Introduction to $\pi$-Calculus
Structural Semantics
Structural Stationarity
Decidability in Bounded Depth

Structural Stationarity in the $\pi$-Calculus

Roland Meyer
Department of Computing Science
University of Oldenburg

Disputation
2009-02-20
A Client-Server System in the \( \pi \)-Calculus

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

\textbf{Graphically}

\begin{itemize}
  \item \textit{ip}
  \item Server
  \item Client
\end{itemize}

\textbf{In \( \pi \)-Calculus}

\[
\nu \text{ip}. \langle \text{ip} \rangle. \text{ip}(x). C \lfloor \text{url}, \text{ip} \rfloor | \text{url}(y). (y \langle \text{dat} \rangle | S \lfloor \text{url}, \text{dat} \rfloor)
\]
A Client-Server System in the $\pi$-Calculus

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

Graphically

- \textit{ip}
- Server
- Client

In $\pi$-Calculus

\[
\nu \text{ip}. \langle \text{ip} \rangle. \text{ip}(x). C \lfloor \text{url}, \text{ip} \rfloor | \text{url}(y). (y\langle \text{dat} \rangle | S \lfloor \text{url}, \text{dat} \rfloor )
\]
A Client-Server System in the $\pi$-Calculus

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

Graphically

\begin{align*}
\text{Client} & \quad \text{Server} \\
\text{ip} & \quad \text{url} \langle \text{ip} \rangle \cdot \text{ip}(x).C[\text{url}, \text{ip}] | \\
& \quad \text{url}(y).NY[\text{dat}] | S[\text{url}, \text{dat}] \\
\end{align*}
A Client-Server System in the $\pi$-Calculus

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

**Graphically**

Client

Server

$ip$

**In $\pi$-Calculus**

$$\nu ip.\overline{url} \langle ip \rangle . ip(x). C[ url, ip ] | url(y). (\overline{y} \langle dat \rangle | S[ url, dat ] )$$
In response server spawns a new thread

**Graphically**

```
ip
```

**In π-Calculus**

```
\nu ip. url\langle ip\rangle . ip(x) . C [url, ip] | url(y) . (y\langle dat\rangle | S [url, dat])
```

Roland Meyer (University of Oldenburg)
A Client-Server System in the π-Calculus

Thread sends on the private channel \textit{ip} data \textit{dat} to the client

Graphically

In π-Calculus

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

Thread terminates, client is ready to contact server again

Graphically

In $\pi$-Calculus

$$\nu ip. C[url, ip] \mid S[url, dat]$$
A Client-Server System in the $\pi$-Calculus

**Assumption**

Environment $E$ generates clients

$$
E \rightarrow E \rightarrow \ldots
$$
A Semantical Approach to Verification

**Goal**

**Automatic** verification of dynamically reconfigurable systems
A Semantical Approach to Verification

Goal

Automatic verification of dynamically reconfigurable systems

- Occurrence number properties: Is there exactly one server?
A Semantical Approach to Verification

<table>
<thead>
<tr>
<th>Goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic verification of dynamically reconfigurable systems</td>
</tr>
<tr>
<td>- Occurrence number properties: Is there exactly one server?</td>
</tr>
<tr>
<td>- Temporal properties: Does a request create a new thread?</td>
</tr>
</tbody>
</table>
A Semantical Approach to Verification

Goal

Automatic verification of dynamically reconfigurable systems

- Occurrence number properties: Is there exactly one server?
- Temporal properties: Does a request create a new thread?
- Topological properties: Is a thread always connected to a client?
A Semantical Approach to Verification

**Goal**

**Automatic** verification of dynamically reconfigurable systems

- Occurrence number properties: Is there exactly one server?
- Temporal properties: Does a request create a new thread?
- Topological properties: Is a thread always connected to a client?

**Problem**

**Finite representation** of infinite state space required
A Semantical Approach to Verification

Goal
Automatic verification of dynamically reconfigurable systems
- Occurrence number properties: Is there exactly one server?
- Temporal properties: Does a request create a new thread?
- Topological properties: Is a thread always connected to a client?

Problem
Finite representation of infinite state space required

Approach
Translate \( \pi \)-Calculus into place/transition Petri nets
Overview

1. Introduction to π-Calculus
2. Structural Semantics
3. Structural Stationarity
4. Decidability in Bounded Depth
Idea of the Structural Semantics

Problem

Unbounded number of clients and threads

\[
\text{ip}_1 \rightleftharpoons \text{ip}_2 \rightleftharpoons E \rightleftharpoons T \rightleftharpoons \text{ip}_4
\]

\[
\text{ip}_3 \rightleftharpoons \text{ip}_4 \rightleftharpoons T \rightleftharpoons \text{ip}_5
\]

Observation

Finite number of connection patterns

\[
\text{ip} \rightleftharpoons E \rightleftharpoons T \rightleftharpoons \text{ip}
\]
Idea of the Structural Semantics

Represent Connections in a Petri net

- Every connection pattern yields a place

Example:

```
ip_1  ➔  E  ➔  T  ➔  ip_4

ipa  ➔  E  ➔  ipb
```

Roland Meyer (University of Oldenburg) Structural Stationarity in the π-Calculus 5 / 20
Idea of the Structural Semantics

Represent Connections in a Petri net
- Every connection pattern yields a place
- Every occurrence of the pattern yields a token

Example

```
ip_1 ------ E ------ T ------ ip_4
           \     |     /            \     |     /
ip_2 ------ T ------ ip_5
           \     |     /            \     |     /
           \     |     /            \     |     /
ip_3 ------ T

E    T
```

Roland Meyer (University of Oldenburg)
Idea of the Structural Semantics

Transitions model the evolution of patterns

In $\pi$-Calculus

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

The Structural Semantics

$$E$$  $$ip$$  $$T$$
Idea of the Structural Semantics

Transitions model the evolution of patterns

In $\pi$-Calculus

$$E \xrightarrow{ip} E \xrightarrow{ip} T \xrightarrow{ip} E$$

The Structural Semantics
Idea of the Structural Semantics

Transitions model the evolution of patterns

In $\pi$-Calculus

\[
\begin{align*}
E & \rightarrow ip & E & \rightarrow \text{transitions} & T & \rightarrow \text{transitions} & E \\
\text{patterns} & & \text{patterns} & & \text{patterns} & & \text{patterns}
\end{align*}
\]

The Structural Semantics

Roland Meyer (University of Oldenburg)
Idea of the Structural Semantics

Transitions model the evolution of patterns

In $\pi$-Calculus

$$E \xrightarrow{ip} E \xrightarrow{T} ip E$$

The Structural Semantics

$$E \xrightarrow{ip}$$
Restricted Form of Processes

Purpose

- Formalise the idea of connection patterns
- Define depth and breadth
Restricted Form of Processes

Purpose

- Formalise the idea of connection patterns
- Define depth and breadth

Idea

Minimise the scopes of restricted names
Restricted Form of Processes

Purpose
- Formalise the idea of connection patterns
- Define depth and breadth

Idea
Minimise the scopes of restricted names

Example (Restricted Form)

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

**Purpose**

- Formalise the idea of connection patterns
- Define depth and breadth

**Idea**

Minimise the scopes of restricted names

**Example (Restricted Form)**

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

Roland Meyer (University of Oldenburg)  
Structural Stationarity in the \(\pi\)-Calculus  
6 / 20
Fragments

Topmost parallel components are called fragments

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

Fragments

Topmost parallel components are called fragments

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

Fragments correspond to connection patterns
Properties of the Semantics

<table>
<thead>
<tr>
<th>Theorem (Full Retrievability)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transition systems of $P$ and $N[P]$ are <strong>isomorphic</strong>. Reachable processes can be computed from markings.</td>
</tr>
</tbody>
</table>
Properties of the Semantics

Theorem (Full Retrievability)

Transition systems of $P$ and $\mathcal{N}[P]$ are isomorphic. Reachable processes can be computed from markings.

Theorem (Full Abstraction)

Equality of the semantics coincides with structural congruence:

\[ P \equiv Q \iff \mathcal{N}[P] = \mathcal{N}[Q] \]
Properties of the Semantics

**Theorem (Full Retrievability)**

Transition systems of $P$ and $\mathcal{N}[P]$ are isomorphic. Reachable processes can be computed from markings.

**Theorem (Full Abstraction)**

Equality of the semantics coincides with structural congruence:

$$P \equiv Q \iff \mathcal{N}[P] = \mathcal{N}[Q]$$

**Lemma**

The structural semantics of a closed process is communication-free, i.e., every transition has a single place in its preset.
Structural Stationarity and Finiteness

Finiteness

- Structural semantics may be an infinite Petri net
- Automatic verification methods require finite nets
Structural Stationarity and Finiteness

Finiteness

- Structural semantics may be an infinite Petri net
- Automatic verification methods require finite nets

Definition (Structural Stationarity)

A process is **structurally stationary** iff there are **finitely many fragments** every reachable process consists of.
Structural Stationarity and Finiteness

**Finiteness**
- Structural semantics may be an infinite Petri net
- Automatic verification methods require finite nets

**Definition (Structural Stationarity)**
A process is structurally stationary iff there are finitely many fragments every reachable process consists of.

**Lemma (Finiteness)**

\[ \mathcal{N}[P] \text{ is finite if and only if process } P \text{ is structurally stationary.} \]
A First Characterisation of Structural Stationarity

Structural Stationarity is Hard to Prove
Is there a characterisation?

Theorem (Characterisation via $\|\cdot\|_{\mathcal{S}}$)
A process is structurally stationary if and only if the number of sequential processes in every reachable fragment is bounded, i.e.,
$$\exists k \in \mathbb{N} : \forall Q \in \text{Reach}(P) : \forall F \in \text{fg}(\text{rf}(Q)) : ||F||_{\mathcal{S}} \leq k.$$
A First Characterisation of Structural Stationarity

Structural Stationarity is Hard to Prove
Is there a characterisation?

Theorem (Characterisation via $|\ )$

A process is **structurally stationary** if and only if the number of sequential processes in every reachable fragment is bounded, i.e.,

$$\exists k \in \mathbb{N} : \forall Q \in \text{Reach}(P) : \forall F \in f_{\text{fg}}(r_{\text{f}}(Q)) : \|F\|_S \leq k.$$
Corollary (Restriction-free Processes are Structurally Stationary)

Fragments are sequential processes: bound 1
Applications of the Characterisation

Corollary (Restriction-free Processes are Structurally Stationary)

Fragments are sequential processes: bound 1

Definition (Finitary Process [MP95a, Pis99, MP01])

A process is finitary, if the number of sequential processes in every reachable process is bounded:

$$\exists k \in \mathbb{N} : \forall Q \in \text{Reach}(P) : \| Q \|_S \leq k.$$
Applications of the Characterisation

Corollary (Restriction-free Processes are Structurally Stationary)

Fragments are sequential processes: bound 1

Definition (Finitary Process [MP95a, Pis99, MP01])

A process is finitary, if the number of sequential processes in every reachable process is bounded:

$$\exists k \in \mathbb{N} : \forall Q \in \text{Reach}(P) : \|Q\|_S \leq k.$$

Corollary (Finitary Processes are Structurally Stationary)

Take $k$ as bound on number of sequential processes in fragments:

$$\|F\|_S \leq \|rf(Q)\|_S = \|Q\|_S \leq k.$$
A Second Characterisation of Structural Stationarity

Understanding Structural Stationarity

- Which processes are not structurally stationary?
- Is there a characterisation in terms of $\nu$?
A Second Characterisation of Structural Stationarity

Understanding Structural Stationarity

- Which processes are not structurally stationary?
- Is there a characterisation in terms of $\nu$?

Example

A server with local control channel $l$ is not structurally stationary:

\[
\begin{array}{c}
\text{server}
\end{array}
\rightarrow
\begin{array}{c}
\text{server}
\end{array}
\rightarrow
\begin{array}{c}
\text{server}
\end{array}
\rightarrow
\begin{array}{c}
\text{server}
\end{array}
W
\rightarrow
\ldots
\]

Roland Meyer (University of Oldenburg)
Breadth of Processes

Breadth
Maximal number of sequential processes sharing a restricted name

Example

For $F := \begin{array}{c|c}
    & W \\
\hline
W & \\
\end{array}$ we have $\|F\|_B = 3$. 
Breadth of Processes

**Breadth**
Maximal number of sequential processes sharing a restricted name

**Example**

```
  /\  W
 W
```

For $F := W$ we have $\|F\|_B = 3$.

**Problem**
Boundedness in breadth does not ensure structural stationarity
Depth of Processes

Example

Lists are bounded in breadth but not structurally stationary:

\[ LE \rightarrow LI \begin{array}{c} id_1 \end{array} LE \rightarrow LI \begin{array}{c} id_1 \end{array} LI \begin{array}{c} id_2 \end{array} LE \rightarrow \ldots \]
Depth of Processes

Example

Lists are bounded in breadth but not structurally stationary:

\[
LE \rightarrow LI \text{id}_1 \rightarrow LI \text{id}_2 \rightarrow \ldots
\]

Depth

- Minimal nesting of restrictions in the congruence class
- Corresponds to the length of the longest simple path

Example

For \( F := LI \text{id}_1 LI \text{id}_2 LE \) we have \( \| F \|_D = 2 \).
A process is **structurally stationary** if and only if it is **bounded in breadth** and bounded in **depth**.
Decidability in Bounded Depth

Theorem

If a process is bounded in depth, termination and infinity of states are decidable.
Decidability in Bounded Depth

**Theorem**

If a process is bounded in depth, termination and infinity of states are decidable.

**Example**

Server with control channel $l$ is bounded in depth by 3

![Diagram of server with control channel](attachment:image.png)
Decidability in Bounded Depth

**Theorem**

*If a process is bounded in depth, termination and infinity of states are decidable.*

**Example**

Server with control channel $I$ is bounded in depth by 3

\[
\begin{array}{c}
\text{server} \quad \text{ip}_1 \\
\downarrow \\
T \\
\downarrow \\
I \quad T \\
\downarrow \\
\text{ip}_2 \quad \text{client}
\end{array}
\]
Decidability in Bounded Depth

Theorem

*If a process is bounded in depth, termination and infinity of states are decidable.*

Example

Server with control channel \( l \) is bounded in depth by 3
Decidability in Bounded Depth

Theorem

If a process is bounded in depth, termination and infinity of states are decidable.

Example

Server with control channel $l$ is bounded in depth by 3

\[ \text{Server with control channel } l \text{ is bounded in depth by 3} \]
Well-Structured Transition Systems

WSTS [Fin90, FS01, AČJT00]

- Framework for infinite state systems
- Generalises decidability results for particular models
Well-Structured Transition Systems

WSTS [Fin90, FS01, AČJT00]
- Framework for infinite state systems
- Generalises decidability results for particular models

Technically: WSTS = \((S, \rightarrow, \leq)\)
- \((S, \rightarrow)\) is a transition system
- \(\leq \subseteq S \times S\) is a simulation relation and a well-quasi-ordering
Well-Structured Transition Systems

WSTS [Fin90, FS01, AČJT00]

- Framework for infinite state systems
- Generalises decidability results for particular models

Technically: WSTS = (S, →, ≤)

- (S, →) is a transition system
- ≤ ⊆ S × S is a simulation relation and a well-quasi-ordering
Well-Structured Transition Systems

**WSTS** [Fin90, FS01, AČJT00]
- Framework for infinite state systems
- Generalises decidability results for particular models

Technically: **WSTS** = \((S, \rightarrow, \leq)\)
- \((S, \rightarrow)\) is a transition system
- \(\leq \subseteq S \times S\) is a simulation relation and a well-quasi-ordering

\(\leq \subseteq S \times S\) is a Well-Quasi-Ordering
Every infinite sequence contains two comparable states

\[
S_0 \rightarrow S_1 \rightarrow \ldots \rightarrow S_i \rightarrow \ldots \rightarrow S_j \rightarrow \ldots
\]
Instantiation of the Framework—The Ordering $\leq_P$

**Intuition**

Use hypergraph embedding as ordering

**Example**

![Diagram](image-url)
Instantiation of the Framework—The Ordering $\leq_p$

**Intuition**
Use hypergraph embedding as ordering

**Example**

```
Roland Meyer (University of Oldenburg)
Structural Stationarity in the $\pi$-Calculus 15 / 20
```
Instantiation of the Framework—The Ordering $\leq_P$

**Intuition**
Use hypergraph embedding as ordering

**Example**

```
  ip  T  l
```

$\leq_P$

```
  ip  T  W
```

**Technically**

Fragments may be added

$$\nu a.(F | G) \leq_P \nu a.(F | G | H)$$
Instantiation of the Framework—The Ordering $\preceq_P$

**Intuition**
Use hypergraph embedding as ordering

**Example**

```
\begin{align*}
\nu a.(F | G) \preceq_P & \nu a.(F | G | H)
\end{align*}
```

**Technically**
Fragments may be added
Theorem

If $P$ is a process of bounded depth, then $(\text{Reach}(P)/\equiv, \rightarrow, \preceq_P)$ is a well-structured transition system.
Decidability Results for WSTS [Fin90, FS01, AČJT00]

**Finite Reachability Tree**

- **Build computation tree** (finite branching)
- If a new node covers predecessor stop and mark node by +

Theorem ([Fin90, FS01, AČJT00])

Non-terminating computation exists iff tree contains + node.

Infinite state iff + node is strictly larger.
Decidability Results for WSTS [Fin90, FS01, AČJT00]

**Finite Reachability Tree**

- **Build computation tree** (finite branching)
- If a new node covers predecessor stop and mark node by $+$

Theorem ([Fin90, FS01, AČJT00])

Non-terminating computation exists iff tree contains $+$ node.

Infinite state iff $+$ node is strictly larger.
Decidability Results for WSTS [Fin90, FS01, AČJT00]

Finite Reachability Tree

- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by +

Theorem ([Fin90, FS01, AČJT00])
Non-terminating computation exists iff tree contains + node.
Infinite state iff + node is strictly larger.
Decidability Results for WSTS [Fin90, FS01, AČJT00]

**Finite Reachability Tree**

- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by +

![Finite Reachability Tree Diagram](image-url)
Finite Reachability Tree

- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by $+$

\[ S \quad \rightarrow \quad t^+ \]

Theorem ([Fin90, FS01, AČJT00])

*Non-terminating* computation exists *iff* tree contains $+$ node.
Decidability Results for WSTS [Fin90, FS01, AČJT00]

**Finite Reachability Tree**

- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by +

![Finite Reachability Tree Diagram](image)

**Theorem ([Fin90, FS01, AČJT00])**

Non-terminating computation exists iff tree contains + node.

Infinite state iff + node is strictly larger.
Application to the Client-Server System

Build the computation tree

\[ \ldots \rightarrow I \xrightarrow{T} ip \rightarrow \ldots \rightarrow I \xrightarrow{W} T \xrightarrow{ip} \]
Application to the Client-Server System

Build the computation tree

Results

System does not terminate and is infinite state.
Application to the Client-Server System

Build the computation tree

Results
System does not terminate and is infinite state.
### Related Work

**Processes as Graphs**
Due to Milner [Mil79, MM79, MPW92, Mil99, SW01]

**Automata-theoretic Semantics**
- Concurrency [Eng96, MP01, AM02, BG08, DKK06]
- Structure [MP95b]

**Model Checking Tools**
MWB [VM94, Dam96], HAL [FGMP03], SLMC [Cai04]

**Normal Forms**
Decidability of structural congruence [EG99, EG04a, EG04b, EG07]
Survey
References I

Algorithmic analysis of programs with well quasi-ordered domains.

R. M. Amadio and C. Meyssonnier.
On decidability of the control reachability problem in the asynchronous $\pi$-calculus.

J. Bauer.
*Analysis of Communication Topologies by Partner Abstraction*.

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

N. Busi, M. Gabbrielli, and G. Zavattaro.
Replication vs. recursive definitions in channel based calculi.
N. Busi, M. Gabbrielli, and G. Zavattaro.
Comparing recursion, replication, and iteration in process calculi.

N. Busi, M. Gabbrielli, and G. Zavattaro.
On the expressive power of recursion, replication, and iteration in process calculi.

J. Bauer, T. Toben, and B. Westphal.
Mind the shapes: Abstraction refinement via topology invariants.

L. Caires.
Behavioural and spatial observations in a logic for the \( \pi \)-Calculus.


M. Dam.
Model checking mobile processes.
C. Dufourd, A. Finkel, and Ph. Schnoebelen.
Reset nets between decidability and undecidability.

R. Demangeon, D. Hirschkoff, and D. Sangiorgi.
Static and dynamic typing for the termination of mobile processes.

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

G. Delzanno, J.-F. Raskin, and L. Van Begin.
Towards the automated verification of multithreaded java programs.

Y. Deng and D. Sangiorgi.
Ensuring termination by typability.
References IV

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

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.

A model-checking verification environment for mobile processes.
A. Finkel.
Reduction and covering of infinite reachability trees.

A. Finkel and Ph. Schnoebelen.
Well-structured transition systems everywhere!

G. Higman.
Ordering by divisibility in abstract algebras.

B. König and V. Kozioura.
Counterexample-guided abstraction refinement for the analysis of graph transformation systems.

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

R. Milner.
Flowgraphs and flow algebras.
R. Milner.
*Communicating and Mobile Systems: the π-Calculus.*

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

U. Montanari and M. Pistore.
Checking bisimilarity for finitary π-calculus.

U. Montanari and M. Pistore.
Concurrent semantics for the π-calculus.

U. Montanari and M. Pistore.
History dependent automata.

R. Milner, J. Parrow, and D. Walker.
A calculus of mobile processes, part I.
F. Orava and J. Parrow.
An algebraic verification of a mobile network.

M. Pistore.
*History Dependent Automata*.

J.-F. Raskin and L. Van Begin.
Petri nets with non-blocking arcs are difficult to analyze.

A. Rensink.
Canonical graph shapes.

D. Sangiorgi and D. Walker.
*The π-calculus: a Theory of Mobile Processes*.

T. Toben.
*Analysis of Dynamic Evolution Systems by Spotlight Abstraction Refinement*.
B. Victor and F. Moller.
The mobility workbench: A tool for the $\pi$-calculus.
MWB: http://www.it.uu.se/research/group/mobility/mwb, last access 2008-11-28.

B. Westphal.
*Specification and Verification of Dynamic Topology Systems.*

B. Wachter and B. Westphal.
The spotlight principle. on combining process-summarising state abstractions.

N. Yoshida, M. Berger, and K. Honda.
Strong normalisation in the $\pi$-Calculus.
More Related Work

WSTS

- Finkel inspired by Petri nets [Fin90, FS01], termination and boundedness problems
- Abdulla inspired by lossy channel systems [AČJT00], temporal and simulation properties

WSTS and Process Algebras

Replication and recursion in CCS [BGZ03, BGZ04, BGZ08]

Termination

Type systems [YBH04, DS06, DHS08]
More Related Work

**GRS and Verification**
Semi-decision procedures [Bau06, Ren04, KK06]

**AVACS**
- Spotlight abstraction [WW07, Wes08] + invariants [BTW07]
- Refinement cycle [Tob08]

**Extended Petri Nets**
- Petri nets with marking dependent arc cardinalities [DFS98]
- Relation to multithreaded JAVA [DRB02]
- Undecidability of LTL [RB04]
Why is $\preceq_P$ a WQO?

### Proof Idea
Understand fragments as (syntax) trees
- Sequential processes are leaves
- Restricted names are nodes

### Example
$$\nu l. (S [url, dat, l] | \nu ip. (T [l, ip] | C [url, ip]))$$

![Diagram](image.png)
Why is $\preceq_P$ a WQO?

**Proof Idea**

Understand fragments as (syntax) trees
- Sequential processes are **leafs**
- Restricted names are nodes

**Example**

\[
\nu l. ( S[url, dat, l] \mid \nu ip. ( T[l, ip] \mid C[url, ip] ))
\]
Why is $\preceq_P$ a WQO?

**Proof Idea**

Understand fragments as (syntax) trees

- Sequential processes are leaves
- Restricted names are nodes

**Example**

$\nu l . (S[l, dat, l] \mid \nu ip . (T[l, ip] \mid C[url, ip]))$
Why is $\preceq_P$ a WQO?

**Proof Idea**

Use a suitable wqo on trees

**Example**

\[
\begin{array}{c}
S \\
T \\
C
\end{array}
\preceq_T
\begin{array}{c}
S \\
T \\
C
\end{array}
\]

\[
\begin{array}{c}
S \\
T \\
C
\end{array}
\preceq_T
\begin{array}{c}
S \\
T \\
C
\end{array}
W
\]
Why is $\preceq_P$ a WQO?

Proof Idea

Use a suitable wqo on trees

Example

$S \preceq_P T \preceq_P C \preceq_P W$

Roland Meyer (University of Oldenburg)
Why is $\leq_P$ a WQO?

**Proof Idea**

Use a suitable wqo on trees
- Wqo on trees of bounded depth
- Induction on depth $+$ Higman’s result [Hig52]

**Example**

```
  S   T
 /   / \
I   ip C       I   ip W
     T   C       T   C
```

Roland Meyer (University of Oldenburg)  Structural Stationarity in the $\pi$-Calculus 30 / 20
Why Order Name Places?

**Counterexample**

- Consider $\overline{a} \mid K[a]$ with $K(x) := \nu z_0. (\overline{z_0} \mid x.z_0.K[x])$
- Process deadlocks after four steps
Counterexample

- Consider $\overline{a} | K[a]$ with $K(x) := \nu z_0.(\overline{z_0} | x.z_0.K[x])$
- Process deadlocks after four steps
Why Order Name Places?

Counterexample

- Consider $\overline{a} \mid K[a]$ with $K(x) := \nu z_0. (\overline{z_0} \mid x.z_0.K[x])$
- Process deadlocks after four steps
Why Order Name Places?

Counterexample

- Consider $a \mid K[a]$ with $K(x) := \nu z_0. (\overline{z_0} \mid x.z_0.K[x])$
- Process deadlocks after four steps
- Insert dependence between $z_0$ and $z_1$
Why Order Name Places?

Counterexample

- Consider \( \bar{a} \mid K[a] \) with \( K(x) := \nu z_0.(\overline{z_0} \mid x.z_0.K \langle x \rangle) \)
- Process deadlocks after four steps
- Insert dependence between \( z_0 \) and \( z_1 \)
Why Order Name Places?

Counterexample

- Consider \( \overline{a} \mid K[a] \) with \( K(x) := \nu z_0.(\overline{z_0} \mid x.z_0.K[x]) \)
- Process deadlocks after four steps
- Insert dependence between \( z_0 \) and \( z_1 \)
Why Order Name Places?

Counterexample

- Consider \( \overline{a} \parallel K[a] \) with \( K(x) := \nu z_0. (\overline{z_0} \mid x.z_0.K[x]) \)
- Process deadlocks after four steps
- Insert dependence between \( z_0 \) and \( z_1 \)
Why Order Name Places?

### Counterexample

- Consider $\bar{a} \mid K[a]$ with $K(x) := \nu z_0.(\overline{z_0} \mid x.z_0.K[x])$
- Process deadlocks after four steps
- Insert **dependence** between $z_0$ and $z_1$
Why Order Name Places?

Counterexample

- Consider $\bar{a} \mid K[a]$ with $K(x) := \nu z_0.(\overline{z_0} \mid x.z_0.K[x])$
- Process deadlocks after four steps
- Insert dependence between $z_0$ and $z_1$
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]

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

- Create copy \( c' \) of counter \( c \)

Diagram:

- \( l \)
- \( l' \)
- \( l'' \)
- \( c \)
- \( S_{\text{trash}} \)
- \( c' \)

Roland Meyer (University of Oldenburg)
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 \( c \)

\[ \begin{align*}
  l & \quad \text{if } c = 0 \text{ then goto } l'; & \text{ else } c := c - 1; \text{ goto } l''; \\
  c & \quad \text{Create copy } c' \text{ of counter } c \\
  c' & \quad \text{Return}
\end{align*} \]
Undecidability of Reachability in Depth One

Test for Zero in Petri Nets with Transfer [DFS98]

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

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

![Petri Net Diagram]

Roland Meyer (University of Oldenburg) | Structural Stationarity in the \( \pi \)-Calculus
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 \( c' \)

[Diagram of Petri net with states and transitions]

Roland Meyer (University of Oldenburg) Structural Stationarity in the \( \pi \)-Calculus
Undecidability of Reachability in Depth One

Process Bunch

Modify an arbitrary number of processes with one communication

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

Example

\[ \nu a.(PB[a, i, d, t] | \bar{a} | \bar{a}) \]
Undecidability of Reachability in Depth One

Process Bunch
Modify an arbitrary number of processes with one communication

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

Example

\[ \bar{t} \mid \nu a.(t.\nu b.PB[b, i, d, t] + \ldots \mid \bar{a} \mid \bar{a}) \]
Undecidability of Reachability in Depth One

**Process Bunch**

Modify an arbitrary number of processes with one communication

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

**Example**

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

Algorithms avoid costly state space computations:

- Occurrence number properties
  - Use S-invariants
- Temporal properties
  - Inspect graph structure of the Petri net
- Topological properties
  - Inspect set of places (using regular expressions)
Unfolding-based Verification

<table>
<thead>
<tr>
<th>Model</th>
<th>[KKN06] unf</th>
<th>sat</th>
<th>MWB dl</th>
<th>HAL $\pi 2fc$ unf</th>
<th>sat</th>
<th>Struct</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns2</td>
<td>1</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>ns3</td>
<td>7</td>
<td>&lt; 1</td>
<td>1</td>
<td>8</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>ns4</td>
<td>69</td>
<td>1</td>
<td>577</td>
<td>382</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>ns5</td>
<td>532</td>
<td>58</td>
<td>—</td>
<td>—</td>
<td>17</td>
<td>3</td>
</tr>
<tr>
<td>ns6</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>1518</td>
<td>84</td>
</tr>
<tr>
<td>gsm [OP92]</td>
<td>n/a</td>
<td>—</td>
<td>18</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
<td></td>
</tr>
</tbody>
</table>

Verified car platoon and autonomous transport

<table>
<thead>
<tr>
<th>Instance</th>
<th>Struct</th>
<th>Model Checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>T</td>
<td>unf</td>
</tr>
<tr>
<td>1p 6m 4v ref</td>
<td>937</td>
<td>1371</td>
</tr>
</tbody>
</table>

Roland Meyer (University of Oldenburg)
Size of the Translation

Idea

\[ N_1, \ N_2, \ N_3, \ldots \]
Size of the Translation

Idea

\[ N_1, \ N_2, \ N_3, \ldots \]

\[ \mathcal{P}[N_1], \ \mathcal{P}[N_2], \ \mathcal{P}[N_3], \ldots \]

Linear
Size of the Translation

Idea

\[ \mathcal{N}_1, \; \mathcal{N}_2, \; \mathcal{N}_3, \ldots \]

linear

\[ \mathcal{P}[\mathcal{N}_1], \; \mathcal{P}[\mathcal{N}_2], \; \mathcal{P}[\mathcal{N}_3], \ldots \]

\[ \mathcal{N}_i \text{ state yields } \mathcal{N}[\mathcal{P}[\mathcal{N}_i]] \text{ place} \]

\[ \mathcal{N}[\mathcal{P}[\mathcal{N}_1]], \; \mathcal{N}[\mathcal{P}[\mathcal{N}_2]], \; \mathcal{N}[\mathcal{P}[\mathcal{N}_3]], \ldots \]
Size of the Translation

**Idea**

\[ N_1, N_2, N_3, \ldots \]

linear

\[ P[N_1], P[N_2], P[N_3], \ldots \]

\[ N_i \text{ state yields } N[P[N_i]] \text{ place} \]

\[ N[P[N_1]], N[P[N_2]], N[P[N_3]], \ldots \]

**Theorem (Size of the Structural Semantics)**

The size of the structural semantics \( N[P] \) is not bounded by a primitive recursive function in the size of the process \( P \).
Technically

\[ \nu \text{act.} (\overline{\text{act}} \mid \text{act.} K_t (\overline{\text{act}}) \mid s_1 (\overline{\text{act}})) \]

\[ K_t (\text{act}, s_1, s_2) := s_1 (x). (\overline{\text{act}} \mid \text{act.} K_t (\overline{\text{act}}) \mid s_2 (\overline{\text{act}})) \]
Technically

\[
\nu \text{act.}(\overline{\text{act}} \mid \text{act. } K_t[\text{act}, s_1, s_2] \mid \overline{s_1}(\text{act}))
\]

\[
K_t(\text{act}, s_1, s_2) := s_1(x).\overline{(\text{act} \mid \text{act. } K_t[\text{act}, s_1, s_2] \mid \overline{s_2}(\text{act}))}
\]
Hierarchy of Processes

- structurally stationary
- restriction-free
- restriction-bounded
- mixed-bounded (p/t Petri nets)
- bounded depth (WSTS)
- bounded breadth (2CM)
Hierarchy of Processes

- Restriction-free
- Structurally stationary
- Mixed-bounded (p/t Petri nets)
- Restriction-bounded
- Bounded depth (WSTS)
- Bounded breadth (2CM)
Hierarchy of Processes

bounded breadth (2CM)  
bounded depth (WSTS)  
mixed-bounded (p/t Petri nets)
structurally stationary  
restriction-bounded  
restriction-free
Finite Handler Processes—Participants

- Register at handler processes via distinguished public channels

Example

- Channel \textit{cfa} is a distinguished name
- A free agent is a participant:

\[ \nu id, ca, rq. \overline{cfa\langle id \rangle} \cdot \overline{id\langle ca \rangle} \cdot \overline{id\langle rq \rangle} \ldots \]
Finite Handler Processes—Participants

- Register at handler processes via distinguished public channels
- Continue to communicate via private names only

Example

- Channel \( cfa \) is a distinguished name
- A free agent is a participant:

\[
\nu id, ca, rq. \overline{cfa}\langle id\rangle. \overline{id}\langle ca\rangle. \overline{id}\langle rq\rangle \ldots
\]
Finite Handler Processes—Handler

- **Listen** on the distinguished channels

**Example**

The *MRG* process is a handler

\[
\text{cfa}(id_x).id_x(ca_x) \ldots \text{cfa}(id_y) \ldots id_y(rq_y).\overline{ca_x} \langle rq_y \rangle.MRG[\text{cfa}]
\]
Finite Handler Processes—Handler

- Listen on the distinguished channels
- Receive \textit{finitely many} processes

Example

The \textit{MRG} process is a handler

\[
\text{cfa}(id_x).id_x(ca_x)\ldots \text{cfa}(id_y)\ldots id_y(rq_y).\overline{ca_x}\langle rq_y \rangle.MRG[cfa]
\]
Finite Handler Processes—Handler

- Listen on the distinguished channels
- Receive finitely many processes
- Communicate with the registered participants only

Example

The *MRG* process is a handler

\[ \text{cfa}(id_x).id_x(ca_x) \ldots \text{cfa}(id_y) \ldots id_y(rq_y). \overline{ca_x}\langle rq_y \rangle . MRG[cfa] \]
Theorem

*Finite handler processes are structurally stationary.*

The car platooning example is a finite handler process