On Boundedness in Depth in the π-Calculus

Roland Meyer

University of Oldenburg

2008-02-09

A Client/Server System in the π-Calculus

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

Graphically

C — $\boxed{ip}$

S — $\boxed{l}$

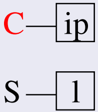
In π-Calculus

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

A Client/Server System in the π-Calculus

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

Graphically



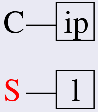
In π-Calculus

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

A Client/Server System in the π-Calculus

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

Graphically



In π-Calculus

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

$$\nu l.url(y).(\nu sn.\overline{y}\langle sn\rangle.T\lfloor sn, l\rfloor \mid S\lfloor url, l\rfloor)$$

A Client/Server System in the π-Calculus

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

Graphically

$C - \boxed{ip}$

$S - \boxed{1}$

In π-Calculus

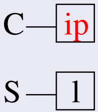
$\nu ip. \overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url, ip\rfloor \mid$

$\nu l. url(y).(\nu sn.\overline{y}\langle sn\rangle.T\lfloor sn, l\rfloor \mid S\lfloor url, l\rfloor)$

A Client/Server System in the π-Calculus

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

Graphically



In π-Calculus

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

A Client/Server System in the π-Calculus

Server spawns a new thread that handles the session with the client

Graphically

$\mathrm{C} - \boxed{\mathrm{ip}}$

$\mathrm{S} - \boxed{1}$

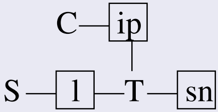
In π-Calculus

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

A Client/Server System in the π-Calculus

Thread sends back a private session *sn* on the private channel *ip*

Graphically



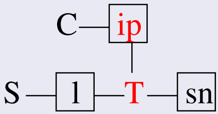
In π-Calculus

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

A Client/Server System in the π-Calculus

Thread sends back a private session sn on the private channel ip

Graphically



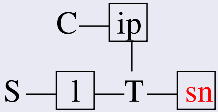
In π-Calculus

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

A Client/Server System in the π-Calculus

Thread sends back a private session *sn* on the private channel *ip*

Graphically



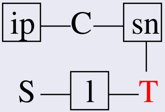
In π-Calculus

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

A Client/Server System in the π-Calculus

The thread switches its mode

Graphically



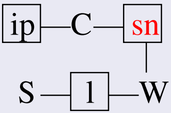
In π-Calculus

$$\nu sn.(\nu ip.sn(x).C\lfloor url, ip\rfloor \mid \nu l.(T\lfloor sn, l\rfloor \mid S\lfloor url, l\rfloor))$$

A Client/Server System in the π-Calculus

To terminate the session, the thread sends the private session object *sn* on the channel *sn* itself

Graphically



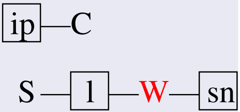
In π-Calculus

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

A Client/Server System in the π-Calculus

The thread writes back information to the server and terminates, the client is ready to contact the server again

Graphically



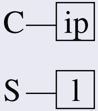
In π-Calculus

$$\nu ip.C\lfloor url, ip\rfloor \mid$$
$$\nu l.(\nu sn.W\lfloor l, sn\rfloor \mid S\lfloor url, l\rfloor)$$

A Client/Server System in the π-Calculus

This yields the initial state

Graphically



In π-Calculus

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

Motivation and Contribution

Motivation: Verification of dynamically reconfigurable systems

- Does the client/server system terminate?
- Is it finite state?

Contribution: This can be done automatically

- For systems of bounded depth

Overview

1. A client/server system in the π-Calculus ✓
2. Systems of bounded depth
3. From bounded depth to well-structured transition systems
4. Decidability results

A normal form for processes

- Normalise the process
  - Minimise the scopes of restrictions
  - Yields parallel composition of fragments F, G

Example

$$\nu l.\nu sn.\nu sn'.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid W\lfloor l, sn'\rfloor)$$

A normal form for processes

- Normalise the process
  - Minimise the scopes of restrictions
  - Yields parallel composition of fragments F, G

Example

$$\nu l.\nu sn.\, \nu sn'.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid W\lfloor l, sn'\rfloor)$$

A normal form for processes

- Normalise the process
  - Minimise the scopes of restrictions
  - Yields parallel composition of fragments F, G

Example

$$\begin{aligned} & \nu l.\nu sn.\boxed{\nu sn'}.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid \boxed{W\lfloor l, sn'\rfloor}) \\ \equiv \quad & \nu l.\nu sn.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid \boxed{\nu sn'}.\boxed{W\lfloor l, sn'\rfloor}) \end{aligned}$$

A normal form for processes

- Normalise the process
  - Minimise the scopes of restrictions
  - Yields parallel composition of fragments F, G

Example

$$\begin{aligned} & \nu l.\nu sn.\nu sn'.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid W\lfloor l, sn'\rfloor) \\ \equiv\ & \nu l.\nu sn.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid \nu sn'.W\lfloor l, sn'\rfloor) \end{aligned}$$

A normal form for processes

- Normalise the process
  - Minimise the scopes of restrictions
  - Yields parallel composition of fragments F, G

Example

$$
\begin{aligned}
& \nu l.\nu sn.\nu sn'.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid W\lfloor l, sn'\rfloor) \\
\equiv\ & \nu l.\boxed{\nu sn}.(S\lfloor url, l\rfloor \mid \boxed{W\lfloor l, sn\rfloor} \mid \nu sn'.W\lfloor l, sn'\rfloor) \\
\equiv\ & \nu l.(S\lfloor url, l\rfloor \mid \boxed{\nu sn}.\boxed{W\lfloor l, sn\rfloor} \mid \nu sn'.W\lfloor l, sn'\rfloor)
\end{aligned}
$$

A normal form for processes

- Normalise the process
  - Minimise the scopes of restrictions
  - Yields parallel composition of fragments F, G

Example

$$
\begin{aligned}
& \nu l.\nu sn.\nu sn'.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid W\lfloor l, sn'\rfloor) \\
\equiv\; & \nu l.\nu sn.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid \nu sn'.W\lfloor l, sn'\rfloor) \\
\equiv\; & \nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor \mid \nu sn'.W\lfloor l, sn'\rfloor)
\end{aligned}
$$

The latter process is a fragment

The nesting of restrictions

- Count the nesting of restrictions in the fragment

Example

$$nest_\nu\left(\nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor \mid \nu sn'.W\lfloor l, sn'\rfloor)\right)$$

The nesting of restrictions

- Count the nesting of restrictions in the fragment

Example

$$\begin{aligned} & nest_\nu \left(\nu l.(\ S\lfloor url, l \rfloor \mid \nu sn.W\lfloor l, sn \rfloor \mid \nu sn'.W\lfloor l, sn' \rfloor \ ) \right) \\ = \ & 1 + max\{\ldots\} \end{aligned}$$

The nesting of restrictions

- Count the nesting of restrictions in the fragment

Example

$$\begin{aligned} & nest_\nu\left(\nu l.(\,S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor \mid \nu sn'.W\lfloor l, sn'\rfloor)\right) \\ = \;& 1 + max\{0, \ldots\} \end{aligned}$$

The nesting of restrictions

- Count the nesting of restrictions in the fragment

Example

$$\begin{aligned} & nest_\nu \left(\nu l.(S\lfloor url, l\rfloor \mid \boxed{\nu sn.W\lfloor l, sn\rfloor} \mid \nu sn'.W\lfloor l, sn'\rfloor)\right) \\ = \quad & 1 + max\{0, 1, \ldots\} \end{aligned}$$

The nesting of restrictions

- Count the nesting of restrictions in the fragment

Example

$$\begin{aligned} & nest_\nu\left(\nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor \mid \boxed{\nu sn'.W\lfloor l, sn'\rfloor)}\right) \\ = \;& 1 + max\{0, 1, 1\} \\ = \;& 2 \end{aligned}$$

The nesting of restrictions

- Count the nesting of restrictions in the fragment
- Problem: The fragment representation is not unique

Example

$$
\begin{aligned}
& \nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor \mid \nu sn'.W\lfloor l, sn'\rfloor) =: F \\
\equiv\ & \nu sn.\nu sn'.\nu l.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid W\lfloor l, sn'\rfloor) =: G
\end{aligned}
$$

Then $nest_\nu(F) = 2$ but $nest_\nu(G) = 3$

The nesting of restrictions

- Count the nesting of restrictions in the fragment
- Problem: The fragment representation is not unique

Example

$$\nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor \mid \nu sn'.W\lfloor l, sn'\rfloor) =: F$$
$$\equiv \quad \nu sn.\nu sn'.\nu l.(S\lfloor url, l\rfloor \mid W\lfloor l, sn\rfloor \mid W\lfloor l, sn'\rfloor) =: G$$

Then $nest_\nu(F) = 2$ but $nest_\nu(G) = 3$

Boundedness in depth

- Solution: Define the depth of a fragment as the nesting of restrictions in the flattest representation

Depth

$$depth(F) = min\,\{nest_\nu(G) \mid G \equiv F\}$$

Boundedness in depth

- Solution: Define the depth of a fragment as the nesting of restrictions in the flattest representation

Depth

$$depth(F) = min\{nest_{\nu}(G) \mid G \equiv F\}$$

- A process is bounded in depth if the depth of all reachable fragments is bounded

A Characterisation of Boundedness in Depth

- Problem: No good intuition to processes of bounded depth
- How to find the flat representation?

A Characterisation of Boundedness in Depth

Theorem

A process is bounded in depth if and only if

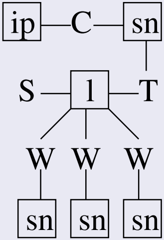
the length of the longest simple paths in the graphs is bounded

- Simple paths do not repeat hyperedges
- Reachable states are star-like
- Anchored fragments are flat representations

Example: The Client/Server System

- In the case study, the depth is bounded by 4

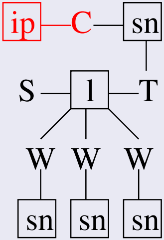
A longest simple path



Example: The Client/Server System

- In the case study, the depth is bounded by 4

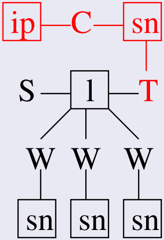
A longest simple path



Example: The Client/Server System

- In the case study, the depth is bounded by 4

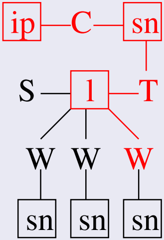
A longest simple path



Example: The Client/Server System

- In the case study, the depth is bounded by 4

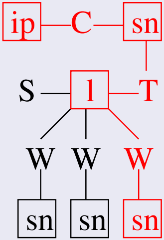
A longest simple path



Example: The Client/Server System

- In the case study, the depth is bounded by 4

A longest simple path



More Examples

All decidable subclasses of π-Calculus are bounded in depth

- Finitary agents [FGMP03], finite control processes [Dam96], bounded processes [Cai04] (finite state systems)
- Structurally stationary processes [Mey08], finite handler processes [Mey08], restriction-free processes [AM02] (Petri nets)
- Bounded input unique receiver systems [AM02] (subclass of transfer nets)
- Finite net processes [BG95, BG08] (subclass of inhibitor nets)

Well-Structured Transition Systems

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

Technically

WSTS=$(S, \rightarrow, \leq)$ where

- $(S, \rightarrow)$ is a transition system
- $\leq \subseteq S \times S$ is an ordering on the states with two properties

Well-Structured Transition Systems

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

Technically

WSTS$=(S, \rightarrow, \leq)$ where

- $(S, \rightarrow)$ is a transition system
- $\leq \subseteq S \times S$ is an ordering on the states with two properties

Well-Structured Transition Systems

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

Technically

WSTS=$(S, \rightarrow, \leq)$ where

- $(S, \rightarrow)$ is a transition system
- $\leq \subseteq S \times S$ is an ordering on the states with two properties

Well-Structured Transition Systems

$\leq \subseteq S \times S$ is a well-quasi-ordering

In every infinite sequence of states, there are two comparable ones

$$\leq$$

$$s_0 \longrightarrow s_1 \longrightarrow \dots \longrightarrow s_i \longrightarrow \dots \longrightarrow s_j \longrightarrow \dots$$

Well-Structured Transition Systems

$\leq \subseteq S \times S$ is a well-quasi-ordering

In every infinite sequence of states, there are two comparable ones

$$\begin{tikzcd} s_0 \arrow[r] & s_1 \arrow[r] & \dots \arrow[r] & s_i \arrow[rr, dash, bend left, "\leq"] \arrow[r] & \dots \arrow[r] & s_j \arrow[r] & \dots \end{tikzcd}$$

$\leq \subseteq S \times S$ is a simulation

Larger states can imitate the transition behaviour of smaller ones

$$\begin{tikzcd} s \arrow[r] & s' & & s' \\ t \arrow[u, phantom, "\rotatebox{90}{$\leq$}"] & \Rightarrow & t \arrow[r] & t' \arrow[u, phantom, "\rotatebox{90}{$\leq$}"] \end{tikzcd}$$

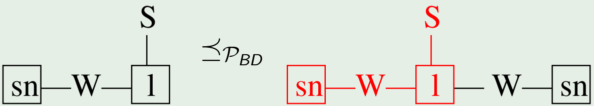
Instantiation of the framework—the ordering $\preceq_{\mathcal{P}_{BD}}$

- Intuitively: Hypergraph embedding so that
  - No connections are added to vertices
  - No connections are removed from vertices

Instantiation of the framework—the ordering $\preceq_{\mathcal{P}_{BD}}$

- Intuitively: Hypergraph embedding so that
  - No connections are added to vertices
  - No connections are removed from vertices

Example



Instantiation of the framework—the ordering $\preceq_{\mathcal{P}_{BD}}$

- Intuitively: Hypergraph embedding so that
  - No connections are added to vertices
  - No connections are removed from vertices
- Technically: Parallel composition of fragments may be added

$$\nu a.(F \mid G) \preceq_{\mathcal{P}_{BD}} \nu a.(F \mid G \mid H)$$

Instantiation of the framework—the ordering $\preceq_{\mathcal{P}_{BD}}$

- Intuitively: Hypergraph embedding so that
  - No connections are added to vertices
  - No connections are removed from vertices
- Technically: Parallel compositions of fragments may be added

$$\nu a.(F \mid G) \preceq_{\mathcal{P}_{BD}} \nu a.(F \mid G \mid \boxed{H})$$

Example

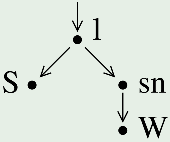
$$\begin{aligned} & \nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor) \\ \preceq_{\mathcal{P}_{BD}} \quad & \nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor \mid \boxed{\nu sn'.W\lfloor l, sn'\rfloor}) \end{aligned}$$

Why is it a wqo?

- Understand fragments as (syntax) trees
  - Processes are leafs
  - Restricted names are nodes

Example

$$\nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor) \quad \rightsquigarrow$$

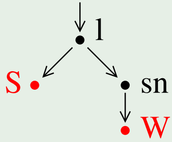


Why is it a wqo?

- Understand fragments as (syntax) trees
  - Processes are leafs
  - Restricted names are nodes

Example

$$\nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor) \rightsquigarrow$$

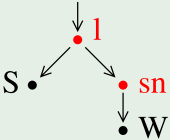


Why is it a wqo?

- Understand fragments as (syntax) trees
  - Processes are leafs
  - Restricted names are nodes

Example

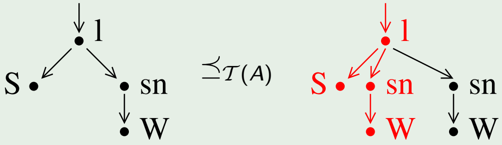
$$\nu l.(S\lfloor url, l\rfloor \mid \nu sn.W\lfloor l, sn\rfloor) \rightsquigarrow$$



Why is it a wqo?

- Understand fragments as (syntax) trees
  - Processes are leafs
  - Restricted names are nodes
- Use a suitable wqo on trees

Example



- Wqo on trees of bounded depth
- Induction on depth + Higman's result [Hig52]

Main Result

- Simulation is easier to prove

Theorem

If P is a process of bounded depth, then $(\mathit{Reach}(P)/_{\equiv}, \rightarrow, \preceq_{\mathcal{P}_{BD}})$ is a well-structured transition system.

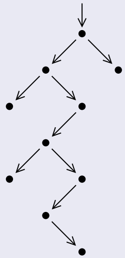
Decidability Results for WSTS [Fin90, FS01, AČJT00]

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

Decidability Results for WSTS [Fin90, FS01, AČJT00]

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

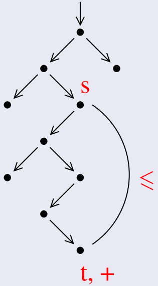
The Finite Reachability Tree



Decidability Results for WSTS [Fin90, FS01, AČJT00]

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

The Finite Reachability Tree



Decidability Results for WSTS [Fin90, FS01, AČJT00]

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

Theorem ([Fin90, FS01, AČJT00])

There is a non-terminating computation if and only if the tree contains a node marked by $+$.

Decidability Results for WSTS [Fin90, FS01, AČJT00]

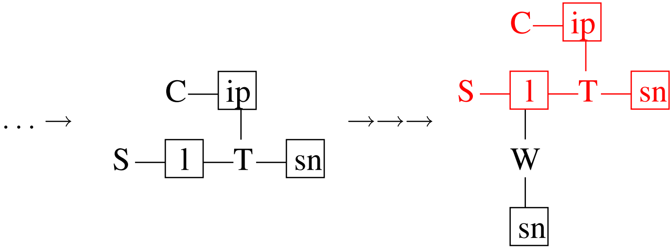
- Build the computation tree (finite branching)
- If a new node covers a predecessor stop the computation, mark the node by $+$

Theorem ([Fin90, FS01, AČJT00])

There is a non-terminating computation if and only if the tree contains a node marked by $+$.
Infinite state if and only if $+$ node is truely bigger

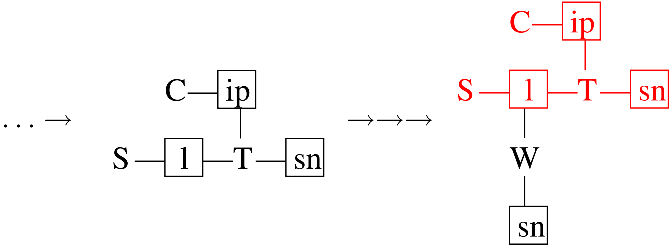
Application to the Client/Server System

- Build the computation tree



Application to the Client/Server System

- Build the computation tree



Results

The system does not terminate, the system is infinite state

Related Work

- Interpretation of processes as graphs due to Milner (flow graphs) [Mil79, MM79]
  - For π-Calculus in [MPW92, Mil99, SW01]
  - We relate depth on terms with the longest simple paths in graphs
- Normal forms for π-Calculus by Engelfriet and Gelsema [EG99, EG04] and Milner [Mil99]
  - Similar to minimising scopes
  - Anchored fragments are more stringent

Related Work

- WSTS by Finkel [Fin90, FS01] and Abdulla et. al. [AČJT00]
  - Finkel inspired by Petri nets, termination and boundedness problems
  - Abdulla inspired by lossy channel systems, temporal and simulation properties
  - First instantiation for π-Calculus
  - $\preceq_{\mathcal{P}_{BD}}$ is non-trivial
- Importance of termination for π-Calculus by Yoshida et. al. [YBH04] and Sangiorgi [DS06]
  - Type systems that ensure termination of well-typed processes
  - Instantiate WSTS framework, derive decidability of termination as corollary

Conclusion and Thanks

Processes of bounded depth are graphs where the longest simple path is bounded

- Star-like structures
- Unbounded parallelism/unboundedly many restricted names

They have well-structured transition systems

- Termination is decidable
- Infinity of states is decidable

The class is huge

- Contains all decidable subclasses of π-Calculus known so far [BG95, Dam96, AM02, FGMP03, Cai04, Mey08, BG08]

Conclusion and Thanks

Thanks for your attention

References I

P. A. Abdulla, K. Čerans, B. Jonsson, and Y.-K. Tsay.
Algorithmic analysis of programs with well quasi-ordered domains.
Information and Computation, 160(1–2):109–127, 2000.

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

N. Busi and R. Gorrieri.
A Petri net semantics for π-calculus.
In *Proc. of the 6th International Conference on Concurrency Theory, CONCUR 1995*, volume 962 of *LNCS*, pages 145–159. Springer-Verlag, 1995.

N. Busi and R. Gorrieri.
Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs.
To appear in JLAP, August 2008.

L. Caires.
Behavioural and spatial observations in a logic for the π-Calculus.
In *Proc. of the 7th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2004*, volume 2987 of *LNCS*, pages 72–89. Springer-Verlag, 2004.
Spatial Logic Model Checker: http://ctp.di.fct.unl.pt/SLMC/.

References II

M. Dam.
Model checking mobile processes.
Information and Computation, 129(1):35–51, 1996.

Y. Deng and D. Sangiorgi.
Ensuring termination by typability.
Information and Computation, 204(7):1045–1082, 2006.

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

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

G.-L. Ferrari, S. Gnesi, U. Montanari, and M. Pistore.
A model-checking verification environment for mobile processes.
ACM Transactions on Software Engineering and Methodology, 12(4):440–473, 2003.
HAL: http://fmt.isti.cnr.it:8080/hal/.

A. Finkel.
Reduction and covering of infinite reachability trees.
Information and Computation, 89(2):144–179, 1990.

References III

A. Finkel and Ph. Schnoebelen.
Well-structured transition systems everywhere!
Theoretical Computer Science, 256(1–2):63–92, 2001.

G. Higman.
Ordering by divisibility in abstract algebras.
Proc. London Math. Soc. (3), 2(7):326–336, 1952.

R. Meyer.
A theory of structural stationarity in the π-calculus.
55 pages, submitted for publication, June 2008.

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.
Cambridge University Press, 1999.

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

References IV

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

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

N. Yoshida, M. Berger, and K. Honda.
Strong normalisation in the π-Calculus.
Information and Computation, 191(2):145–202, 2004.