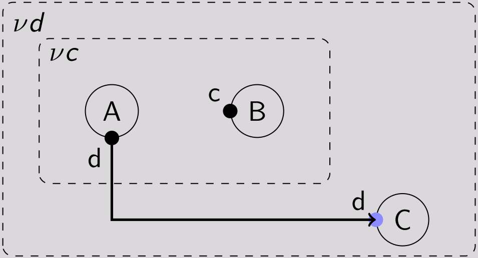


About Mobility

Physical vs Logical Mobility

A process is mobile if it changes neighbours

How Can a Process Change Neighbours?



The Modeling Problems

Two main problems

- Binders
- Guards

Binders in mobile languages

- Binders: dynamic names (escaped names and inputs)
- π-calculus operational, mixes free and bound names
- Solution: binders are uniquely identified by when/where created
- advantage: fresh by construction, avoid α-conversion issues

Guards

- Reminder: $[\varphi]P$ means if φ then P
- Not observations, but necessary for compos. Where do they go?
- Solution: in locations

Example 2

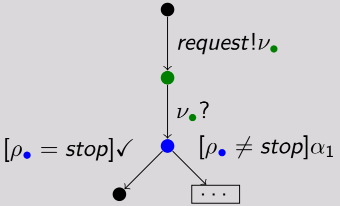
Process with guards and extrusion

$$(\nu\ in)request!in.in?out.$$
$$([out = stop]SKIP + [out \neq stop]Communicate(in, out))$$

What does this process do?

Behaviour

LTS



Traces

$$\{$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \checkmark::(\rho_{\epsilon\triangleright\triangleright} = stop, \diamond^2_1)\rangle,$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \alpha_1::(\rho_{\epsilon\triangleright\triangleright} \neq stop, \diamond^2_2), \ldots\rangle,$$
$$\ldots$$
$$\}$$

Example 2

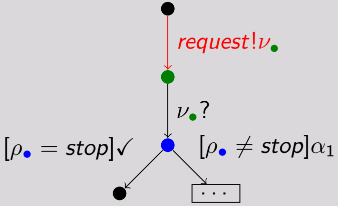
Process with guards and extrusion

$(\nu\ in)request!in.in?out.$
$([out = stop]SKIP + [out \neq stop]Communicate(in, out))$

Extruded names: ν_{where}

Behaviour

LTS



Traces

$\{$
$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \checkmark::(\rho_{\epsilon\triangleright\triangleright} = stop, \diamond_1^2)\rangle,$
$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \alpha_1::(\rho_{\epsilon\triangleright\triangleright} \neq stop, \diamond_2^2), \dots\rangle,$
$\dots$
$\}$

Example 2

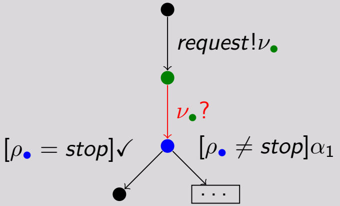
Process with guards and extrusion

$$(\nu\ in)request!in.in?out.$$
$$([out = stop]SKIP + [out \neq stop]Communicate(in, out))$$

Input *observations* have no object; received names: ρ_{where}

Behaviour

LTS



Traces

$$\{$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \checkmark::(\rho_{\epsilon\triangleright\triangleright} = stop, \diamond_1^2)\rangle,$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \alpha_1::(\rho_{\epsilon\triangleright\triangleright} \neq stop, \diamond_2^2), \ldots\rangle,$$
$$\ldots$$
$$\}$$

Example 2

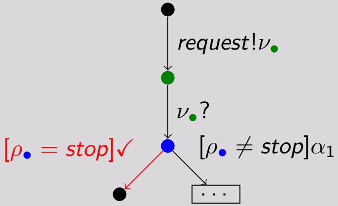
Process with guards and extrusion

$$(\nu\ in)request!in.in?out.$$
$$([out = stop]SKIP + [out \neq stop]Communicate(in, out))$$

In traces the guard of an observation prefixes its location

Behaviour

LTS



Traces

$$\{$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \checkmark::(\rho_{\epsilon\triangleright\triangleright} = stop, \diamond_1^2)\rangle,$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \alpha_1::(\rho_{\epsilon\triangleright\triangleright} \neq stop, \diamond_2^2), \ldots\rangle,$$
$$\ldots$$
$$\}$$

Example 2

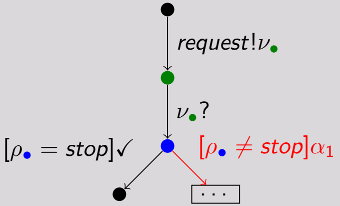
Process with guards and extrusion

$$(\nu\ in)request!in.in?out.$$
$$([out = stop]SKIP + [out \neq stop]Communicate(in, out))$$

In traces the guard of an observation prefixes its location

Behaviour

LTS



Traces

$$\{$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \checkmark::(\rho_{\epsilon\triangleright\triangleright} = stop, \diamond_1^2)\rangle,$$
$$\langle request!\nu_{\epsilon\triangleright}::\triangleright, \nu_{\epsilon\triangleright}?::\triangleright, \alpha_1::(\rho_{\epsilon\triangleright\triangleright} \neq stop, \diamond_2^2), \ldots\rangle,$$
$$\ldots$$
$$\}$$

Outline

1. Basics: Locations
2. Mobility
3. Equivalence & Refinement

Equivalence and Normal Forms

Dealing with redundancy

- Problem: model very fine-grained
- Solution: rewrite rules to trim redundancy

Theorem

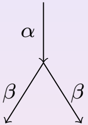
Let T be a trace set. Suppose T_1 and T_2 such that $T \longrightarrow^* T_1 \not\rightarrow$ and $T \longrightarrow^* T_2 \not\rightarrow$. Then $T_1 = T_2 = \widehat{T}$.

Interest

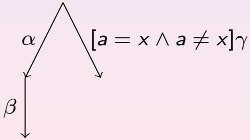
- Equivalence checking: normalise then test isomorphism
- Much simpler than existing equivalence checking for mobility
- Only possible because no binders in semantic

Example

$$P = \alpha.(\beta + \beta)$$
$$traces(P) = \{\langle \alpha{::}\triangleright, \beta{::}\diamond^2_1\rangle, \langle \alpha{::}\triangleright, \beta{::}\diamond^2_2\rangle\}$$

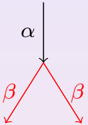


$$Q = \alpha.\beta + [a = x \wedge a \neq x]\gamma$$
$$traces(Q) = \{\langle \alpha{::}\diamond^2_1, \beta{::}\triangleright\rangle, \langle \gamma{::}(a = x \wedge a \neq x, \diamond^2_2)\rangle\}$$

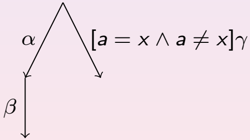


Example

$$P = \alpha.(\beta + \beta)$$
$$traces(P) = \{\langle \alpha{::}\triangleright, \beta{::}\diamond^2_1\rangle, \langle \alpha{::}\triangleright, \beta{::}\diamond^2_2\rangle\}$$



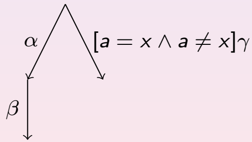
$$Q = \alpha.\beta + [a = x \wedge a \neq x]\gamma$$
$$traces(Q) = \{\langle \alpha{::}\diamond^2_1, \beta{::}\triangleright\rangle, \langle \gamma{::}(a = x \wedge a \neq x, \diamond^2_2)\rangle\}$$



Example

$$P = \alpha.(\beta + \beta)$$
$$traces(P) = \{\langle \alpha{::}\triangleright, \beta{::}\diamond^2_1\rangle, \langle \alpha{::}\triangleright, \beta{::}\diamond^2_2\rangle\}$$
$$traces(P) \xrightarrow{merge} \{\langle \alpha{::}\triangleright, \beta{::}\triangleright\rangle\}$$

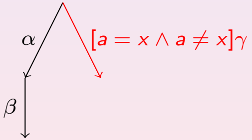
$$Q = \alpha.\beta + [a = x \wedge a \neq x]\gamma$$
$$traces(Q) = \{\langle \alpha{::}\diamond^2_1, \beta{::}\triangleright\rangle, \langle \gamma{::}(a = x \wedge a \neq x, \diamond^2_2)\rangle\}$$



Example

$$P = \alpha.(\beta + \beta)$$
$$traces(P) = \{\langle \alpha{::}\triangleright, \beta{::}\diamond_1^2\rangle, \langle \alpha{::}\triangleright, \beta{::}\diamond_2^2\rangle\}$$
$$traces(P) \xrightarrow{merge} \{\langle \alpha{::}\triangleright, \beta{::}\triangleright\rangle\}$$

$$Q = \alpha.\beta + [a = x \wedge a \neq x]\gamma$$
$$traces(Q) = \{\langle \alpha{::}\diamond_1^2, \beta{::}\triangleright\rangle, \langle \gamma{::}(a = x \wedge a \neq x, \diamond_2^2)\rangle\}$$
$$traces(Q) \xrightarrow{false} \{\langle \alpha{::}\triangleright, \beta{::}\triangleright\rangle\}$$



Example

$$\begin{tikzcd} {} \arrow[d, "\alpha"'] \\ {} \arrow[d, "\beta"'] \\ {} \end{tikzcd}$$

$$P = \alpha.(\beta + \beta)$$
$$traces(P) = \{\langle \alpha{::}\triangleright, \beta{::}\diamond^2_1\rangle, \langle \alpha{::}\triangleright, \beta{::}\diamond^2_2\rangle\}$$
$$traces(P) \xrightarrow{merge} \{\langle \alpha{::}\triangleright, \beta{::}\triangleright\rangle\}$$

$$\begin{tikzcd} {} \arrow[d, "\alpha"'] \\ {} \arrow[d, "\beta"'] \\ {} \end{tikzcd}$$

$$Q = \alpha.\beta + [a = x \wedge a \neq x]\gamma$$
$$traces(Q) = \{\langle \alpha{::}\diamond^2_1, \beta{::}\triangleright\rangle, \langle \gamma{::}(a = x \wedge a \neq x, \diamond^2_2)\rangle\}$$
$$traces(Q) \xrightarrow{false} \{\langle \alpha{::}\triangleright, \beta{::}\triangleright\rangle\}$$

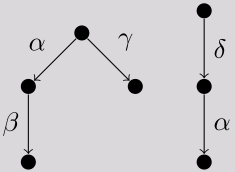
Delayed Sum

What is the delayed sum?

- The way to refinement
- Strict generalisation of process sum
- Grafting any behaviour anywhere in branching structure
- Two parameters: a location and a substitution from symbols to special names

Delayed Sum Example

$$P \overset{\text{def}}{=} \alpha.\beta + \gamma \qquad Q \overset{\text{def}}{=} \delta.\alpha \qquad P +^{Id}_{\epsilon\diamond^{2}_{1}} Q = \alpha.(\beta + \delta.\alpha) + \gamma$$



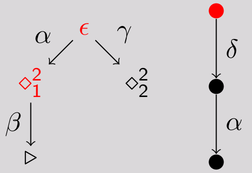
Delayed Sum

What is the delayed sum?

- The way to refinement
- Strict generalisation of process sum
- Grafting any behaviour anywhere in branching structure
- Two parameters: a location and a substitution from symbols to special names

Delayed Sum Example

$$P \stackrel{\text{def}}{=} \alpha.\beta + \gamma \qquad Q \stackrel{\text{def}}{=} \delta.\alpha \qquad P +^{Id}_{\epsilon\diamond^2_1} Q = \alpha.(\beta + \delta.\alpha) + \gamma$$



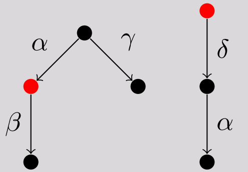
Delayed Sum

What is the delayed sum?

- The way to refinement
- Strict generalisation of process sum
- Grafting any behaviour anywhere in branching structure
- Two parameters: a location and a substitution from symbols to special names

Delayed Sum Example

$$P \stackrel{\text{def}}{=} \alpha \bullet \beta + \gamma \qquad Q \stackrel{\text{def}}{=} \delta.\alpha \qquad P +^{Id}_{\epsilon\diamond_1^2} Q = \alpha.(\beta + \delta.\alpha) + \gamma$$



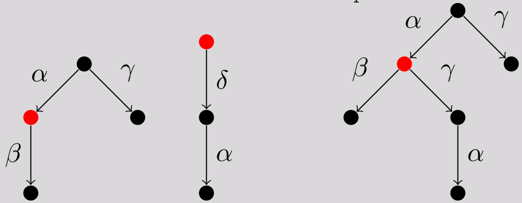
Delayed Sum

What is the delayed sum?

- The way to refinement
- Strict generalisation of process sum
- Grafting any behaviour anywhere in branching structure
- Two parameters: a location and a substitution from symbols to special names

Delayed Sum Example

$$P \overset{\mathrm{def}}{=} \alpha.\beta + \gamma \qquad Q \overset{\mathrm{def}}{=} \delta.\alpha \qquad P +^{Id}_{\epsilon\diamond^2_1} Q = \alpha.(\beta + \delta.\alpha) + \gamma$$



Refinement

Definition

$P \sqsubseteq Q \iff \exists \mathcal{R}\mathcal{L} = \bigcup_{i=1}^{n} \{(R_i, l_i, \sigma_i)\}$ s. t.
$P =_\diamond Q +_{l_1}^{\sigma_1} R_1 \ldots +_{l_n}^{\sigma_n} R_n$

Why?

- Refinement relation nearly for free
- The delayed sum cannot be compositional... is refinement?

Conclusion

What we did

- CSP vs π-calculus: a step towards bridging the gap
- Denotational theory for mobility with intuitive refinement
- Operational semantics w/o π-calculus pitfalls
- Axiomatic semantics
- A Hoare-like logic [LAM09]

What next?

- Finish writing the thesis...
- Proving that refinement is compositional
- Equivalence/refinement checking algorithm