On the Relationship between $\pi$-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 $\pi$-calculus

Client sends on public channel \textit{url} his private address \textit{ip} to server

Graphically

In $\pi$-calculus
A Client-Server System in \( \pi \)-calculus

Client sends on public channel \( url \) his private address \( ip \) to server

\begin{itemize}
  \item Graphically
  \begin{itemize}
    \item \( ip \)
    \item \( \text{Client} \)
    \item \( \text{Server} \)
  \end{itemize}
  \item In \( \pi \)-calculus
  \begin{itemize}
    \item \( \nu ip. url (x). ip(x). C \mid url(y). (y|S \langle dat \rangle) \)
  \end{itemize}
\end{itemize}
A Client-Server System in $\pi$-calculus

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

Graphically

In $\pi$-calculus

$$\nu ip. \overline{url} \langle ip \rangle. ip(x). C[url, ip] \mid url(y). (y \langle dat \rangle \mid S[url, dat])$$
A Client-Server System in $\pi$-calculus

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

**Graphically**

- $ip$
- Client
- Server

**In $\pi$-calculus**

A Client-Server System in $\pi$-calculus

In response server spawns a new thread

Graphically

In $\pi$-calculus

$$\nu ip.\, \text{url}(ip).ip(x).C[url, ip] \mid url(y).(\bar{y}\langle dat \rangle \mid S[url, dat])$$
A Client-Server System in $\pi$-calculus

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

Graphically

In $\pi$-calculus

$$\nu ip . (ip(x).C[url, ip] \mid \overline{ip} \langle dat \rangle) \mid S[url, dat]$$
A Client-Server System in $\pi$-calculus

Thread terminates, client is ready to contact server again

Graphically

\[ \text{ip} \]

\[ \text{pc} \]

\[ \text{server} \]

In $\pi$-calculus

\[ \nu \text{ip}. C[url, \text{ip}] \mid S[url, \text{dat}] \]
A Client-Server System in $\pi$-calculus

Assumption

Environment $E$ generates clients

$E \rightarrow E \rightarrow E \rightarrow \ldots$
### 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:** $\mathcal{I}(P) \approx \mathcal{I}(\mathcal{N}[P])$
- **Finiteness:** $\mathcal{N}[P]$ finite iff ...
- **Expressiveness:** Unbounded concurrency and restrictions
A Hierarchy of Process Classes

- restriction-free
- structurally stationary
- mixed-bounded + depth one
- mixed-bounded

bounded depth

≤ WSTS

> finite p/t Petri nets

= finite p/t Petri nets

R. Meyer, R. Gorrieri (Universities Paris 7 and Bologna)
A Hierarchy of Process Classes

- Restriction-free
- Structurally stationary
- Mixed-bounded
- Mixed-bounded + depth one
- Bounded depth

\[ \leq \text{WSTS} \]

\[ > \text{finite p/t Petri nets} \]

\[ = \text{finite p/t Petri nets} \]
Idea of Structural Semantics

**Problem**

Unbounded number of clients and threads

![Diagram showing unbounded number of clients and threads](image)

**Observation**

Finite number of connection patterns

![Diagram showing finite number of connection patterns](image)
Idea of Structural Semantics

Represent Connections in a Petri Net
- Connection patterns yield places

Example

\[ \text{Connection patterns yield places} \]
Idea of Structural Semantics

Represent Connections in a Petri Net
- Connection patterns yield places
- Occurrence of a pattern yields a token

Example

- Connection patterns yield places
- Occurrence of a pattern yields a token
Idea of Structural Semantics

Transitions model evolution of patterns

In $\pi$-calculus

\[ E \rightarrow ip \quad E \rightarrow \text{ev} \quad E \rightarrow T \]
Idea of Structural Semantics

Transitions model evolution of patterns

In $\pi$-calculus

$E \rightarrow ip \quad E \rightarrow \text{computer} \quad T \rightarrow ip \quad ip \rightarrow E$

Structural Semantics

$E \quad ip \quad \text{computer} \quad T$
Idea of Structural Semantics

Transitions model evolution of patterns

In $\pi$-calculus

$E \rightarrow ip \quad E \rightarrow ip \rightarrow T \rightarrow ip \quad E$

Structural Semantics

$E \quad \rightarrow \quad ip \quad \rightarrow \quad T \quad \rightarrow \quad E$
Idea of Structural Semantics

Transitions model evolution of patterns

In $\pi$-calculus

- $E \rightarrow ip$ and $E \rightarrow \pi\rightarrow E$
- $E \rightarrow ip \rightarrow T \rightarrow ip \rightarrow E$

Structural Semantics

- $E \rightarrow \pi \rightarrow E$
- $E \rightarrow \pi \rightarrow T \rightarrow \pi \rightarrow E$
Restricted Form of Processes

Formalise idea of connection patterns

Example (Restricted Form)

Minimise scopes of restricted names

$$\nu ip.(ip(x).C[url, ip] \mid ip/dat \mid S[url, dat])$$

Lemma (Finiteness)

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

Formalise idea of connection patterns

Example (Restricted Form)

Minimise scopes of restricted names

\[ \nu \text{ip.}( \text{ip}(x).C[\text{url}, \text{ip}] \mid \text{ip}(\text{dat}) \mid S[\text{url}, \text{dat}] ) \]

\[ \equiv \nu \text{ip.}( \text{ip}(x).C[\text{url}, \text{ip}] \mid \text{ip}(\text{dat}) ) \mid S[\text{url}, \text{dat}] \]
Restricted Form of Processes

**Fragments**

Topmost parallel components are called **fragments**

\[
\nu ip.(ip(x).C[url, ip] \mid \overline{ip}\langle dat\rangle) \mid S[url, dat]
\]
Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

\[ \nu ip.(ip(x).C[\text{url, ip}] \mid \overline{ip}\langle\text{dat}\rangle) \mid S[\text{url, dat}] \]

Fragments correspond to connection patterns

\[ \text{Diagram of connection pattern} \]

\[ \text{Diagram of connection pattern} \]
Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

$$\nu ip.(ip(x).C[url, ip] \mid ip\langle dat\rangle) \mid S[url, dat]$$

Fragments correspond to connection patterns

Lemma (Finiteness)

$$\mathcal{N}_S[P] \text{ finite iff there is a finite set of fragments every reachable process consists of}$$
Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

\[ \nu ip.(ip(x).C[\text{url}, ip] \mid \overline{ip}\langle\text{dat}\rangle) \mid S[\text{url}, \text{dat}] \]

Fragments correspond to connection patterns

Lemma (Finiteness)

\( N_{S}[P] \) finite iff there is a finite set of fragments every reachable process consists of (structural stationarity)
A Hierarchy of Process Classes

- Restriction-free
- Structurally stationary
- Mixed-bounded
- Mixed-bounded + depth one
- Bounded depth

\[ \text{restriction-free} \leq \text{WSTS} \]
\[ \text{WSTS} > \text{finite p/t Petri nets} \]
\[ \text{finite p/t Petri nets} = \text{finite p/t Petri nets} \]

R. Meyer, R. Gorrieri (Universities Paris 7 and Bologna)
A Hierarchy of Process Classes

- restriction-free
- structurally stationary
- mixed-bounded
- mixed-bounded + depth one
- bounded depth

\[ \leq \text{WSTS} \]
\[ > \text{finite p/t Petri nets} \]
\[ = \text{finite p/t Petri nets} \]
A Modified Server

Server Maintains Control Channel with Threads

![Diagram showing the server maintaining control channel with threads.]
A Modified Server

Server Maintains Control Channel with Threads

Structural Semantics Infinite

R. Meyer, R. Gorrieri (Universities Paris 7 and Bologna)
Idea of Concurrency Semantics

- Treat restricted names as free names

Example

```
  I
  \_______________________\
  |                     |
  |                     |
  |                     |
  |                     |
  \_______________________/  T

T

T

T
```
Idea of Concurrency Semantics

- Treat restricted names as free names

Example

```
T  I  T
```

Crucial Technicality

Preserve order in which free names are generated
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

![Concurrent Processes Diagram]
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

\[ \nu z_0. P \]
Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices $\nu z_0. P$
- Process $P$ is in standard form $sf(P)$

$$\bar{a} \mid \nu z_0. ( \overline{z_0} \mid a.z_0.K[a] )$$
Name-Aware Processes

Assumption and Preliminaries

- Restricted names have indices $\nu z_0. P$
- Process $P$ is in standard form $sf(P)$

$$\bar{a} \mid \nu z_0. (\overline{z_0} \mid a.z_0.K[a])$$

$$\equiv \nu z_0. (\overline{a} \mid \overline{z_0} \mid a.z_0.K[a])$$
Name-Aware Processes

**Assumption and Preliminaries**

- Restricted names have indices $\nu z_0.P$
- 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 $\nu z_0. P$
- 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 $\nu z_0.P$
- 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

$\nu z_0. (\bar{a} | \bar{z}_0 | a.z_0.K[a]) \rightsquigarrow (\bar{a} | \bar{z}_0 | a.z_0.K[a], \{z_0\})$
Name-Aware Reaction Relation

Example

Idea: Generate names by incrementing indices
Name-Aware Reaction Relation

Example

Idea: Generate names by **incrementing indices**

\[ K[a] \rightarrow \nu z.(\bar{z} \mid a.z.K[a]) \]
Name-Aware Reaction Relation

Example

Idea: Generate names by incrementing indices

\[ K[a] \rightarrow \nu z.(\bar{z} \mid a.z.K[a]) \]

\[ \rightsquigarrow \]

\[ (K[a], \{z_0\}) \rightarrow^{na} (\bar{z}_1 \mid a.z_1.K[a], \{z_0, z_1\}) \]
Name-Aware Reaction Relation

Example

Idea: Generate names by incrementing indices

\[ K[a] \rightarrow \nu z. (\overline{z} \mid a.z.K[a]) \]

\[ \rightsquigarrow \]

\[ (K[a], \{z_0\}) \rightarrow^{na} (\overline{z_1} \mid a.z_1.K[a], \{z_0, z_1\}) \]

Definition (Name-Aware Reaction Relation)

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

Indices in \( \tilde{b} \) incremented
Bisimilarity

Lemma (Bisimilarity)

\[ T(P) \cong T_{na}(P \neq \nu, \tilde{a}) \text{ with } sf(P) = \nu \tilde{a}.P \neq \nu \]

Example \[ \tilde{a} \mid K[a] \text{ with } K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K[a]) \]

\[ \tilde{a} \mid K[a] \bullet \bullet \bullet (\tilde{a} \mid K[a], \emptyset) \]
Bisimilarity

Lemma (Bisimilarity)

\[ T(P) \approx T_{na}(P \neq \nu, \bar{a}) \text{ with } sf(P) = \nu \bar{a}.P \neq \nu \]

Example \( \bar{a} | K[a] \) with \( K(a) = \nu z_0.(\bar{z}_0 | a.z_0.K[a]) \)

\[ \nu z_0.(\bar{a} | \bar{z}_0 | a.z_0.K[a]) \]

\[ (\bar{a} | K[a], \emptyset) \]

\[ (\bar{a} | \bar{z}_0 | a.z_0.K[a], \{z_0\}) \]
Lemma (Bisimilarity)

\[ \mathcal{T}(P) \approx \mathcal{T}_{na}(P \neq \nu, \tilde{a}) \text{ with } sf(P) = \nu \tilde{a}.P \neq \nu \]

Example \( \bar{a} \mid K \bar{a} \) with \( K(a) = \nu z_0.(\bar{z}_0 \mid a.z_0.K \bar{a}) \)

\begin{align*}
\nu z_0.(\bar{a} \mid \bar{z}_0 \mid a.z_0.K \bar{a}) & \xrightarrow{na} \mathcal{T}_{na}(P \neq \nu, \tilde{a}) \xrightarrow{na} \mathcal{T}(P) \\
\nu z_0.(\bar{z}_0 \mid z_0.K \bar{a}) & \xrightarrow{na} \mathcal{T}_{na}(P \neq \nu, \tilde{a}) \xrightarrow{na} \mathcal{T}(P)
\end{align*}
Bisimilarity

**Lemma (Bisimilarity)**

\[ T(P) \approx T_{na}(P \neq \nu, \tilde{a}) \text{ with } sf(P) = \nu \tilde{a}.P \neq \nu \]

**Example** \( \overline{a} \mid K[a] \) with \( K(a) = \nu z_0. (\overline{z_0} \mid a.z_0.K[a]) \)

- \( \nu z_0. (\overline{a} \mid \overline{z_0} \mid a.z_0.K[a]) \)
  - \( \nu z_0. (\overline{z_0} \mid z_0.K[a]) \)
  - \( K[a] \)
  - \( (\overline{a} \mid K[a], \emptyset) \)
- \( (\overline{a} \mid \overline{z_0} \mid a.z_0.K[a], \{z_0\}) \)
- \( (\overline{z_0} \mid z_0.K[a], \{z_0\}) \)
- \( (K[a], \{z_0\}) \)
Bisimilarity

Lemma (Bisimilarity)

\[ \mathcal{T}(P) \approx \mathcal{T}_{na}(P \not\equiv \nu, \bar{a}) \text{ with } sf(P) = \nu \bar{a}.P \not\equiv \nu \]

Example \( \bar{a} \mid K \downharpoonright a \) with \( K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K \downharpoonright a) \)

\[ \begin{align*}
\nu z_0.(\bar{a} \mid \overline{z_0} \mid a.z_0.K \downharpoonright a) & \quad \vdash \quad (\bar{a} \mid K \downharpoonright a, \emptyset) \\
\nu z_0.(\overline{z_0} \mid z_0.K \downharpoonright a) & \quad \vdash \quad (\overline{z_0} \mid z_0.K \downharpoonright a, \{z_0\}) \\
K \downharpoonright a & \quad \vdash \quad (K \downharpoonright a, \{z_0\}) \\
\nu z_0.(\overline{z_0} \mid a.z_0.K \downharpoonright a) & \quad \vdash \quad (\overline{z_1} \mid a.z_1.K \downharpoonright a, \{z_0, z_1\})
\end{align*} \]
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

\[ z_0, z_1, z_2 \]
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

\[
\begin{align*}
&z_0 \\
&z_1 \\
&z_2
\end{align*}
\]

\[K[a] \quad a.z_0.K[a]\]
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 $\overline{a} \mid K \overline{a}$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K \overline{a})$
Example $\bar{a} \mid K[a]$ with $K(a) = \nu z_0. (\bar{z}_0 \mid a.z_0.K[a])$
Example $\overline{a} \mid K[a]$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K[a])$
Example $\overline{a} \mid K[a]$ with $K(a) = \nu z_0.(\overline{z_0} \mid a.z_0.K[a])$

Name-Aware TS

\[
\begin{align*}
&\text{na}((\overline{a} \mid K[a], \emptyset)) \\
&\text{na}((\overline{a} \mid z_0 \mid a.z_0.K[a], \{z_0\})) \\
&\text{na}((z_0 \mid z_0.K[a], \{z_0\})) \\
&\text{na}(K[a], \{z_0\})
\end{align*}
\]

Concurrency Semantics

\[
\begin{align*}
&z_0 \rightarrow z_1 \\
&z_0 \rightarrow a.z_0.K[a] \\
&z_2 \rightarrow a.z_1.K[a] \\
&z_1 \rightarrow z_0.K[a] \\
&z_1 \rightarrow \overline{a}
\end{align*}
\]
Example $\overline{a} \mid K[a]$ with $K(a) = \nu z_0.(\overline{z}_0 \mid a.z_0.K[a])$

**Name-Aware TS**

1. $(\overline{a} \mid K[a], \emptyset)$
2. $(\overline{a} \mid \overline{z}_0 \mid a.z_0.K[a], \{z_0\})$
3. $(\overline{z}_0 \mid z_0.K[a], \{z_0\})$
4. $(K[a], \{z_0\})$
5. $(\overline{z}_1 \mid a.z_1.K[a], \{z_0,z_1\})$

**Concurrency Semantics**

- $z_0$
- $z_1$
- $z_2$

$\overline{z}_0 \rightsquigarrow \overline{z}_1 \rightsquigarrow \overline{z}_2$

$_{\nu z_0.(\overline{z}_0 \mid a.z_0.K[a])}$

\[z_0 \rightsquigarrow z_1 \rightleftarrows z_2 \rightsquigarrow z_3 \rightsquigarrow z_4\]

\[\overline{a} \rightsquigarrow \overline{a}\]
Properties of Translation

Theorem (Bisimilarity)

\[ \mathcal{I}(P) \approx \mathcal{I}(\mathcal{N}_c[P]) \]
Properties of Translation

Theorem (Bisimilarity)

\[ T(P) \approx T(N_c[P]) \]

Theorem (Finiteness)

\[ N_c[P] \text{ finite iff } P \text{ generates finitely many restricted names} \]
Properties of Translation

Theorem (Bisimilarity)
\[ \mathcal{T}(P) \approx \mathcal{T}(\mathcal{N}_c[P]) \]

Theorem (Finiteness)
\[ \mathcal{N}_c[P] \text{ finite iff } P \text{ generates finitely many restricted names} \]
\[ \text{(restriction boundedness)} \]
Properties of Translation

**Theorem (Bisimilarity)**

\[ T(P) \approx T(\mathcal{N}_c[P]) \]

**Theorem (Finiteness)**

\[ \mathcal{N}_c[P] \text{ finite iff } P \text{ generates finitely many restricted names} \]  
(rediction boundedness)

**Proof Idea \( \Leftarrow \): Construct Process Places from Initial Process**

- Removing prefixes yields finite set of derivatives
Properties of Translation

Theorem (Bisimilarity)
\[ T(P) \approx T(\mathcal{N}_c[P]) \]

Theorem (Finiteness)
\[ \mathcal{N}_c[P] \text{ finite iff } P \text{ generates finitely many restricted names (restriction boundedness)} \]

Proof Idea \( \iff \): 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

- Structurally stationary
- Restriction-free
- Restriction-bounded
- Mixed-bounded
- Mixed-bounded + depth one
- Bounded depth

\[ \leq \text{WSTS} \]
\[ > \text{finite p/t Petri nets} \]
\[ = \text{finite p/t Petri nets} \]
A Hierarchy of Process Classes

- bounded depth
- mixed-bounded + depth one
- mixed-bounded
- structurally stationary
- restriction-bounded
- restriction-free

\[ \text{mixed-bounded} \leq \text{WSTS} \leq \text{finite p/t Petri nets} \]

R. Meyer, R. Gorrieri (Universities Paris 7 and Bologna)
Can Translate

\[ T \rightarrow \text{Structural Semantics} \]

\[ T \rightarrow \text{Mixed Semantics} \]

\[ T \rightarrow \text{Borderline to Finite P/T Nets} \]
Can Translate

Structural Semantics

Concurrency Semantics

Properties

Back to Server
How about

T 
\( \pi \)

T 
\( \pi \)

T 
\( \pi \)

ip_1 

ip_2 

ip_3
How about

Idea

Tags determine translation of restricted names

\[ T \xrightarrow{\ell C} T \]

\[ \text{ip}_1 \quad T \quad \text{ip}_2 \]

\[ \text{ip}_3 \]

\[ \text{ip}_3 \]

\[ \text{ip}_3 \]
How about

T

\( \pi \)

\( \pi \)

T

ip_1

T

ip_2

T

ip_3

Idea

Tags determine translation of restricted names

- Translate \( \pi^C \) with concurrency semantics
How about

<table>
<thead>
<tr>
<th>$ip_1$</th>
<th>$T$</th>
<th>$ip_2$</th>
<th>$T$</th>
<th>$ip_3$</th>
</tr>
</thead>
</table>

Idea

Tags determine translation of restricted names
- Translate $I^C$ with concurrency semantics
- Translate $ip$ with structural semantics
How about

Idea

Tags determine translation of restricted names
- Translate $I^C$ with concurrency semantics
- Translate $ip$ with structural semantics

Problems
- Fragments?
How about

Idea

Tags determine translation of restricted names

- Translate $I^C$ with concurrency semantics
- Translate $ip$ with structural semantics

Problems

- Fragments?
- Name-aware transition system?
Mixed Normal Form

Combine Standard and Restricted Form

\[ \nu l^C \cdot (S[\text{url, } l^C] \mid \nu ip.(T[l^C, ip] \mid C[\text{url, ip}])) \]

Fragment  |  Fragment

\[ T \]  
\[ ip \]  
\[ \text{Network} \]
Mixed Normal Form

Combine Standard and Restricted Form

$$\nu l^C (S[lurl, l^C] | \nu ip(T[l^C, ip] | C[url, ip]))$$

Standard form over fragments
Recursive Function $mf(\_)$

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

$$\nu ip. (\nu l^C. (S|url, l^C|T|l^C, ip)|C|url, ip)$$
Mixed Normal Form

Recursive Function $mf(\ )$

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

\[
\nu ip. ( \nu l^C . ( S \lfloor url, l^C \rfloor \mid T \lfloor l^C, ip \rfloor ) \mid C \lfloor url, ip \rfloor ) =
\equiv \nu ip. ( \nu l^C . ( S \lfloor url, l^C \rfloor \mid T \lfloor l^C, ip \rfloor \mid C \lfloor url, ip \rfloor ) )
\]
Mixed Normal Form

Recursive Function $mf(-)$

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

\[
\nu ip. (\nu l^C. (S[lurl, l^C] \mid T[l^C, ip] \mid C[lurl, ip]) \mid C[lurl, ip]) \equiv \\
u ip. (\nu l^C. (S[lurl, l^C] \mid T[l^C, ip] \mid C[lurl, ip]))
\]
Mixed Normal Form

Recursive Function $mf(\_)$

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

\[
\]
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

\[
(\nu \neq P, \tilde{a}) \xrightarrow{na} (\nu \neq Q, \tilde{a} \uplus \tilde{b})
\]

\[
\rightsquigarrow
\]

\[
(\nu \neq P^\text{rf}, \tilde{a}^C) \xrightarrow{na} (\nu \neq Q^\text{rf}, \tilde{a}^C \uplus \tilde{b}^C)
\]
Translation of Modified Server
Properties of the Translation

Theorem (Bisimilarity)

\[ \mathcal{T}(P) \approx \mathcal{N}_M[P] \]

Theorem (Conservative Extension)

- *All names tagged*

\[ \mathcal{N}_M[P] = \mathcal{N}_C[P] \]
Properties of the Translation

Theorem (Bisimilarity)
\[ \mathcal{T} (P) \approx \mathcal{N}_M[P] \]

Theorem (Conservative Extension)
- All names tagged

\[ \mathcal{N}_M[P] = \mathcal{N}_C[P] \]

Reason: mixed normal form = standard form
Properties of the Translation

Theorem (Bisimilarity)

\[ I(P) \approx N_M[P] \]

Theorem (Conservative Extension)

- All names tagged

\[ N_M[P] = N_c[P] \]

Reason: mixed normal form = standard form

- No tagged names:

\[ N_M[P] = N_S[P] \]
Properties of the Translation

Theorem (Bisimilarity)

\[ \mathcal{I}(P) \approx \mathcal{N}_M[P] \]

Theorem (Conservative Extension)

- **All names tagged**

  \[ \mathcal{N}_M[P] = \mathcal{N}_C[P] \]

  *Reason: mixed normal form = standard form*

- **No tagged names:**

  \[ \mathcal{N}_M[P] = \mathcal{N}_S[P] \]

  *Reason: no name places*
A Hierarchy of Process Classes

- **bounded depth**
- **mixed-bounded + depth one**
- **mixed-bounded**
- **structurally stationary**
- **restriction-bounded**
- **restriction-free**

\[ \leq \text{WSTS} \]
\[ > \text{finite p/t Petri nets} \]
\[ = \text{finite p/t Petri nets} \]
A Hierarchy of Process Classes

restriction-free

structurally stationary

mixed-bounded

mixed-bounded + depth one

bounded depth

≤ WSTS

> finite p/t Petri nets

= finite p/t Petri nets

restriction-bounded

finite p/t Petri nets

R. Meyer, R. Gorrieri (Universities Paris 7 and Bologna)
Undecidability of Reachability in Depth One

Test for Zero in Petri Nets with Transfer [DFS98]

\[
\begin{align*}
    l : & \text{ if } c = 0 \text{ then goto } l' \text{; else } c := c - 1 \text{; goto } l''; \\
    \textbf{Create copy } c' \text{ of counter } c
\end{align*}
\]

![Petri Net Diagram]

R. Meyer, R. Gorrieri (Universities Paris 7 and Bologna)
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]

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

* 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 \( c \)
- Test for zero removes all tokens from \( c' \)

\[ \begin{align*}
  l & \quad \text{if } c = 0 \quad \text{then goto } l' \quad \text{else } c := c - 1 \quad \text{goto } l'' \\
  c & \quad \text{create copy } c' \quad \text{of counter } c \\
  c' & \quad \text{test for zero removes all tokens from } c' \\
  S_{\text{trash}} & \\
  l' & \\
  l'' &
\end{align*} \]
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 \( c' \)

\[ l : \text{if } c = 0 \text{ then goto } l' ; \text{ else } c := c - 1 ; \text{ goto } l'' ; \]
Process Bunch

- Offers **increment** and decrement operations

\[
PB(a, i, d, t) := i.(PB[a, i, d, t] \mid \overline{a})
\]

Example

\[
\nu a.\left(PB\left[a, i, d, t\right] \mid \overline{a} \mid \overline{a}\right) \rightarrow^* \nu a.\left(PB\left[a, i, d, t\right] \mid \overline{a} \mid \overline{a} \mid \overline{a}\right)
\]
Process Bunch

- Offers increment and **decrement** operations

\[
PB(a, i, d, t) := i.(PB[a, i, d, t] | \overline{a})
\]
\[
+ d.a.PB[a, i, d, t]
\]

Example

\[
\nu a.(PB[a, i, d, t] | \overline{a} | \overline{a}) \rightarrow^* \nu a.(PB[a, i, d, t] | \overline{a})
\]
Process Bunch

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

\[ PB(a, i, d, t) := i.\left( PB[a, i, d, t] \mid \overline{a} \right) + d.a.PB[a, i, d, t] + t.\nu b.PB[b, i, d, t] \]

Example

\[ t \mid \nu a.( t.\nu b.PB[b, i, d, t] + \ldots ) \mid \overline{a} \mid \overline{a} \]
Process Bunch

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

\[
P B(a, i, d, t) := i.(P B[a, i, d, t] \mid \bar{a}) + d.a.P B[a, i, d, t] + t.\nu b. P B[b, i, d, t]
\]

Example

\[
\bar{t} \mid \nu a.(t.\nu b. P B[b, i, d, t] + \ldots \mid \bar{a} \mid \bar{a})
\]

\[
\rightarrow \nu b. P B[b, i, d, t] \mid \nu a.(\bar{a} \mid \bar{a})
\]
Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\bar{a}$ per $\nu a. (PB[a, i, d, t] \mid \bar{a})$
Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\bar{a}$ per $\nu a. (PB[a, i, d, t] | \bar{a})$
  
  Not translatable by structural semantics
Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\bar{a}$ per $\nu a. (PB[a, i, d, t] \mid \bar{a})$
- Unbounded number of instances of $\nu a$
Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\bar{a}$ per $\nu a. (PB [a, i, d, t] | \bar{a})$
- Unbounded number of instances of $\nu a$
  
  Not translatable by concurrency semantics
Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\bar{a}$ per $\nu a. (PB[a, i, d, t] \mid \bar{a})$
- Unbounded number of instances of $\nu a$
- Drop any of the restrictions yields mixed boundedness
Bound is Tight

Undecidability Relies on Combination of Two Features

- Unbounded number of processes $\bar{a}$ per $\nu a. (PB[ a, i, d, t ] | \bar{a})$
- Unbounded number of instances of $\nu a$
- 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 $\pi$-calculus.

N. Busi and R. Gorrieri.
A Petri net semantics for $\pi$-calculus.

N. Busi and R. Gorrieri.
Distributed semantics for the $\pi$-calculus based on Petri nets with inhibitor arcs.

C. Dufourd, A. Finkel, and Ph. Schnoebelen.
Reset nets between decidability and undecidability.

R. Devillers, H. Klaudel, and M. Koutny.
A compositional Petri net translation of general $\pi$-calculus terms.

J. Engelfriet and T. Gelsema.
Multisets and structural congruence of the pi-calculus with replication.
References II

J. Engelfriet and T. Gelsema.
The decidability of structural congruence for replication restricted pi-calculus processes.
Revised 2005.

J. Engelfriet and T. Gelsema.
A new natural structural congruence in the pi-calculus with replication.

J. Engelfriet and T. Gelsema.
An exercise in structural congruence.

J. Engelfriet.
A multiset semantics for the pi-calculus with replication.

V. Khomenko, M. Koutny, and A. Niaouris.
Applying Petri net unfoldings for verification of mobile systems.

R. Meyer.
A theory of structural stationarity in the $\pi$-calculus.
References III

R. Milner.
Flowgraphs and flow algebras.

R. Milner.
*Communicating and Mobile Systems: the \( \pi \)-Calculus.*
CUP, 1999.

G. Milne and R. Milner.
Concurrent processes and their syntax.

U. Montanari and M. Pistore.
Checking bisimilarity for finitary \( \pi \)-calculus.

U. Montanari and M. Pistore.
Concurrent semantics for the \( \pi \)-calculus.

R. Milner, J. Parrow, and D. Walker.
A calculus of mobile processes, part I.
M. Pistore.

*History Dependent Automata.*

D. Sangiorgi and D. Walker.

*The π-calculus: a Theory of Mobile Processes.*