Structural Stationarity in the π-Calculus

Roland Meyer

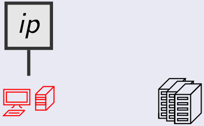
Department of Computing Science
University of Oldenburg

Disputation
2009-02-20

A Client-Server System in the π-Calculus

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

Graphically

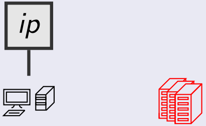


In π-Calculus

A Client-Server System in the π-Calculus

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

Graphically

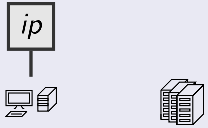


In π-Calculus

A Client-Server System in the π-Calculus

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

Graphically



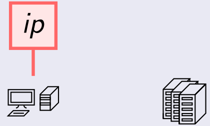
In π-Calculus

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

A Client-Server System in the π-Calculus

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

Graphically



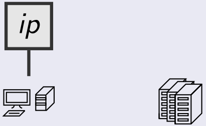
In π-Calculus

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

A Client-Server System in the π-Calculus

In response server spawns a new thread

Graphically



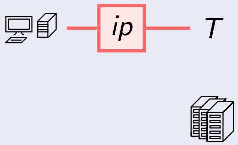
In π-Calculus

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

A Client-Server System in the π-Calculus

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

Graphically



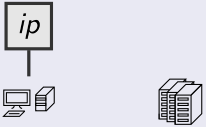
In π-Calculus

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

A Client-Server System in the π-Calculus

Thread terminates, client is ready to contact server again

Graphically



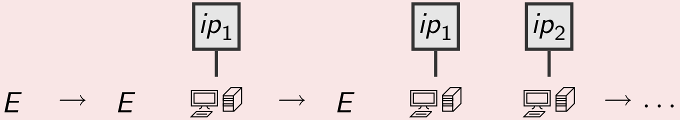
In π-Calculus

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

A Client-Server System in the π-Calculus

Assumption

Environment E generates clients



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

Goal

Automatic verification of dynamically reconfigurable systems

- Occurrence number properties: Is there exactly one server?
- Temporal properties: Does a request create a new thread?

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

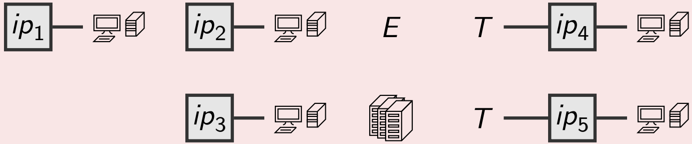
Translate π-Calculus into place/transition Petri nets

Overview

Idea of the Structural Semantics

Problem

Unbounded number of clients and threads



Observation

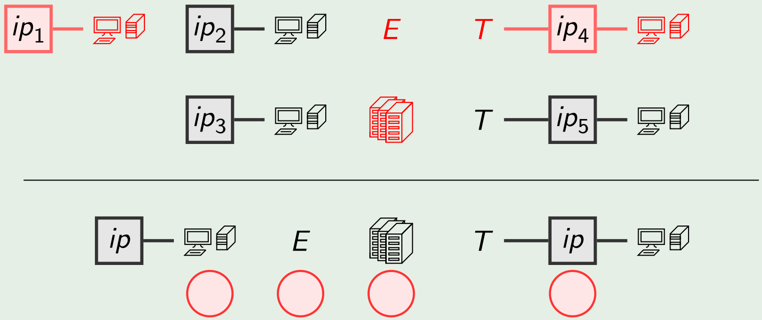
Finite number of connection patterns

Idea of the Structural Semantics

Represent Connections in a Petri net

- Every connection pattern yields a place

Example

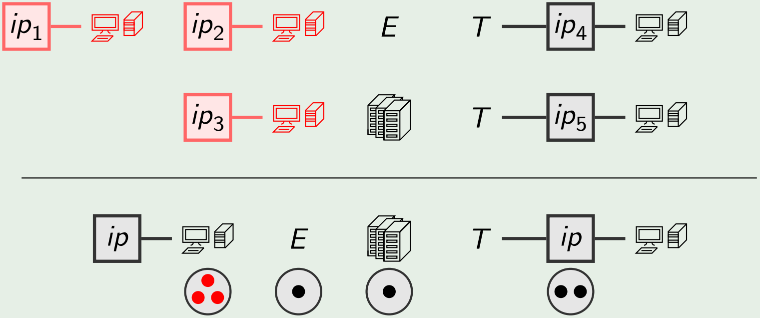


Idea of the Structural Semantics

Represent Connections in a Petri net

- Every connection pattern yields a place
- Every occurence of the pattern yields a token

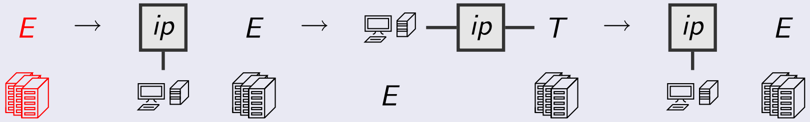
Example



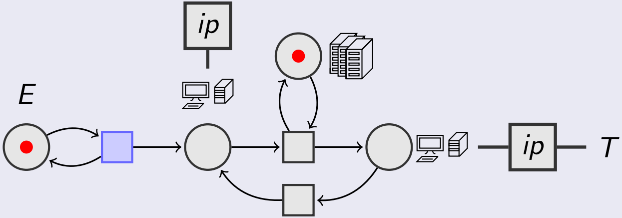
Idea of the Structural Semantics

Transitions model the evolution of patterns

In π-Calculus



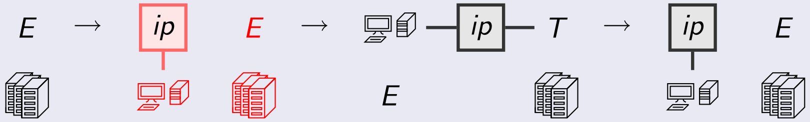
The Structural Semantics



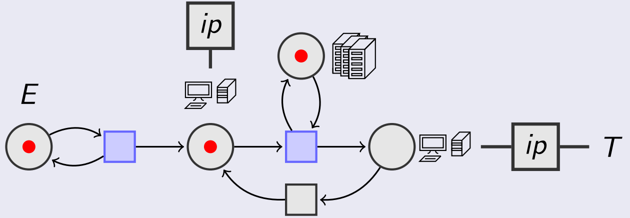
Idea of the Structural Semantics

Transitions model the evolution of patterns

In π-Calculus



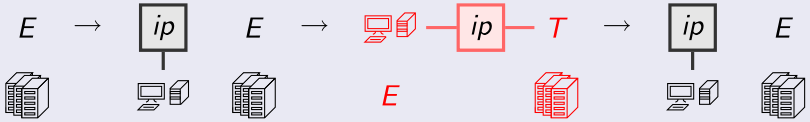
The Structural Semantics



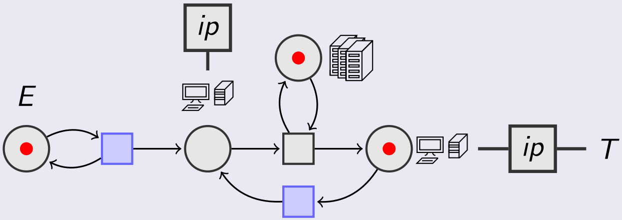
Idea of the Structural Semantics

Transitions model the evolution of patterns

In π-Calculus



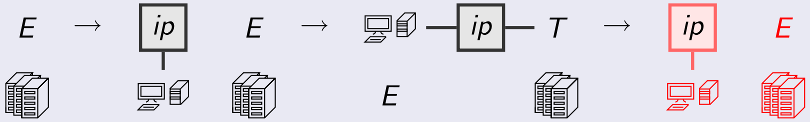
The Structural Semantics



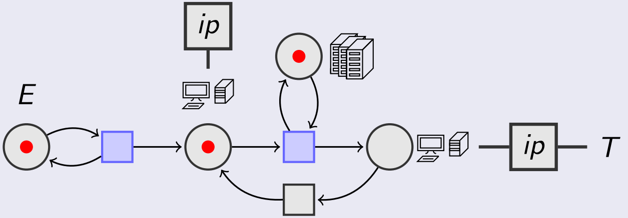
Idea of the Structural Semantics

Transitions model the evolution of patterns

In π-Calculus



The Structural Semantics



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)

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

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)

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

Restricted Form of Processes

Fragments

Topmost parallel components are called fragments

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

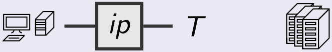
Restricted Form of Processes

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



Properties of the Semantics

Theorem (Full Retrievability)

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

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:

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

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:

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

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)

Structural semantics $\mathcal{N}[\![P]\!]$ is finite if and only if process P is structurally stationary.

A First Characterisation of Structural Stationarity

Structural Stationarity is Hard to Prove

Is there a characterisation?

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.,

$$\exists k \in \mathbb{N} : \forall Q \in Reach(P) : \forall F \in fg\,(rf\,(Q)) : \|F\|_{\mathcal{S}} \leq k.$$

Applications of the Characterisation

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:

$$\exists k \in \mathbb{N} : \forall Q \in Reach(P) : \|Q\|_{\mathcal{S}} \leq k.$$

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:

$$\exists k \in \mathbb{N} : \forall Q \in Reach(P) : \|Q\|_{\mathcal{S}} \leq k.$$

Corollary (Finitary Processes are Structurally Stationary)

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

$$\|F\|_{\mathcal{S}} \leq \|rf(Q)\|_{\mathcal{S}} = \|Q\|_{\mathcal{S}} \leq k.$$

A Second Characterisation of Structural Stationarity

Understanding Structural Stationarity

- Which processes are not structurally stationary?
- Is there a characterisation in terms of ν?

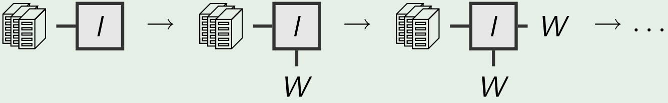
A Second Characterisation of Structural Stationarity

Understanding Structural Stationarity

- Which processes are not structurally stationary?
- Is there a characterisation in terms of ν?

Example

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

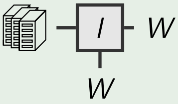


Breadth of Processes

Breadth

Maximal number of sequential processes sharing a restricted name

Example

For $F :=$ 

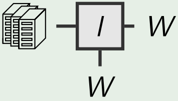
we have $\|F\|_{\mathcal{B}} = 3$.

Breadth of Processes

Breadth

Maximal number of sequential processes sharing a restricted name

Example

For $F :=$ 

we have $\|F\|_{\mathcal{B}} = 3$.

Problem

Boundedness in breadth does not ensure structural stationarity

Depth of Processes

Example

Lists are bounded in breadth but not structurally stationary:

$$LE \;\rightarrow\; LI - \boxed{id_1} - LE \;\rightarrow\; LI - \boxed{id_1} - LI - \boxed{id_2} - LE \;\rightarrow\; \ldots$$

Depth of Processes

Example

Lists are bounded in breadth but not structurally stationary:

$$LE \;\rightarrow\; LI - \boxed{id_1} - LE \;\rightarrow\; LI - \boxed{id_1} - LI - \boxed{id_2} - LE \;\rightarrow\; \ldots$$

Depth

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

Example

For $F := LI - \boxed{id_1} - LI - \boxed{id_2} - LE$ we have $\|F\|_{\mathcal{D}} = 2$.

Characterisation of Structural Stationarity via ν

Theorem

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.

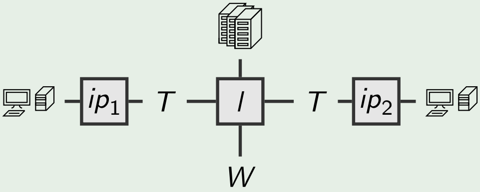
Decidability in Bounded Depth

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



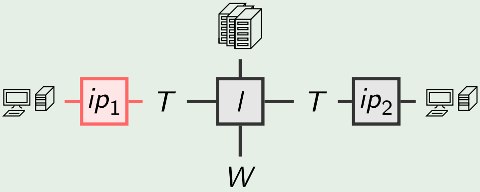
Decidability in Bounded Depth

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



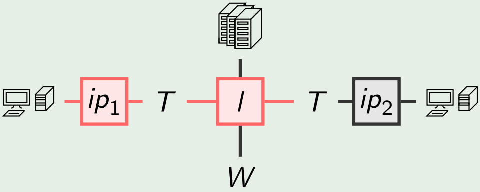
Decidability in Bounded Depth

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



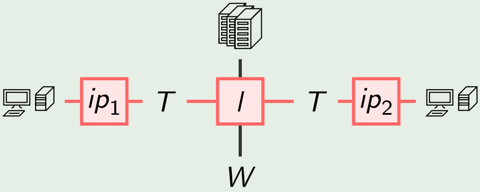
Decidability in Bounded Depth

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



Well-Structured Transition Systems

WSTS [Fin90, FS01, AČJT00]

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

Well-Structured Transition Systems

WSTS [Fin90, FS01, AČJT00]

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

Technically: WSTS $= (S, \to, \leq)$

- $(S, \to)$ 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 decidablity 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

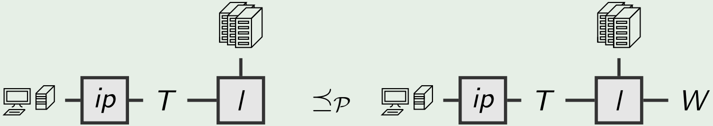
$$s_0 \longrightarrow s_1 \longrightarrow \ldots \longrightarrow \overbrace{s_i \longrightarrow \ldots \longrightarrow s_j}^{\leq} \longrightarrow \ldots$$

Instantiation of the Framework—The Ordering $\preceq_{\mathcal{P}}$

Intuition

Use hypergraph embedding as ordering

Example

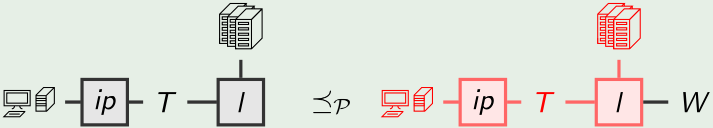


Instantiation of the Framework—The Ordering $\preceq_{\mathcal{P}}$

Intuition

Use hypergraph embedding as ordering

Example

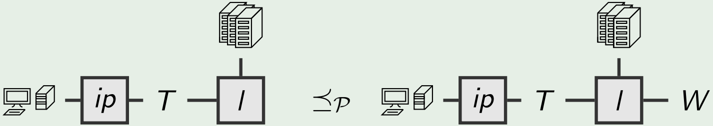


Instantiation of the Framework—The Ordering $\preceq_{\mathcal{P}}$

Intuition

Use hypergraph embedding as ordering

Example



Technically

Fragments may be added

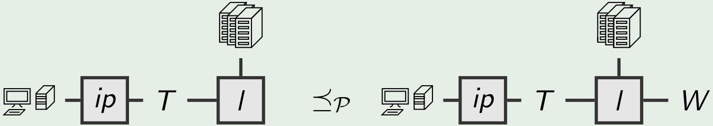
$$\nu a.(F \mid G) \preceq_{\mathcal{P}} \nu a.(F \mid G \mid H)$$

Instantiation of the Framework—The Ordering $\preceq_{\mathcal{P}}$

Intuition

Use hypergraph embedding as ordering

Example



Technically

Fragments may be added

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

Instantiation Theorem

Theorem

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

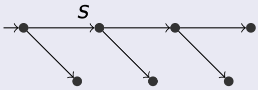
▸ WQO Proof

▸ Undecidability of Reachability

Decidability Results for WSTS [Fin90, FS01, AČJT00]

Finite Reachability Tree

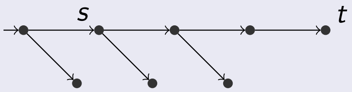
- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by $+$



Decidability Results for WSTS [Fin90, FS01, AČJT00]

Finite Reachability Tree

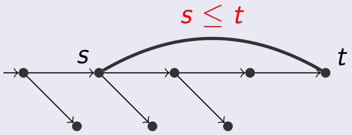
- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by $+$



Decidability Results for WSTS [Fin90, FS01, AČJT00]

Finite Reachability Tree

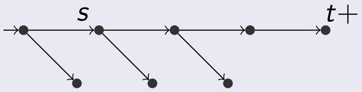
- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by $+$



Decidability Results for WSTS [Fin90, FS01, AČJT00]

Finite Reachability Tree

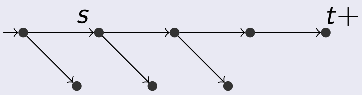
- Build computation tree (finite branching)
- If a new node covers predecessor stop and mark node by $+$



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 $+$



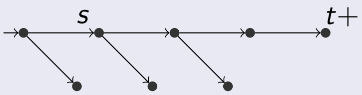
Theorem ([Fin90, FS01, AČJT00])

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

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 $+$

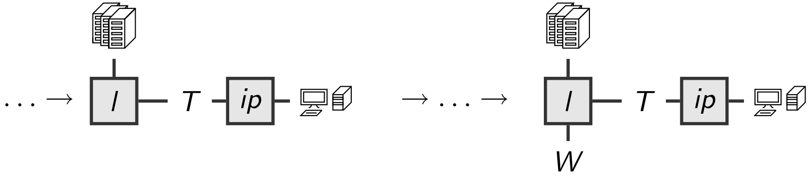


Theorem ([Fin90, FS01, AČJT00])

Non-terminating computation exists iff tree contains + node.
Infinite state iff + node is strictly larger.

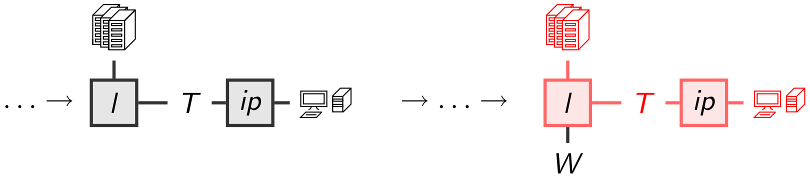
Application to the Client-Server System

Build the computation tree



Application to the Client-Server System

Build the computation tree

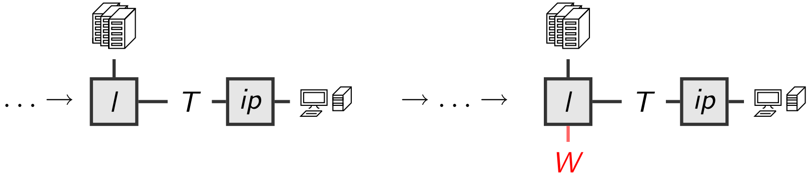


Results

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

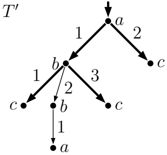
MWB [VM94, Dam96], HAL [FGMP03], SLMC [Cai04]

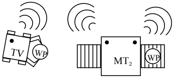
Normal Forms

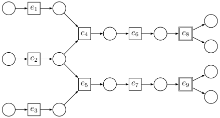
Decidability of structural congruence [EG99, EG04a, EG04b, EG07]

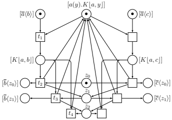
More Related Work

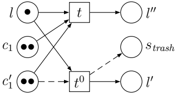
Survey

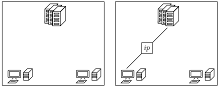

T'
a
1
2
b
c
1
2
3
c
b
c
1
a

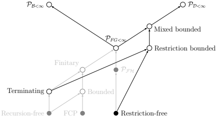

TV
WP
MT_2
WP

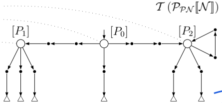

e_1
e_2
e_3
e_4
e_5
e_6
e_7
e_8
e_9


$[a(y).K\lfloor a,y\rfloor]$
$[\overline{a}\langle b\rangle]$
$[\overline{a}\langle c\rangle]$
t_1
$[K\lfloor a,b\rfloor]$
$[K\lfloor a,c\rfloor]$
t_2
t_3
t_4
$[\overline{b}\langle z_0\rangle]$
$[\overline{b}\langle z_1\rangle]$
$[\overline{c}\langle z_0\rangle]$
$[\overline{c}\langle z_1\rangle]$


l
t
l''
c_1
s_{trash}
c_1'
t^0
l'


ip


$\mathcal{P}_{B<\infty}$
$\mathcal{P}_{D<\infty}$
$\mathcal{P}_{FG<\infty}$
Mixed bounded
Restriction bounded
Finitary
$\mathcal{P}_{\mathcal{FH}}$
Terminating
Bounded
Recursion-free
FCP
Restriction-free


$\mathcal{T}(\mathcal{P}_{\mathcal{PN}}[\![\mathcal{N}]\!])$
$[P_1]$
$[P_0]$
$[P_2]$

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.

J. Bauer.
Analysis of Communication Topologies by Partner Abstraction.
PhD thesis, Department of Computer Science, Saarland University, 2006.

N. Busi and R. Gorrieri.
Distributed semantics for the π-calculus based on Petri nets with inhibitor arcs.
To appear in the Journal of Logic and Algebraic Programming, 46 pages, August 2008.

N. Busi, M. Gabbrielli, and G. Zavattaro.
Replication vs. recursive definitions in channel based calculi.
In *Proc. of the 30th International Colloquium on Automata, Languages and Programming, ICALP 2003*, volume 2719 of *LNCS*, pages 133–144. Springer-Verlag, 2003.

References II

N. Busi, M. Gabbrielli, and G. Zavattaro.
Comparing recursion, replication, and iteration in process calculi.
In *Proc. of the 31th International Colloquium on Automata, Languages and Programming, ICALP 2004*, volume 3142 of *LNCS*, pages 307–319. Springer-Verlag, 2004.

N. Busi, M. Gabbrielli, and G. Zavattaro.
On the expressive power of recursion, replication, and iteration in process calculi.
27 pages, under consideration for publication in Mathematical Structures of Computer Science, 2008.

J. Bauer, T. Toben, and B. Westphal.
Mind the shapes: Abstraction refinement via topology invariants.
In *Proc. of the 5th International Symposium on Automated Technology for Verification and Analysis, ATVA 2007*, volume 4762 of *LNCS*, pages 35–50. Springer-Verlag, 2007.

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/`, last access 2008-11-28.

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

References III

C. Dufourd, A. Finkel, and Ph. Schnoebelen.
Reset nets between decidability and undecidability.
In *Proc. of the 25th International Colloquium on Automata, Languages and Programming, ICALP 1998*, volume 1443 of *LNCS*, pages 103–115. Springer-Verlag, 1998.

R. Demangeon, D. Hirschkoff, and D. Sangiorgi.
Static and dynamic typing for the termination of mobile processes.
In *Proc. of the 5th IFIP International Conference on Theoretical Computer Science, IFIP TCS 2008*, volume 273 of *IFIP*, pages 413–427. Springer-Verlag, 2008.

R. Devillers, H. Klaudel, and M. Koutny.
A Petri net translation of π-Calculus terms.
In *Proc. of the 3rd International Colloquium on Theoretical Aspects of Computing, ICTAC 2006*, volume 4281 of *LNCS*, pages 138–152. Springer-Verlag, 2006.

G. Delzanno, J.-F. Raskin, and L. Van Begin.
Towards the automated verification of multithreaded java programs.
In *Proc. of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2002*, volume 2280 of *LNCS*, pages 173–187. Springer-Verlag, 2002.

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

References IV

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.
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 Informatica, 40(6):385–430, 2004.

J. Engelfriet and T. Gelsema.
An exercise in structural congruence.
Information Processing Letters, 101(1):1–5, 2007.

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

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/`, last access 2008-11-28.

References V

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

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.

B. König and V. Kozioura.
Counterexample-guided abstraction refinement for the analysis of graph transformation systems.
In *Proc. of the 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2006*, volume 3920 of *LNCS*, pages 197–211. Springer-Verlag, 2006.

V. Khomenko, M. Koutny, and A. Niaouris.
Applying Petri net unfoldings for verification of mobile systems.
In *Proc. of the 4th Workshop on Modelling of Objects, Components and Agents, MOCA 2006*, Bericht FBI-HH-B-267/06, pages 161–178. University of Hamburg, 2006.

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

References VI

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.

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-Verlag, 1995.

U. Montanari and M. Pistore.
Concurrent semantics for the π-calculus.
Electronic Notes in Theoretical Computer Science, 1:411–429, 1995.

U. Montanari and M. Pistore.
History dependent automata.
Technical report, Instituto Trentino di Cultura, 2001.

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

References VII

F. Orava and J. Parrow.
An algebraic verification of a mobile network.
Formal Aspects of Computing, 4(6):497–543, 1992.

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

J.-F. Raskin and L. Van Begin.
Petri nets with non-blocking arcs are difficult to analyze.
Electronic Notes in Theoretical Computer Science, 98:35–55, 2004.

A. Rensink.
Canonical graph shapes.
In *Proc. of the 13th European Symposium on Programming, ESOP 2004*, volume 2986 of *LNCS*, pages 401–415. Springer-Verlag, 2004.

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

T. Toben.
Analysis of Dynamic Evolution Systems by Spotlight Abstraction Refinement.
PhD thesis, Department of Computing Science, University of Oldenburg, 2008.

References VIII

B. Victor and F. Moller.
The mobility workbench: A tool for the π-calculus.
In *Proc. of the 6th International Conference on Computer Aided Verification*, volume 818 of *LNCS*, pages 428–440. Springer-Verlag, 1994.
MWB: `http://www.it.uu.se/research/group/mobility/mwb`, last access 2008-11-28.

B. Westphal.
Specification and Verification of Dynamic Topology Systems.
PhD thesis, Department of Computing Science, University of Oldenburg, 2008.

B. Wachter and B. Westphal.
The spotlight principle. on combining process-summarising state abstractions.
In *Proc. of the 8th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2007*, volume 4349 of *LNCS*, pages 182–198. Springer-Verlag, 2007.

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

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]

Return

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]

Return

Why is $\preceq_{\mathcal{P}}$ a WQO?

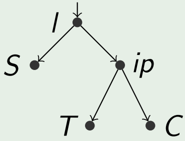
Proof Idea

Understand fragments as (syntax) trees

- Sequential processes are leafs
- Restricted names are nodes

Example

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



Return

Why is $\preceq_{\mathcal{P}}$ a WQO?

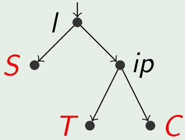
Proof Idea

Understand fragments as (syntax) trees

- Sequential processes are leafs
- Restricted names are nodes

Example

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



Return

Why is $\preceq_{\mathcal{P}}$ a WQO?

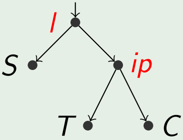
Proof Idea

Understand fragments as (syntax) trees

- Sequential processes are leafs
- Restricted names are nodes

Example

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



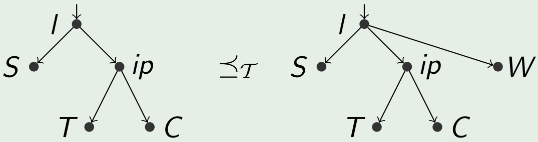
Return

Why is $\preceq_{\mathcal{P}}$ a WQO?

Proof Idea

Use a suitable wqo on trees

Example



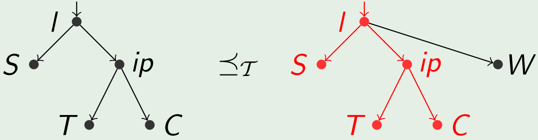
Return

Why is $\preceq_{\mathcal{P}}$ a WQO?

Proof Idea

Use a suitable wqo on trees

Example



Return

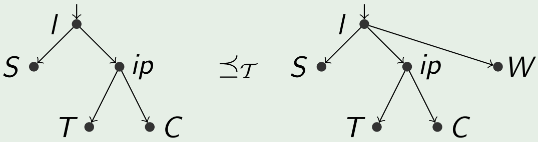
Why is $\preceq_{\mathcal{P}}$ a WQO?

Proof Idea

Use a suitable wqo on trees

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

Example

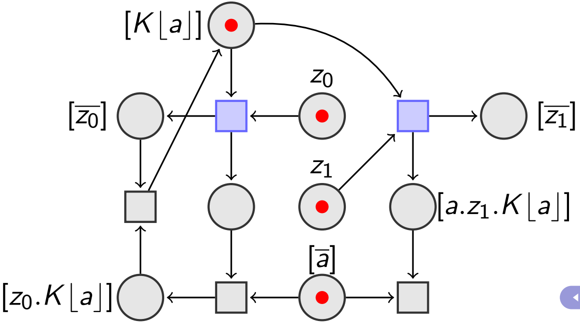


◂ Return

Why Order Name Places?

Counterexample

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

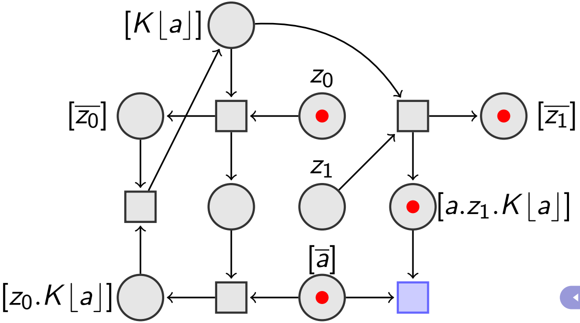


Return

Why Order Name Places?

Counterexample

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

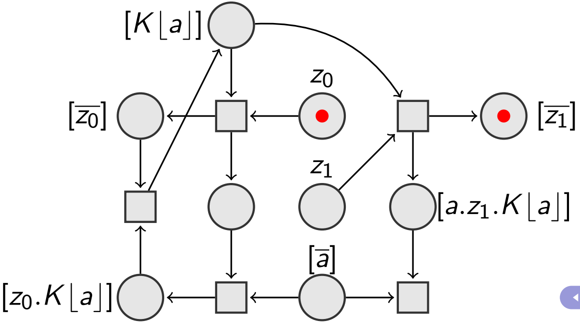


Return

Why Order Name Places?

Counterexample

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

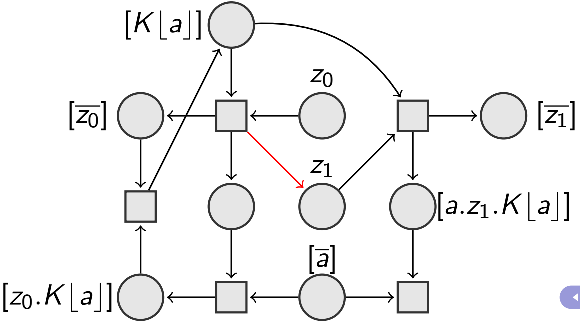


Return

Why Order Name Places?

Counterexample

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

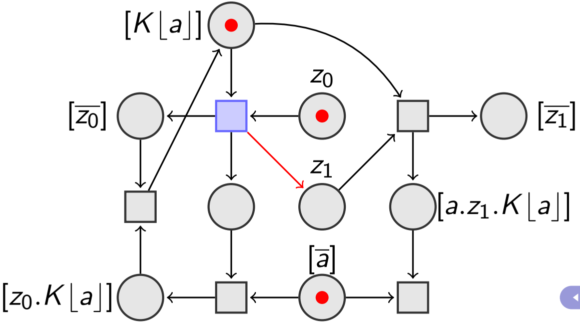


Return

Why Order Name Places?

Counterexample

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

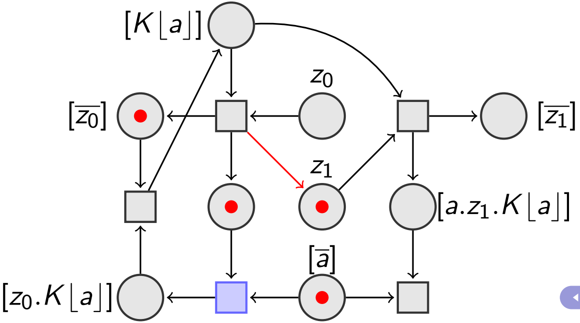


Return

Why Order Name Places?

Counterexample

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

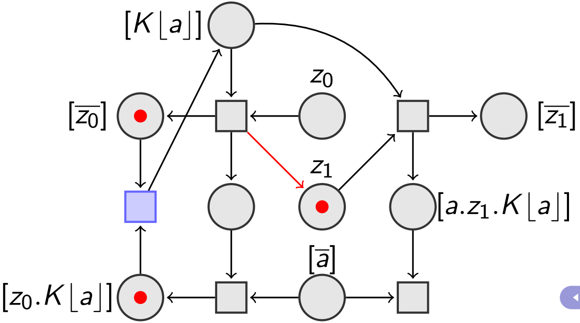


Return

Why Order Name Places?

Counterexample

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

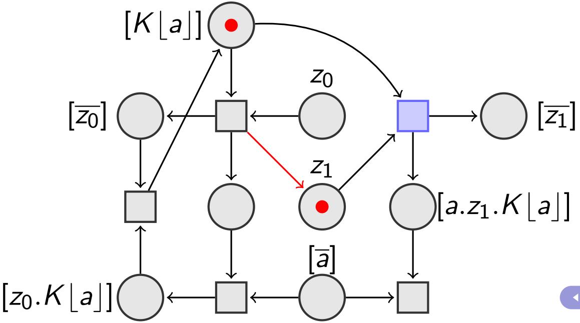


Return

Why Order Name Places?

Counterexample

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

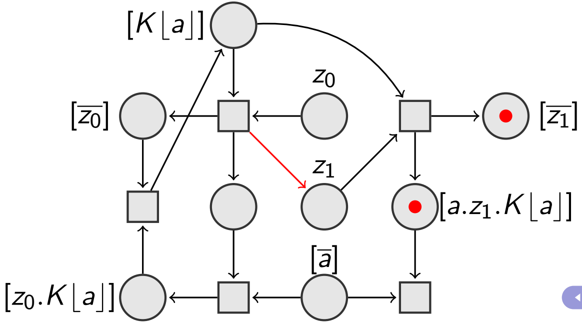


Return

Why Order Name Places?

Counterexample

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



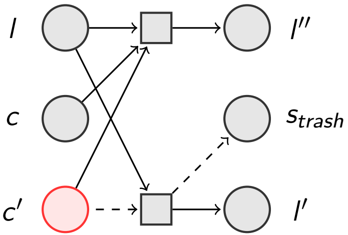
Return

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



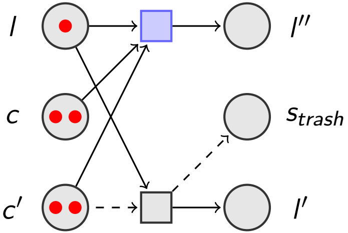
Return

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



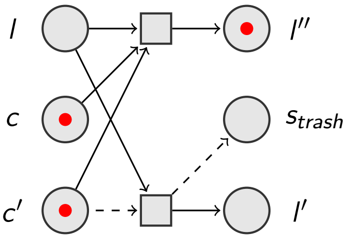
Return

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



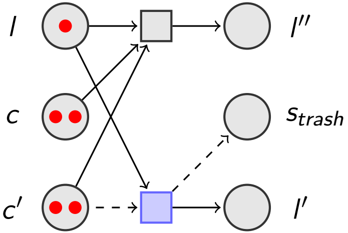
Return

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'



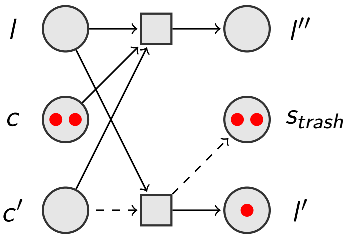
Return

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'



Return

Undecidability of Reachability in Depth One

Process Bunch

Modify an arbitrary number of processes with one communication

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

Example

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

◂ Return

Undecidability of Reachability in Depth One

Process Bunch

Modify an arbitrary number of 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

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

Return

Undecidability of Reachability in Depth One

Process Bunch

Modify an arbitrary number of 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

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

Return

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)

Return

Unfolding-based Verification

Model	[KKN06] unf	[KKN06] sat	MWB dl	HAL π2fc	**Struct** unf	**Struct** sat
ns2	1	< 1	< 1	< 1	< 1	< 1
ns3	7	< 1	1	8	< 1	< 1
ns4	69	1	577	382	< 1	< 1
ns5	532	58	—	—	17	3
ns6	—	—	—	—	1518	84
gsm [OP92]	n/a		—	18	< 1	< 1

Verified car platoon and autonomous transport

Instance	**Struct** $\lvert P\rvert$	**Struct** $\lvert T\rvert$	**Model Checking** unf	**Model Checking** $\lvert B\rvert$	**Model Checking** $\lvert E\rvert$	**Model Checking** sat
1p 6m 4v ref	937	1371	8h	923236	721991	20min

Return

Size of the Translation

Idea

$$\mathcal{N}_1, \quad \mathcal{N}_2, \quad \mathcal{N}_3, \ldots$$

Return

Size of the Translation

Idea

$$\begin{array}{c} \mathcal{N}_1, \quad \mathcal{N}_2, \quad \mathcal{N}_3, \ldots \\ \Big\downarrow \text{linear} \\ \mathcal{P}[\![\mathcal{N}_1]\!], \quad \mathcal{P}[\![\mathcal{N}_2]\!], \quad \mathcal{P}[\![\mathcal{N}_3]\!], \ldots \end{array}$$

Return

Size of the Translation

Idea

$$
\begin{array}{c}
\mathcal{N}_1, \quad \mathcal{N}_2, \quad \mathcal{N}_3, \ldots \\
\Big\downarrow \text{linear} \\
\mathcal{P}[\![\mathcal{N}_1]\!], \quad \mathcal{P}[\![\mathcal{N}_2]\!], \quad \mathcal{P}[\![\mathcal{N}_3]\!], \ldots \\
\Big\downarrow \mathcal{N}_i \text{ state yields } \mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_i]\!]]\!] \text{ place} \\
\mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_1]\!]]\!], \quad \mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_2]\!]]\!], \quad \mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_3]\!]]\!], \ldots
\end{array}
$$

Return

Size of the Translation

Idea

$$
\begin{array}{c}
\mathcal{N}_1, \quad \mathcal{N}_2, \quad \mathcal{N}_3, \ldots \\
\Big\downarrow \text{linear} \\
\mathcal{P}[\![\mathcal{N}_1]\!], \quad \mathcal{P}[\![\mathcal{N}_2]\!], \quad \mathcal{P}[\![\mathcal{N}_3]\!], \ldots \\
\Big\downarrow \mathcal{N}_i \text{ state yields } \mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_i]\!]]\!] \text{ place} \\
\mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_1]\!]]\!], \quad \mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_2]\!]]\!], \quad \mathcal{N}[\![\mathcal{P}[\![\mathcal{N}_3]\!]]\!], \ldots
\end{array}
$$

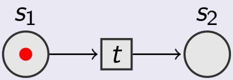
Theorem (Size of the Structural Semantics)

The size of the structural semantics $\mathcal{N}[\![P]\!]$ is not bounded by a primitive recursive function in the size of the process P.

Return

Size of the Translation

Technically



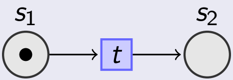
$$\nu act.(\overline{act} \mid act.K_t\lfloor act, s_1, s_2\rfloor \mid \overline{s_1}\langle act\rangle)$$

$$K_t(act, s_1, s_2) := s_1(x).(\overline{act} \mid act.K_t\lfloor act, s_1, s_2\rfloor \mid \overline{s_2}\langle act\rangle)$$

Return

Size of the Translation

Technically

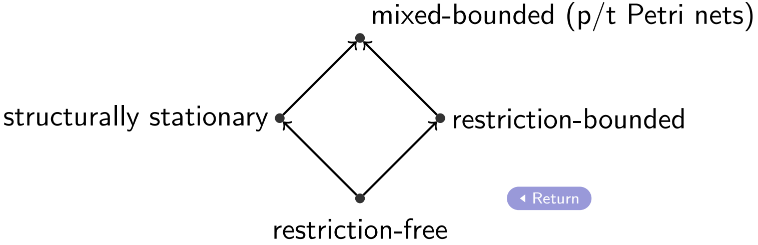


$$\nu act.(\overline{act} \mid act.\,K_t\lfloor act, s_1, s_2\rfloor \mid \overline{s_1}\langle act\rangle)$$

$$K_t(act, s_1, s_2) := s_1(x).(\overline{act} \mid act.K_t\lfloor act, s_1, s_2\rfloor \mid \overline{s_2}\langle act\rangle)$$

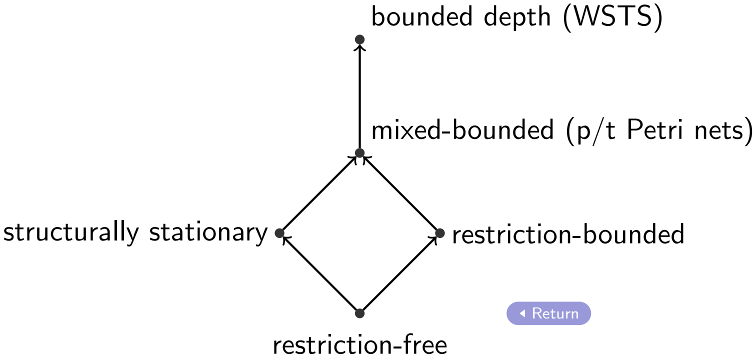
Return

Hierarchy of Processes



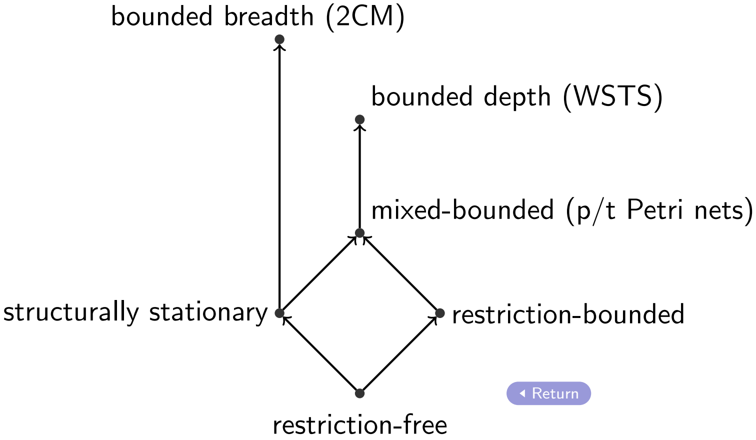
Return

Hierarchy of Processes



Return

Hierarchy of Processes



Return

Finite Handler Processes—Participants

- Register at handler processes via distinguished public channels

Example

- Channel *cfa* is a distinguished name
- A free agent is a participant:

$$\nu id, ca, rq.\, \overline{cfa}\langle id\rangle\, .\overline{id}\langle ca\rangle.\overline{id}\langle rq\rangle \ldots$$

◂ Return

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:

$$\nu id, ca, rq.\overline{cfa}\langle id\rangle.\overline{id}\langle ca\rangle.\overline{id}\langle rq\rangle \ldots$$

Return

Finite Handler Processes—Handler

- Listen on the distinguished channels

Example

The *MRG* process is a handler

$$\boxed{cfa}(id_x).id_x(ca_x)\ldots\boxed{cfa}(id_y)\ldots id_y(rq_y).\overline{ca_x}\langle rq_y\rangle.MRG\lfloor cfa\rfloor$$

Return

Finite Handler Processes—Handler

- Listen on the distinguished channels
- Receive finitely many processes

Example

The *MRG* process is a handler

$$cfa(id_x).id_x(ca_x)\ldots cfa(id_y)\ldots id_y(rq_y).\overline{ca_x}\langle rq_y\rangle.MRG\lfloor cfa\rfloor$$

Return

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

$$cfa(id_x).id_x(ca_x)\ldots cfa(id_y)\ldots id_y(rq_y).\overline{ca_x}\langle rq_y\rangle.MRG\lfloor cfa\rfloor$$

◂ Return

Fundamental Property of Finite Handler Processes

Theorem

Finite handler processes are structurally stationary.

The car platooning example is a finite handler process

◂ Return