On the Relationship between π-calculus and Finite Place/Transition Petri Nets

Roland Meyer[1] Roberto Gorrieri[2]

[1]LIAFA, University Paris 7 & CNRS

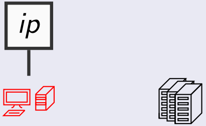
[2]University of Bologna

CONCUR Conference, 2009

A Client-Server System in π-calculus

Client sends on public channel *url* his private address *ip* to server

Graphically

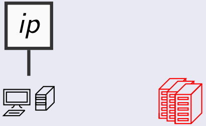


In π-calculus

A Client-Server System in π-calculus

Client sends on public channel *url* his private address *ip* to server

Graphically

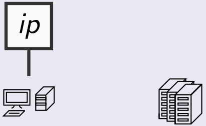


In π-calculus

A Client-Server System in π-calculus

Client sends on public channel *url* his private address *ip* to server

Graphically



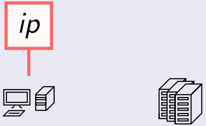
In π-calculus

$$\nu ip.\overline{url}\langle ip\rangle.ip(x).C\lfloor url, ip\rfloor \mid$$
$$url(y).(\overline{y}\langle dat\rangle \mid S\lfloor url, dat\rfloor)$$

A Client-Server System in π-calculus

Client sends on public channel *url* his private address *ip* to server

Graphically



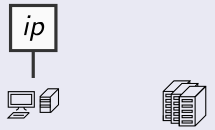
In π-calculus

$$\nu ip.\overline{url}\langle ip\rangle.ip(x).C\lfloor url, ip\rfloor \mid$$
$$url(y).(\overline{y}\langle dat\rangle \mid S\lfloor url, dat\rfloor)$$

A Client-Server System in π-calculus

In response server spawns a new thread

Graphically



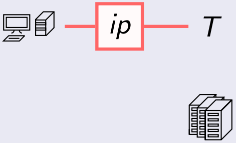
In π-calculus

$$\nu ip.\overline{url}\langle ip\rangle.ip(x).C\lfloor url, ip\rfloor \mid$$
$$url(y).(\ \overline{y}\langle dat\rangle \mid S\lfloor url, dat\rfloor)$$

A Client-Server System in π-calculus

Thread sends on the private channel *ip* data *dat* to the client

Graphically



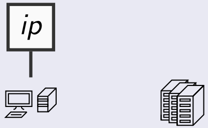
In π-calculus

$$\nu ip.(ip(x).C\lfloor url, ip\rfloor \mid \overline{ip}\langle dat\rangle) \mid S\lfloor url, dat\rfloor$$

A Client-Server System in π-calculus

Thread terminates, client is ready to contact server again

Graphically



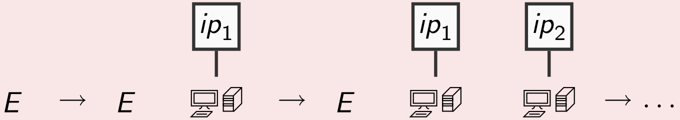
In π-calculus

$$\nu ip.C\lfloor url, ip\rfloor \mid S\lfloor url, dat\rfloor$$

A Client-Server System in π-calculus

Assumption

Environment E generates clients



Contribution

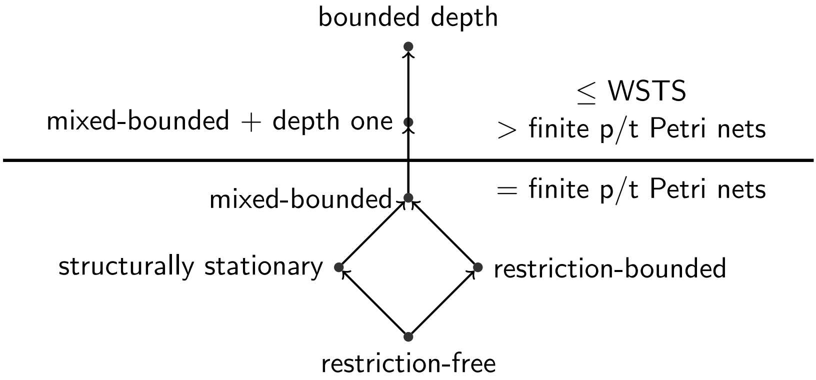
Problem under Study

- Goal: Automatically verify mobile systems
- Approach: Translate system to automata-theoretic model
- Question: When are finite p/t nets sufficient?

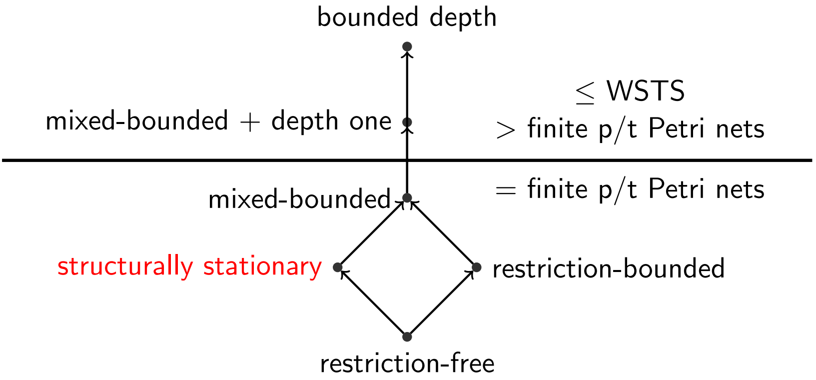
Quality Criteria

- Bisimilarity: $\mathcal{T}(P) \approx \mathcal{T}(\mathcal{N}[\![P]\!])$
- Finiteness: $\mathcal{N}[\![P]\!]$ finite iff ...
- Expressiveness: Unbounded concurrency and restrictions

A Hierarchy of Process Classes



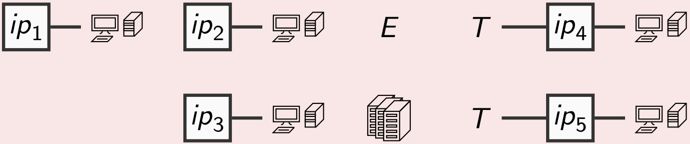
A Hierarchy of Process Classes



Idea of Structural Semantics

Problem

Unbounded number of clients and threads



Observation

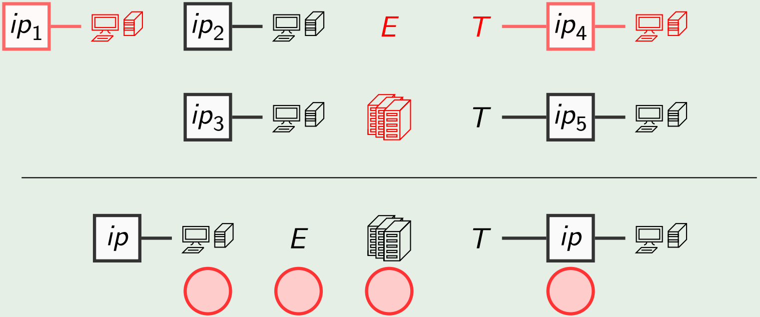
Finite number of connection patterns

Idea of Structural Semantics

Represent Connections in a Petri Net

- Connection patterns yield places

Example

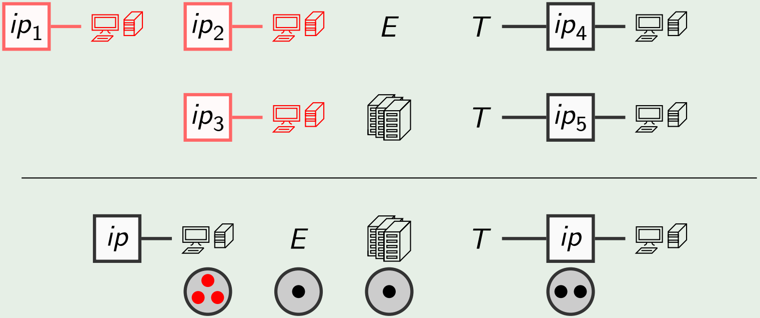


Idea of Structural Semantics

Represent Connections in a Petri Net

- Connection patterns yield places
- Occurence of a pattern yields a token

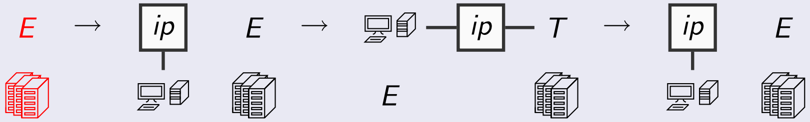
Example



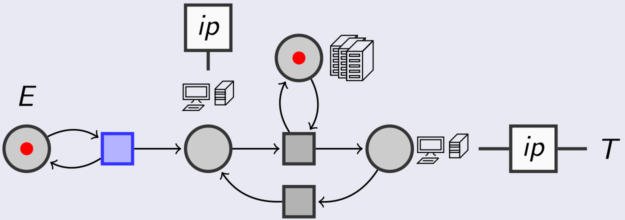
Idea of Structural Semantics

Transitions model evolution of patterns

In π-calculus



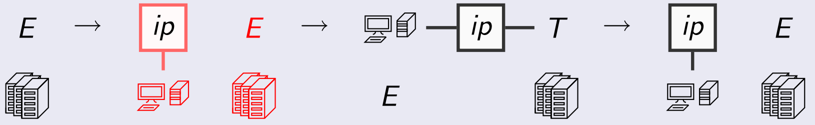
Structural Semantics



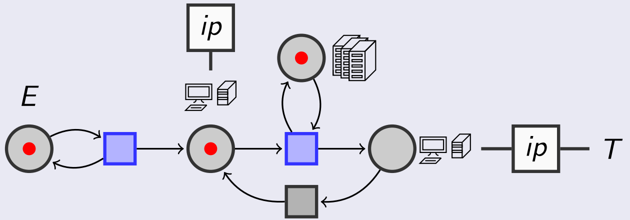
Idea of Structural Semantics

Transitions model evolution of patterns

In π-calculus



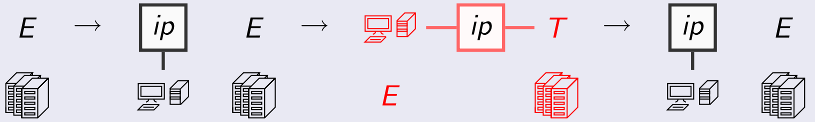
Structural Semantics



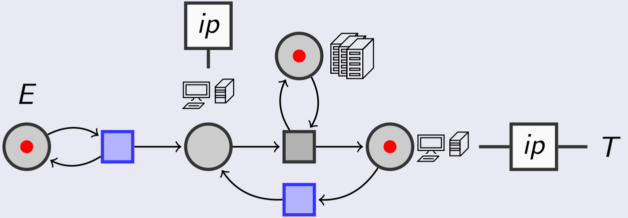
Idea of Structural Semantics

Transitions model evolution of patterns

In π-calculus



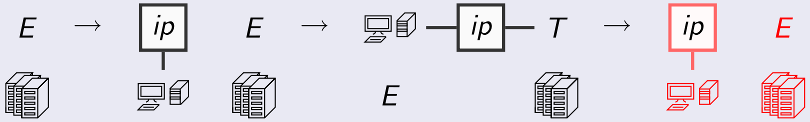
Structural Semantics



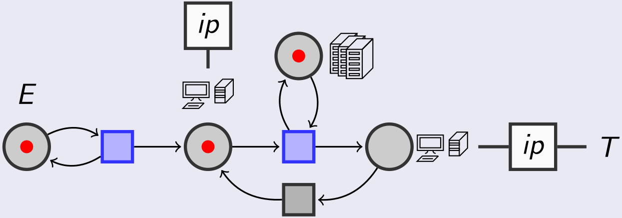
Idea of Structural Semantics

Transitions model evolution of patterns

In π-calculus



Structural Semantics



Restricted Form of Processes

Formalise idea of connection patterns

Example (Restricted Form)

Minimise scopes of restricted names

$$\nu ip.(\ ip(x).C\lfloor url, ip\rfloor \mid \overline{ip}\langle dat\rangle \mid S\lfloor url, dat\rfloor\ )$$

Restricted Form of Processes

Formalise idea of connection patterns

Example (Restricted Form)

Minimise scopes of restricted names

$$\nu ip.(\, ip(x).C\lfloor url, ip \rfloor \mid \overline{ip}\langle dat \rangle \mid S\lfloor url, dat \rfloor \,)$$

$$\equiv \nu ip.(\, ip(x).C\lfloor url, ip \rfloor \mid \overline{ip}\langle dat \rangle \,) \mid S\lfloor url, dat \rfloor$$

Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

$$\nu ip.(ip(x).C\lfloor url, ip\rfloor \mid \overline{ip}\langle dat\rangle) \mid S\lfloor url, dat\rfloor$$

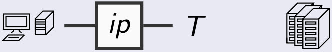
Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

$$\nu ip.(ip(x).C\lfloor url, ip\rfloor \mid \overline{ip}\langle dat\rangle) \mid S\lfloor url, dat\rfloor$$

Fragments correspond to connection patterns



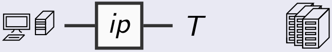
Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

$$\nu ip.(ip(x).C\lfloor url, ip\rfloor \mid \overline{ip}\langle dat\rangle) \mid S\lfloor url, dat\rfloor$$

Fragments correspond to connection patterns



Lemma (Finiteness)

$\mathcal{N}_{\mathcal{S}}[\![P]\!]$ finite iff there is a finite set of fragments every reachable process consists of

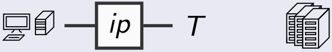
Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

$$\nu ip.(ip(x).C\lfloor url, ip\rfloor \mid \overline{ip}\langle dat\rangle) \mid S\lfloor url, dat\rfloor$$

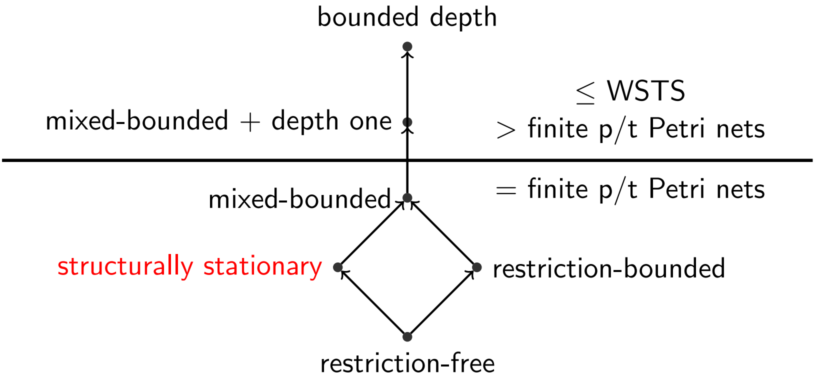
Fragments correspond to connection patterns



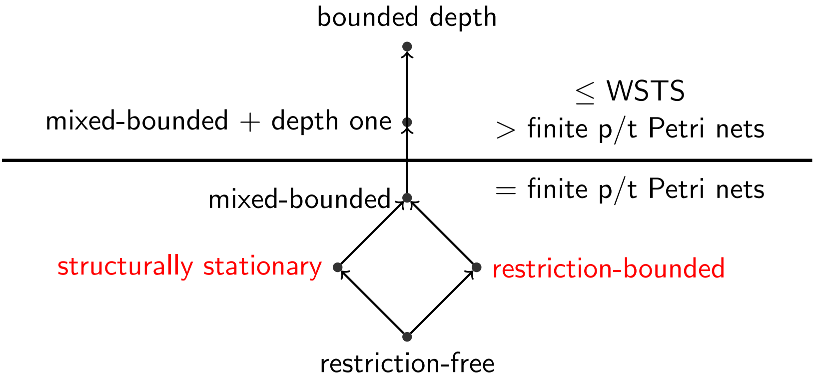
Lemma (Finiteness)

$\mathcal{N}_{\mathcal{S}}[\![P]\!]$ finite iff there is a finite set of fragments every reachable process consists of (structural stationarity)

A Hierarchy of Process Classes

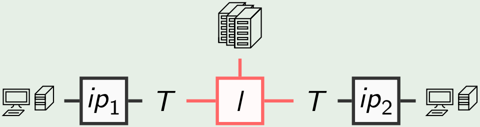


A Hierarchy of Process Classes



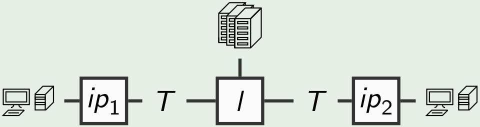
A Modified Server

Server Maintains Control Channel with Threads

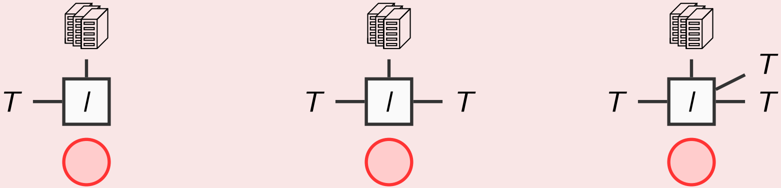


A Modified Server

Server Maintains Control Channel with Threads



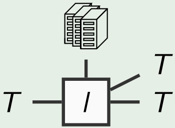
Structural Semantics Infinite



Idea of Concurrency Semantics

- Treat restricted names as free names

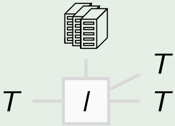
Example



Idea of Concurrency Semantics

- Treat restricted names as free names

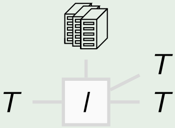
Example

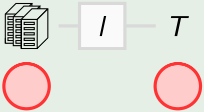


Idea of Concurrency Semantics

- Treat restricted names as free names
- Count number of sequential processes

Example

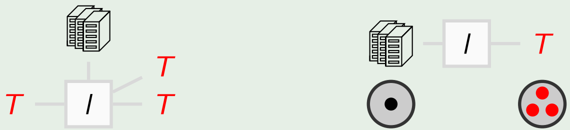




Idea of Concurrency Semantics

- Treat restricted names as free names
- Count number of sequential processes

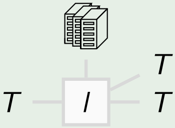
Example

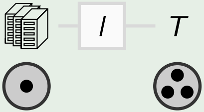


Idea of Concurrency Semantics

- Treat restricted names as free names
- Count number of sequential processes

Example





Crucial Technicality

Preserve order in which free names are generated

Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices

$$\nu z_{\mathbf{0}}.P$$

Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices $\nu z_{\mathbf{0}}.P$
- Process P is in standard form $sf(P)$

$$\overline{a} \mid \nu z_{\mathbf{0}}.(\, \overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor \,)$$

Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices $\nu z_{\mathbf{0}}.P$
- Process P is in standard form $sf(P)$

$$
\begin{aligned}
& \overline{a} \mid \nu z_{\mathbf{0}}.(\, \overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor \,) \\
\equiv\ & \nu z_{\mathbf{0}}.(\, \overline{a} \mid \overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor \,)
\end{aligned}
$$

Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices $\nu z_{\mathbf{0}}.P$
- Process P is in standard form $sf(P)$

Definition (Name-Aware Process)

- Idea: Store generated restrictions syntactically
- Technically: Pair $(P^{\neq\nu}, \tilde{a})$

Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices $\nu z_{\mathbf{0}}.P$
- Process P is in standard form $sf(P)$

Definition (Name-Aware Process)

- Idea: Store generated restrictions syntactically
- Technically: Pair $(P^{\neq\nu}, \tilde{a})$

Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices $\nu z_{\mathbf{0}}.P$
- Process P is in standard form $sf(P)$

Definition (Name-Aware Process)

- Idea: Store generated restrictions syntactically
- Technically: Pair $(P^{\neq\nu}, \tilde{a})$

Example

$$\nu z_{\mathbf{0}}.(\overline{a} \mid \overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor) \quad \rightsquigarrow \quad (\overline{a} \mid \overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor, \quad \{z_{\mathbf{0}}\})$$

Name-Aware Reaction Relation

Example

Idea: Generate names by incrementing indices

Name-Aware Reaction Relation

Example

Idea: Generate names by incrementing indices

$$K\lfloor a\rfloor \rightarrow \nu z.(\overline{z} \mid a.z.K\lfloor a\rfloor)$$

Name-Aware Reaction Relation

Example

Idea: Generate names by incrementing indices

$$K\lfloor a\rfloor \rightarrow \nu z.(\overline{z} \mid a.z.K\lfloor a\rfloor)$$

$$\rightsquigarrow$$

$$(K\lfloor a\rfloor, \quad \{z_0\}) \rightarrow^{na} (\overline{z_{\mathbf{1}}} \mid a.z_{\mathbf{1}}.K\lfloor a\rfloor, \quad \{z_{\mathbf{0}}, z_{\mathbf{1}}\})$$

Name-Aware Reaction Relation

Example

Idea: Generate names by incrementing indices

$$K\lfloor a \rfloor \rightarrow \nu z.(\overline{z} \mid a.z.K\lfloor a \rfloor)$$

$$\rightsquigarrow$$

$$(K\lfloor a \rfloor, \quad \{z_0\}) \rightarrow^{na} (\overline{z_{\mathbf{1}}} \mid a.z_{\mathbf{1}}.K\lfloor a \rfloor, \quad \{z_{\mathbf{0}}, z_{\mathbf{1}}\})$$

Definition (Name-Aware Reaction Relation)

$$(P^{\neq\nu}, \tilde{a}) \rightarrow^{na} (Q^{\neq\nu}, \tilde{a} \uplus \tilde{b}) \quad \text{iff} \quad P^{\neq\nu} \rightarrow \nu\tilde{b}.Q^{\neq\nu} \text{ in sf}$$

Indices in $\tilde{b}$ incremented

Bisimlarity

Lemma (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}_{na}(P^{\neq\nu}, \tilde{a})$ *with* $sf(P) = \nu\tilde{a}.P^{\neq\nu}$

Example $\overline{a} \mid K\lfloor a \rfloor$ **with** $K(a) = \nu z_{\mathbf{0}}.(\overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor)$

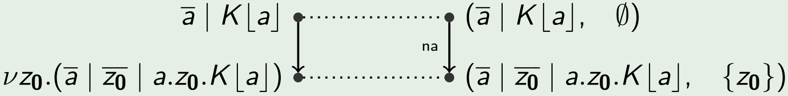
$\overline{a} \mid K\lfloor a \rfloor \bullet \cdots\cdots \bullet (\overline{a} \mid K\lfloor a \rfloor, \quad \emptyset)$

Bisimlarity

Lemma (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}_{na}(P^{\neq\nu}, \tilde{a})$ *with* $sf(P) = \nu\tilde{a}.P^{\neq\nu}$

Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_{\mathbf{0}}.(\overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor)$

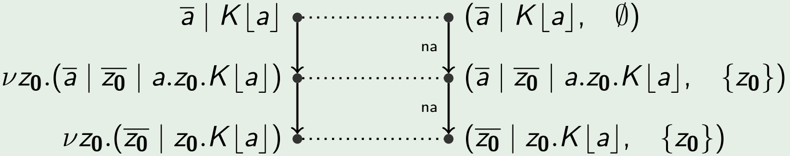


Bisimlarity

Lemma (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}_{na}(P^{\neq\nu}, \tilde{a})$ *with* $sf(P) = \nu\tilde{a}.P^{\neq\nu}$

Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_{\mathbf{0}}.(\overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor)$

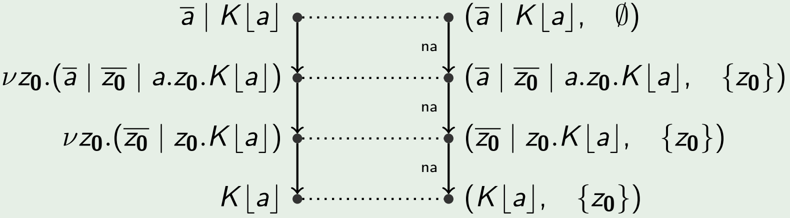


Bisimlarity

Lemma (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}_{na}(P^{\neq\nu}, \tilde{a})$ *with* $sf(P) = \nu\tilde{a}.P^{\neq\nu}$

Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_{\mathbf{0}}.(\overline{z_{\mathbf{0}}} \mid a.z_{\mathbf{0}}.K\lfloor a \rfloor)$



Bisimlarity

Lemma (Bisimilarity)

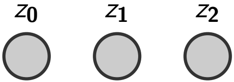
$\mathcal{T}(P) \approx \mathcal{T}_{na}(P^{\neq\nu}, \tilde{a})$ *with* $sf(P) = \nu\tilde{a}.P^{\neq\nu}$

Example $\overline{a} \mid K\lfloor a\rfloor$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K\lfloor a\rfloor)$

$\overline{a} \mid K\lfloor a\rfloor$	$(\overline{a} \mid K\lfloor a\rfloor, \quad \emptyset)$
↓	↓ na
$\nu z_0.(\overline{a} \mid \overline{z_0} \mid a.z_0.K\lfloor a\rfloor)$	$(\overline{a} \mid \overline{z_0} \mid a.z_0.K\lfloor a\rfloor, \quad \{z_0\})$
↓	↓ na
$\nu z_0.(\overline{z_0} \mid z_0.K\lfloor a\rfloor)$	$(\overline{z_0} \mid z_0.K\lfloor a\rfloor, \quad \{z_0\})$
↓	↓ na
$K\lfloor a\rfloor$	$(K\lfloor a\rfloor, \quad \{z_0\})$
↓	↓ na
$\nu z_0.(\overline{z_0} \mid a.z_0.K\lfloor a\rfloor)$	$(\overline{z_1} \mid a.z_1.K\lfloor a\rfloor, \quad \{z_0, z_1\})$

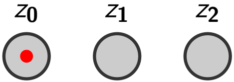
Definition of Concurrency Semantics

- Reachable names (+1) yield name places



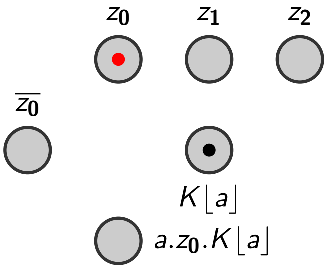
Definition of Concurrency Semantics

- Reachable names (+1) yield name places
- Next name to be generated is marked



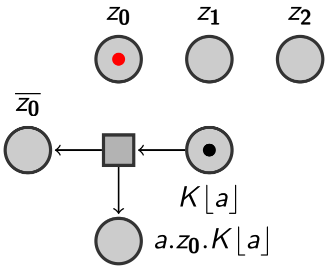
Definition of Concurrency Semantics

- Reachable names (+1) yield name places
- Next name to be generated is marked
- Reachable processes yield process places



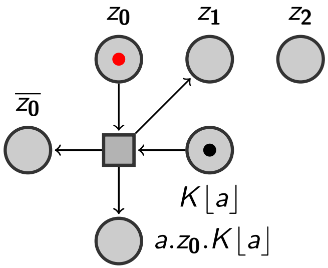
Definition of Concurrency Semantics

- Reachable names (+1) yield name places
- Next name to be generated is marked
- Reachable processes yield process places
- Transitions imitate reactions



Definition of Concurrency Semantics

- Reachable names (+1) yield name places
- Next name to be generated is marked
- Reachable processes yield process places
- Transitions imitate reactions and move name tokens

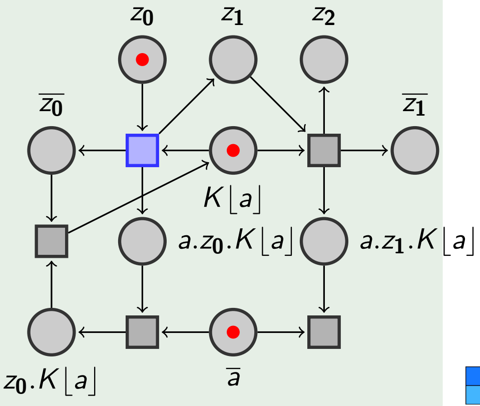


Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K\lfloor a \rfloor)$

Name-Aware TS

- $(\overline{a} \mid K\lfloor a \rfloor, \emptyset)$

Concurrency Semantics



Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K\lfloor a \rfloor)$

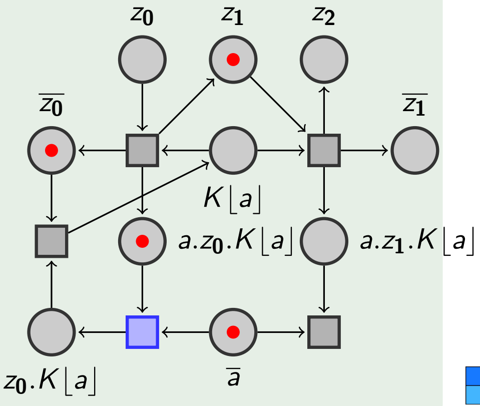
Name-Aware TS

$(\overline{a} \mid K\lfloor a \rfloor, \emptyset)$

$\downarrow$ na

$(\overline{a} \mid \overline{z_0} \mid a.z_0.K\lfloor a \rfloor, \{z_0\})$

Concurrency Semantics



Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K\lfloor a \rfloor)$

Name-Aware TS

$(\overline{a} \mid K\lfloor a \rfloor, \emptyset)$

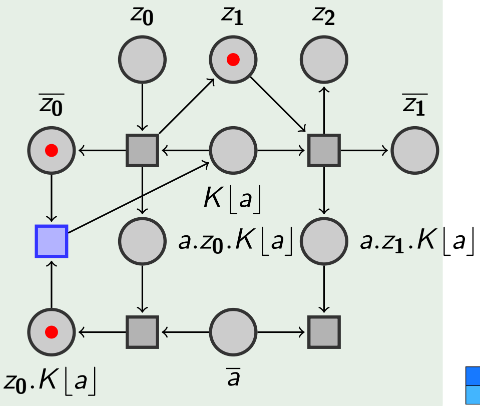
na ↓

$(\overline{a} \mid \overline{z_0} \mid a.z_0.K\lfloor a \rfloor, \{z_0\})$

na ↓

$(\overline{z_0} \mid z_0.K\lfloor a \rfloor, \{z_0\})$

Concurrency Semantics



Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K\lfloor a \rfloor)$

Name-Aware TS

$(\overline{a} \mid K\lfloor a \rfloor, \emptyset)$

na ↓

$(\overline{a} \mid \overline{z_0} \mid a.z_0.K\lfloor a \rfloor, \{z_0\})$

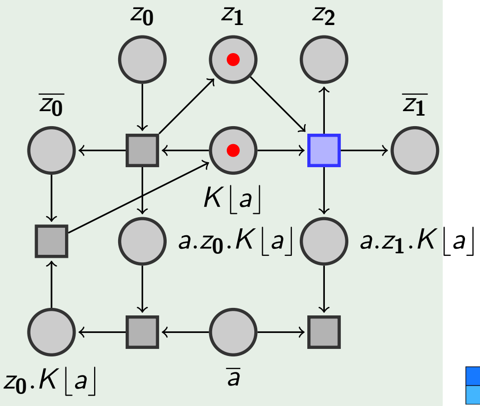
na ↓

$(\overline{z_0} \mid z_0.K\lfloor a \rfloor, \{z_0\})$

na ↓

$(K\lfloor a \rfloor, \{z_0\})$

Concurrency Semantics



Example $\overline{a} \mid K\lfloor a \rfloor$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K\lfloor a \rfloor)$

Name-Aware TS

$(\overline{a} \mid K\lfloor a \rfloor, \emptyset)$

na ↓

$(\overline{a} \mid \overline{z_0} \mid a.z_0.K\lfloor a \rfloor, \{z_0\})$

na ↓

$(\overline{z_0} \mid z_0.K\lfloor a \rfloor, \{z_0\})$

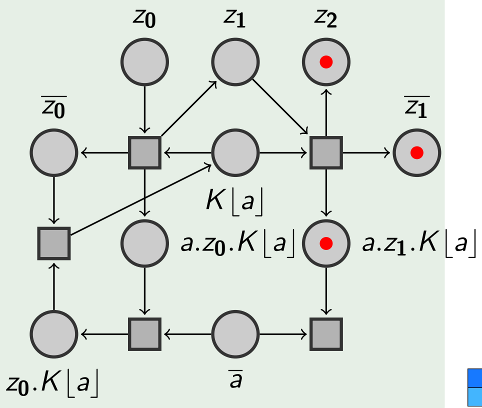
na ↓

$(K\lfloor a \rfloor, \{z_0\})$

na ↓

$(\overline{z_1} \mid a.z_1.K\lfloor a \rfloor, \{z_0, z_1\})$

Concurrency Semantics



Properties of Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}(\mathcal{N}_{\mathcal{C}}[\![P]\!])$

Properties of Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}(\mathcal{N}_{\mathcal{C}}[\![P]\!])$

Theorem (Finiteness)

$\mathcal{N}_{\mathcal{C}}[\![P]\!]$ finite iff P generates finitely many restricted names

Properties of Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}(\mathcal{N}_{\mathcal{C}}[\![P]\!])$

Theorem (Finiteness)

$\mathcal{N}_{\mathcal{C}}[\![P]\!]$ finite iff P generates finitely many restricted names (restriction boundedness)

Properties of Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}(\mathcal{N}_{\mathcal{C}}[\![P]\!])$

Theorem (Finiteness)

$\mathcal{N}_{\mathcal{C}}[\![P]\!]$ finite iff P generates finitely many restricted names (restriction boundedness)

Proof Idea $\Leftarrow$: Construct Process Places from Initial Process

- Removing prefixes yields finite set of derivatives

Properties of Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{T}(\mathcal{N}_{\mathcal{C}}[\![P]\!])$

Theorem (Finiteness)

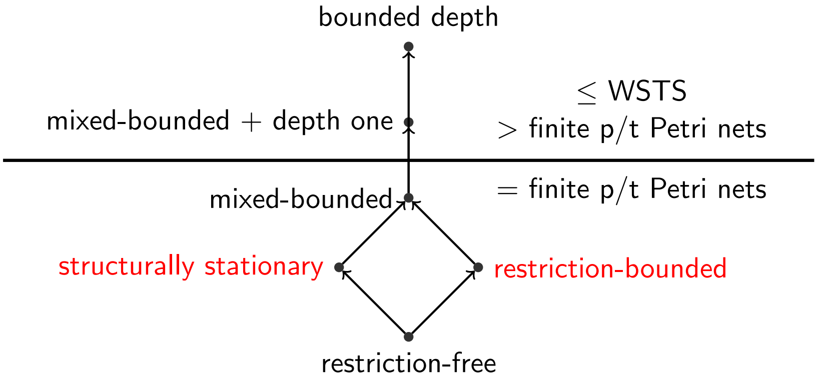
$\mathcal{N}_{\mathcal{C}}[\![P]\!]$ finite iff P generates finitely many restricted names (restriction boundedness)

Proof Idea $\Leftarrow$: Construct Process Places from Initial Process

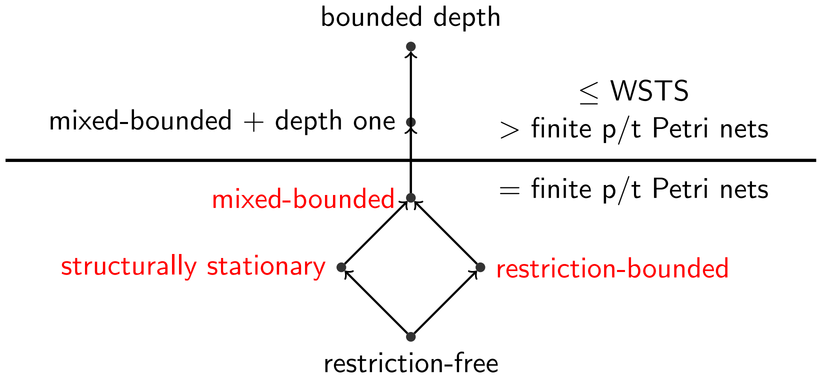
- Removing prefixes yields finite set of derivatives
- Reachable sequential processes = derivatives + substitutions

Finite by boundedness assumption

A Hierarchy of Process Classes

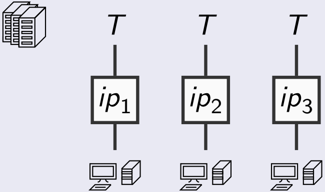


A Hierarchy of Process Classes



Back to Server

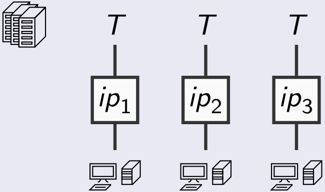
Can Translate



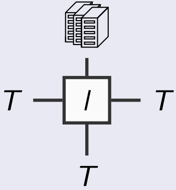
Structural Semantics

Back to Server

Can Translate



and

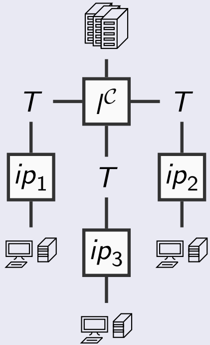


Structural Semantics

Concurrency Semantics

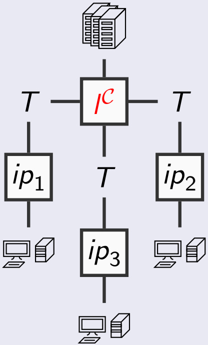
Back to Server

How about



Back to Server

How about

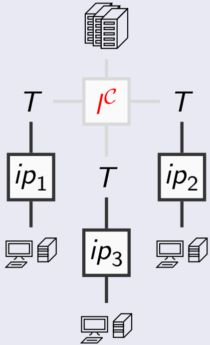


Idea

Tags determine translation of restricted names

Back to Server

How about



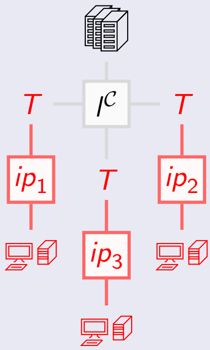
Idea

Tags determine translation of restricted names

- Translate $I^{\mathcal{C}}$ with concurrency semantics

Back to Server

How about



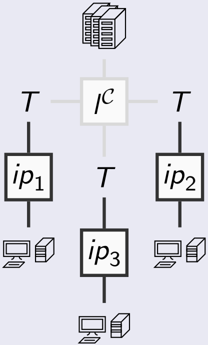
Idea

Tags determine translation of restricted names

- Translate $I^{\mathcal{C}}$ with concurrency semantics
- Translate ip with structural semantics

Back to Server

How about



Idea

Tags determine translation of restricted names

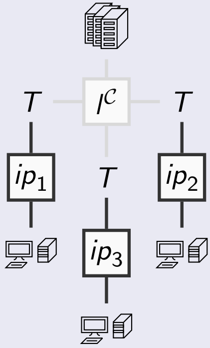
- Translate $I^{\mathcal{C}}$ with concurrency semantics
- Translate *ip* with structural semantics

Problems

- Fragments?

Back to Server

How about



Idea

Tags determine translation of restricted names

- Translate $I^{\mathcal{C}}$ with concurrency semantics
- Translate ip with structural semantics

Problems

- Fragments?
- Name-aware transition system?

Mixed Normal Form

Combine Standard and Restricted Form

$$\nu l^{\mathcal{C}}.(\underbrace{S\lfloor url, l^{\mathcal{C}}\rfloor}_{\text{Fragment}} \mid \underbrace{\nu ip.(T\lfloor l^{\mathcal{C}}, ip\rfloor \mid C\lfloor url, ip\rfloor)}_{\text{Fragment}})$$

Mixed Normal Form

Combine Standard and Restricted Form

$$\underbrace{\nu l^{\mathcal{C}}.(\ S\lfloor url, l^{\mathcal{C}}\rfloor \mid \nu ip.(T\lfloor l^{\mathcal{C}}, ip\rfloor \mid C\lfloor url, ip\rfloor)\ )}_{\text{Standard form over fragments}}$$

Mixed Normal Form

Recursive Function $mf(-)$

- Maximise scopes of tagged names
- Minimise scopes of untagged names

$$\nu ip.(\ \nu l^{\mathcal{C}}.(\ S\lfloor url, l^{\mathcal{C}}\rfloor \mid T\lfloor l^{\mathcal{C}}, ip\rfloor\ )\ \mid C\lfloor url, ip\rfloor)$$

Mixed Normal Form

Recursive Function $mf(-)$

- Maximise scopes of tagged names
- Minimise scopes of untagged names

$$\nu ip.(\nu l^{\mathcal{C}}.(S\lfloor url, l^{\mathcal{C}}\rfloor \mid T\lfloor l^{\mathcal{C}}, ip\rfloor) \mid C\lfloor url, ip\rfloor)$$

$$\equiv \nu ip.(\nu l^{\mathcal{C}}.(S\lfloor url, l^{\mathcal{C}}\rfloor \mid T\lfloor l^{\mathcal{C}}, ip\rfloor \mid C\lfloor url, ip\rfloor))$$

Mixed Normal Form

Recursive Function $mf(-)$

- Maximise scopes of tagged names
- Minimise scopes of untagged names

$$\nu ip.(\nu l^{\mathcal{C}}.(S\lfloor url, l^{\mathcal{C}}\rfloor \mid T\lfloor l^{\mathcal{C}}, ip\rfloor) \mid C\lfloor url, ip\rfloor)$$
$$\equiv \nu ip.(\ \nu l^{\mathcal{C}}.(S\lfloor url, l^{\mathcal{C}}\rfloor \mid T\lfloor l^{\mathcal{C}}, ip\rfloor \mid C\lfloor url, ip\rfloor)\ )$$

Mixed Normal Form

Recursive Function $mf(-)$

- Maximise scopes of tagged names
- Minimise scopes of untagged names

$$\nu ip.(\nu l^{\mathcal{C}}.(S\lfloor url, l^{\mathcal{C}}\rfloor \mid T\lfloor l^{\mathcal{C}}, ip\rfloor) \mid C\lfloor url, ip\rfloor)$$

$$\equiv \nu ip.(\ \nu l^{\mathcal{C}}.(S\lfloor url, l^{\mathcal{C}}\rfloor \mid T\lfloor l^{\mathcal{C}}, ip\rfloor \mid C\lfloor url, ip\rfloor)\ )$$

$$\equiv \nu l^{\mathcal{C}}.(S\lfloor url, l^{\mathcal{C}}\rfloor \mid \nu ip.(\ T\lfloor l^{\mathcal{C}}, ip\rfloor \mid C\lfloor url, ip\rfloor\ )\ )$$

Mixed Semantics

Name-Aware Transition System

Mixed normal form replaces standard form

$$(P^{\neq\nu}, \tilde{a}) \rightarrow^{na} (Q^{\neq\nu}, \tilde{a} \uplus \tilde{b})$$

Mixed Semantics

Name-Aware Transition System

Mixed normal form replaces standard form

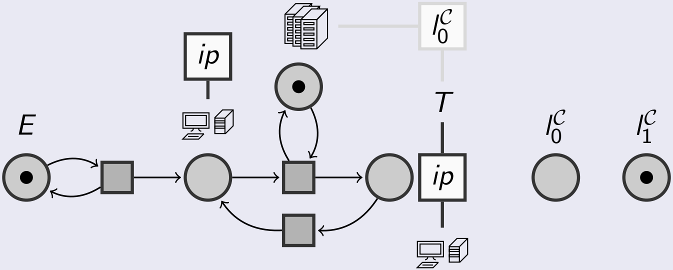
$$(P^{\neq\nu}, \tilde{a}) \rightarrow^{na} (Q^{\neq\nu}, \tilde{a} \uplus \tilde{b})$$

$$\rightsquigarrow$$

$$(P^{rf}, \tilde{a}^{\mathcal{C}}) \rightarrow^{na} (Q^{rf}, \tilde{a}^{\mathcal{C}} \uplus \tilde{b}^{\mathcal{C}})$$

Translation of Modified Server

Translation of Modified Server



Properties of the Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{N}_{\mathcal{M}}[\![P]\!]$

Theorem (Conservative Extension)

- *All names tagged*

$$\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}_{\mathcal{C}}[\![P]\!]$$

Properties of the Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{N}_{\mathcal{M}}[\![P]\!]$

Theorem (Conservative Extension)

- *All names tagged*

$$\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}_{\mathcal{C}}[\![P]\!]$$

Reason: mixed normal form = standard form

Properties of the Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{N}_{\mathcal{M}}[\![P]\!]$

Theorem (Conservative Extension)

- *All names tagged*

$$\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}_{\mathcal{C}}[\![P]\!]$$

  Reason: mixed normal form = standard form

- *No tagged names:*

$$\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}_{\mathcal{S}}[\![P]\!]$$

Properties of the Translation

Theorem (Bisimilarity)

$\mathcal{T}(P) \approx \mathcal{N}_{\mathcal{M}}[\![P]\!]$

Theorem (Conservative Extension)

- *All names tagged*

$$\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}_{\mathcal{C}}[\![P]\!]$$

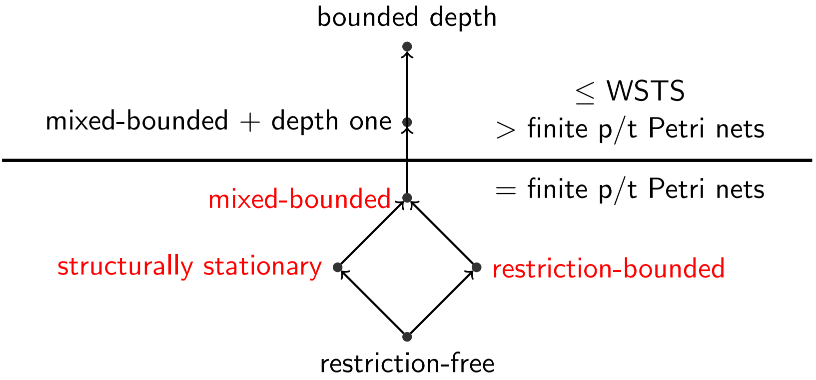
  Reason: mixed normal form = standard form

- *No tagged names:*

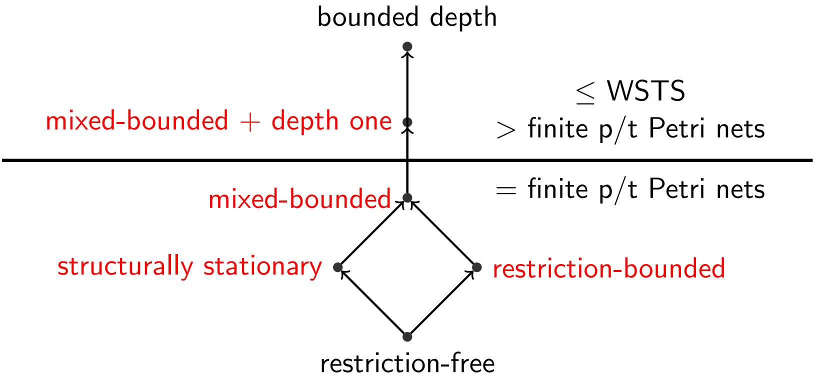
$$\mathcal{N}_{\mathcal{M}}[\![P]\!] = \mathcal{N}_{\mathcal{S}}[\![P]\!]$$

  Reason: no name places

A Hierarchy of Process Classes



A Hierarchy of Process Classes

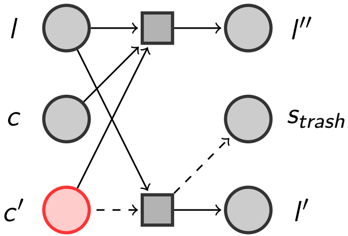


Undecidability of Reachability in Depth One

Test for Zero in Petri Nets with Transfer [DFS98]

$$l: \text{ if } c = 0 \text{ then goto } l'; \text{ else } c := c - 1; \text{ goto } l'';$$

- Create copy c' of counter c

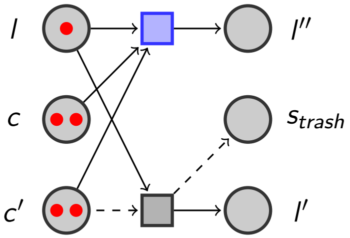


Undecidability of Reachability in Depth One

Test for Zero in Petri Nets with Transfer [DFS98]

$$l: \text{ if } c = 0 \text{ then goto } l'; \text{ else } c := c - 1; \text{ goto } l'';$$

- Create copy c' of counter c

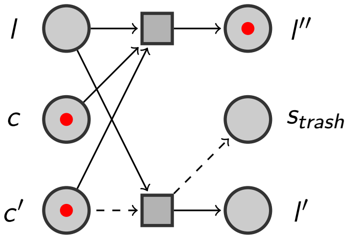


Undecidability of Reachability in Depth One

Test for Zero in Petri Nets with Transfer [DFS98]

$l:$ if $c = 0$ then goto l'; else $c := c - 1$; goto l'';

- Create copy c' of counter c

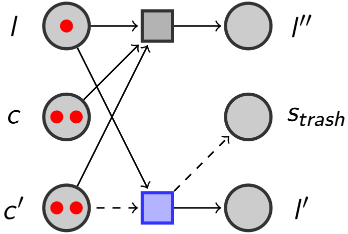


Undecidability of Reachability in Depth One

Test for Zero in Petri Nets with Transfer [DFS98]

$l:$ if $c = 0$ then goto l'; else $c := c - 1$; goto l'';

- Create copy c' of counter c
- Test for zero removes all tokens from c'

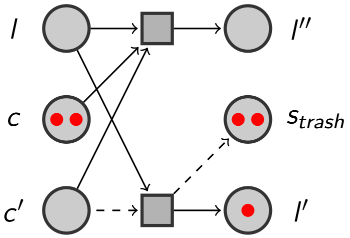


Undecidability of Reachability in Depth One

Test for Zero in Petri Nets with Transfer [DFS98]

$$l: \text{ if } c = 0 \text{ then goto } l'; \text{ else } c := c - 1; \text{ goto } l'';$$

- Create copy c' of counter c
- Test for zero removes all tokens from c'



Process Bunch

- Offers increment and decrement operations

$$PB(a, i, d, t) \quad := \quad i.(PB\lfloor a, i, d, t \rfloor \mid \overline{a})$$

Example

$$\nu a.(PB\lfloor a, i, d, t \rfloor \mid \overline{a} \mid \overline{a}) \rightarrow^{*} \nu a.(PB\lfloor a, i, d, t \rfloor \mid \overline{a} \mid \overline{a} \mid \overline{a})$$

Process Bunch

- Offers increment and decrement operations

$$PB(a,i,d,t) := i.(PB\lfloor a,i,d,t\rfloor \mid \overline{a}) + d.a.PB\lfloor a,i,d,t\rfloor$$

Example

$$\nu a.(PB\lfloor a,i,d,t\rfloor \mid \overline{a} \mid \overline{a}) \rightarrow^{*} \nu a.(PB\lfloor a,i,d,t\rfloor \mid \overline{a})$$

Process Bunch

- Offers increment and decrement operations
- Modifies arbitrarily many processes with one communication

$$\begin{aligned} PB(a,i,d,t) \quad := \quad & i.(PB\lfloor a,i,d,t\rfloor \mid \overline{a}) \\ & + d.a.PB\lfloor a,i,d,t\rfloor \\ & + t.\,\nu b.PB\lfloor b,i,d,t\rfloor \end{aligned}$$

Example

$$\overline{t} \mid \nu a.(t.\nu b.PB\lfloor b,i,d,t\rfloor + \ldots \mid \overline{a} \mid \overline{a})$$

Process Bunch

- Offers increment and decrement operations
- Modifies arbitrarily many processes with one communication

$$
\begin{aligned}
PB(a,i,d,t) \quad := \quad & i.(PB\lfloor a,i,d,t\rfloor \mid \overline{a}) \\
& + d.a.PB\lfloor a,i,d,t\rfloor \\
& + t.\,\nu b.PB\lfloor b,i,d,t\rfloor
\end{aligned}
$$

Example

$$
\begin{aligned}
& \overline{t} \mid \nu a.(t.\nu b.PB\lfloor b,i,d,t\rfloor + \ldots \mid \overline{a} \mid \overline{a}) \\
\rightarrow \quad & \nu b.PB\lfloor b,i,d,t\rfloor \mid \nu a.(\overline{a} \mid \overline{a})
\end{aligned}
$$

Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\overline{a}$ per $\nu a.(PB\lfloor a, i, d, t\rfloor \mid \overline{a})$

Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\overline{a}$ per $\nu a.(PB\lfloor a, i, d, t \rfloor \mid \overline{a})$

Not translatable by structural semantics

Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\overline{a}$ per $\nu a.(PB\lfloor a, i, d, t\rfloor \mid \overline{a})$
- Unbounded number of instances of νa

Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\overline{a}$ per $\nu a.(PB\lfloor a, i, d, t\rfloor \mid \overline{a})$
- Unbounded number of instances of νa

Not translatable by concurrency semantics

Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\overline{a}$ per $\nu a.(PB\lfloor a, i, d, t\rfloor \mid \overline{a})$
- Unbounded number of instances of νa
- Drop any of the restrictions yields mixed boundedness

Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\overline{a}$ per $\nu a.(PB\lfloor a, i, d, t \rfloor \mid \overline{a})$
- Unbounded number of instances of νa
- Drop any of the restrictions yields mixed boundedness

Intuitively

Server where threads gather clients

Related work

Processes as Graphs

Due to Milner [Mil79, MM79, MPW92, Mil99, SW01]

Automata-Theoretic Semantics

- Concurrency
  [Eng96, MP95a, Pis99, AM02, BG95, BG09, DKK08, KKN06]
- Structure [MP95b, Mey09]

Normal Forms

Decidability of structural congruence [EG99, EG04a, EG04b, EG07]

References I

R. Amadio and C. Meyssonnier.
On decidability of the control reachability problem in the asynchronous π-calculus.
Nord. J. Comp., 9(1):70–101, 2002.

N. Busi and R. Gorrieri.
A Petri net semantics for π-calculus.
In *Proc. of CONCUR*, volume 962 of *LNCS*, pages 145–159. Springer, 1995.

N. Busi and R. Gorrieri.
Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs.
J. Log. Alg. Prog., 78(1):138–162, 2009.

C. Dufourd, A. Finkel, and Ph. Schnoebelen.
Reset nets between decidability and undecidability.
In *Proc. of ICALP*, volume 1443 of *LNCS*, pages 103–115. Springer, 1998.

R. Devillers, H. Klaudel, and M. Koutny.
A compositional Petri net translation of general π-calculus terms.
For. Asp. Comp., 20(4–5):429–450, 2008.

J. Engelfriet and T. Gelsema.
Multisets and structural congruence of the pi-calculus with replication.
Theor. Comp. Sci., 211(1-2):311–337, 1999.

References II

J. Engelfriet and T. Gelsema.
The decidability of structural congruence for replication restricted pi-calculus processes.
Technical report, Leiden Institute of Advanced Computer Science, 2004.
Revised 2005.

J. Engelfriet and T. Gelsema.
A new natural structural congruence in the pi-calculus with replication.
Acta Inf., 40(6):385–430, 2004.

J. Engelfriet and T. Gelsema.
An exercise in structural congruence.
Inf. Process. Lett., 101(1):1–5, 2007.

J. Engelfriet.
A multiset semantics for the pi-calculus with replication.
Theor. Comp. Sci., 153(1-2):65–94, 1996.

V. Khomenko, M. Koutny, and A. Niaouris.
Applying Petri net unfoldings for verification of mobile systems.
In *Proc. of MOCA*, Bericht FBI-HH-B-267/06, pages 161–178. University of Hamburg, 2006.

R. Meyer.
A theory of structural stationarity in the π-calculus.
Acta Inf., 46(2):87–137, 2009.

References III

R. Milner.
Flowgraphs and flow algebras.
Journal of the Association for Computing Machinery, 26(4):794–818, 1979.

R. Milner.
Communicating and Mobile Systems: the π-Calculus.
CUP, 1999.

G. Milne and R. Milner.
Concurrent processes and their syntax.
Journal of the Association for Computing Machinery, 26(2):302–321, 1979.

U. Montanari and M. Pistore.
Checking bisimilarity for finitary π-calculus.
In *Proc. of the 6th International Conference on Concurrency Theory, CONCUR 1995*, volume 962 of *LNCS*, pages 42–56. Springer, 1995.

U. Montanari and M. Pistore.
Concurrent semantics for the π-calculus.
Electr. Notes Theor. Comp. Sci., 1:411–429, 1995.

R. Milner, J. Parrow, and D. Walker.
A calculus of mobile processes, part I.
Inf. Comp., 100(1):1–40, 1992.

References IV

M. Pistore.
History Dependent Automata.
PhD thesis, Dipartimento di Informatica, Università di Pisa, 1999.

D. Sangiorgi and D. Walker.
The π-calculus: a Theory of Mobile Processes.
CUP, 2001.