A Practical Approach to Verification of Mobile Systems Using Net Unfoldings

Roland Meyer, Victor Khomenko, Tim Strazny

University of Oldenburg

2007-17-06

A Client/Server System in the π-Calculus

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

Graphically

$$C \xrightarrow{(IP)} S$$

In π-Calculus

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

A Client/Server System in the π-Calculus

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

Graphically

$$C \xrightarrow{(IP)} S$$

In π-Calculus

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

A Client/Server System in the π-Calculus

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

Graphically

$$C \xrightarrow{(IP)} S$$

In π-Calculus

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

$$url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url\rfloor$$

A Client/Server System in the π-Calculus

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

Graphically

$$C \xrightarrow{\text{(IP)}} S$$

In π-Calculus

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

A Client/Server System in the π-Calculus

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

Graphically

$$C \xrightarrow{(IP)} S$$

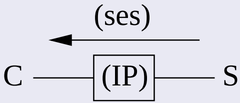
In π-Calculus

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

A Client/Server System in the π-Calculus

Server sends back a private session *ses* on the private channel *ip*

Graphically



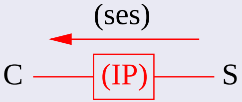
In π-Calculus

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

A Client/Server System in the π-Calculus

Server sends back a private session *ses* on the private channel *ip*

Graphically



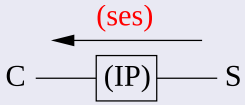
In π-Calculus

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

A Client/Server System in the π-Calculus

Server sends back a private session *ses* on the private channel *ip*

Graphically



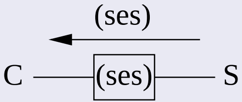
In π-Calculus

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

A Client/Server System in the π-Calculus

To terminate the session, the server sends the private session object *ses* on the channel *ses* itself

Graphically



In π-Calculus

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

A Client/Server System in the π-Calculus

The client is ready to contact the server again on the public channel *url*

Graphically

C S

In π-Calculus

$$C\lfloor url\rfloor \mid S\lfloor url\rfloor$$

A Client/Server System in the π-Calculus

The client is ready to contact the server again on the public channel *url*

Graphically

C S

In π-Calculus

$$C\lfloor url \rfloor \mid S\lfloor url \rfloor$$

A Client/Server System in the π-Calculus

The client is ready to contact the server again on the public channel *url*

Graphically

C S

In π-Calculus

$$C\lfloor url \rfloor \mid S\lfloor url \rfloor$$

A Client/Server System in the π-Calculus

The client is ready to contact the server again on the public channel *url*

Graphically

C S

In π-Calculus

$$C\lfloor url \rfloor \mid S\lfloor url \rfloor$$

Overview

1. A client/server system in the π-Calculus
2. From π-Calculus to Petri nets
3. A Boundedness Result for FCP nets
4. From FCPs to Safe Nets
5. Model Checking with Net Unfoldings

Verficiation of Dynamically Reconfigurable Systems

- Motivation: automatic verification of dynamically reconfigurable systems
- Approach: employ a mapping of π-Calculus processes into Petri nets that preserves the reaction relation
  - The π-Calculus is well suited for modelling dynamically reconfigurable systems
  - Petri nets come with well investigated verification techniques

From π-Calculus to Petri nets

- Observation: a process is represented by several graphs
- Idea: represent this connection structure in a Petri net:
  - Every place represents a possible graph
  - Every token represents an occurence of the graph

From π-Calculus to Petri nets

Example

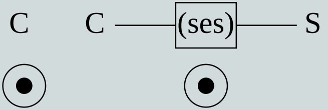
A client and a client who has a session with a server:

In the π-Calculus

C   C —— $\boxed{\text{(ses)}}$ —— S

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

As a Petri net



From π-Calculus to Petri nets

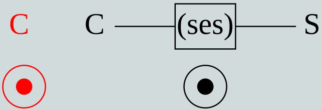
Example

A client and a client who has a session with a server:

In the π-Calculus

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

As a Petri net



From π-Calculus to Petri nets

Example

A client and a client who has a session with a server:

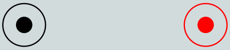
In the π-Calculus

C   C — $\boxed{\text{(ses)}}$ — S

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

As a Petri net

C   C — $\boxed{\text{(ses)}}$ — S



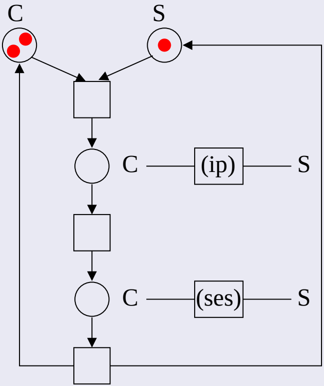
P and $\mathcal{PN}[\![P]\!]$ are isomorphic

In the π-Calculus

↓

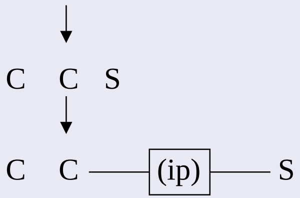
C C S

As a Petri net

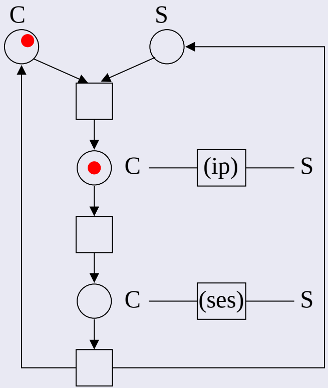


P and $\mathcal{PN}[\![P]\!]$ are isomorphic

In the π-Calculus

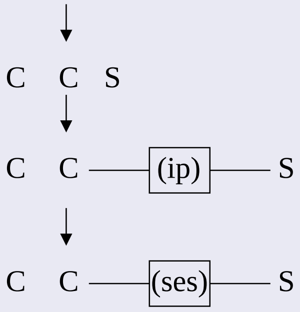


As a Petri net

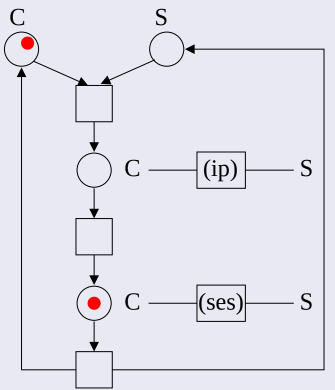


P and $\mathcal{PN}[\![P]\!]$ are isomorphic

In the π-Calculus

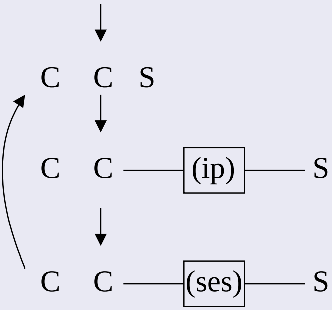


As a Petri net

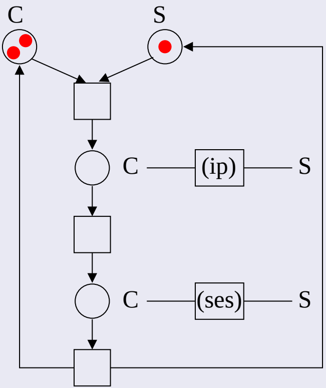


P and $\mathcal{PN}[\![P]\!]$ are isomorphic

In the π-Calculus



As a Petri net



Finite Control Processes

Problem

- Petri net translation may be an infinite or unbounded Petri net
- For model checking we need small bounds $\Rightarrow$ safe nets!

Solution

- Restrict ourselves to an expressive fragment of π-Calculus
- In Finite Control Processes (FCPs) [Dam96], no processes are created, i.e.,

$$P_{\mathcal{FC}} = P_1 \mid \ldots \mid P_n,$$

where the P_i do not use parallel composition $\mid$

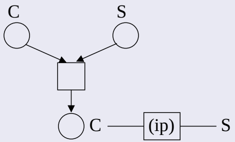
Finite Control Processes

Example (The Client/Server System)

$$C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor$$

Observation

- Initially, each place in the net $\mathcal{PN}[\![P_1 \mid \ldots \mid P_\mathbf{n}]\!]$ carries at most **n** tokens
- Transitions split or join processes



- So, the net $\mathcal{PN}[\![P_1 \mid \ldots \mid P_\mathbf{n}]\!]$ is trivially bounded by **n**
- Better bounds? Where are the safe nets?

Towards the Boundedness Result

Processes use recursive equations to define infinite behaviours

$$K(\tilde{x}) := P.$$

Example

Client and server are defined by:

$$C(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url\rfloor$$
$$S(url) := url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url\rfloor.$$

K, C, and S are process identifiers

Towards the Boundedness Result

Processes use recursive equations to define infinite behaviours

$$K(\tilde{x}) := P.$$

Example

Client and server are defined by:

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

$$S(url) := url(y).\nu ses.\overline{y}\langle ses \rangle.\overline{ses}\langle ses \rangle.S\lfloor url \rfloor.$$

K, C, and S are process identifiers

Towards the Boundedness Result

Process identifiers cannot be renamed:

Example

The following clients are distinct because $C^1 \neq C^2$:

$$\begin{aligned} & \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).\boxed{C^1}\lfloor url\rfloor \\ \neq & \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).\boxed{C^2}\lfloor url\rfloor \end{aligned}$$

Towards the Boundedness Result

Process identifiers cannot be renamed:

Example

The following clients are distinct because $C^1 \neq C^2$:

$$\nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).\boxed{C^1}\lfloor url\rfloor$$
$$\neq \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).\boxed{C^2}\lfloor url\rfloor$$

Observation

So **k** tokens on a place in $\mathcal{PN}[\![P_1 \mid \ldots \mid P_n]\!]$ imply

k processes $P_{i_1}, \ldots, P_{i_k}$ have common process identifiers

Orbits

The orbit of a process is the set of process identifiers a process uses

Example

Consider two clients and a server:

$$C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor, \text{ where}$$

$$C(url) := \nu ip.\overline{url}\langle ip \rangle.ip(s).s(x).C\lfloor url \rfloor$$
$$S(url) := url(y).\nu ses.\overline{y}\langle ses \rangle.\overline{ses}\langle ses \rangle.S\lfloor url \rfloor.$$

Orbits

Example

Consider two clients and a server:

$$C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor, \text{ where}$$

$$\begin{aligned} C(url) &:= \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url \rfloor \\ S(url) &:= url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url \rfloor. \end{aligned}$$

The orbits are:

$$orb(C\lfloor url \rfloor) = \{C\}$$

Orbits

Example

Consider two clients and a server:

$$C\lfloor url\rfloor \mid C\lfloor url\rfloor \mid S\lfloor url\rfloor, \text{ where}$$

$$C(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url\rfloor$$
$$S(url) := url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url\rfloor.$$

The orbits are:

$$orb(C\lfloor url\rfloor) = \{C\}$$
$$orb(C\lfloor url\rfloor) = \{C\}$$

Orbits

Example

Consider two clients and a server:

$$C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor, \text{ where}$$

$$C(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C\lfloor url \rfloor$$
$$S(url) := url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url \rfloor.$$

The orbits are:

$$orb(C\lfloor url \rfloor) = \{C\}$$
$$orb(C\lfloor url \rfloor) = \{C\}$$
$$orb(S\lfloor url \rfloor) = \{S\}.$$

The Boundedness Result

Observation Rephrased

If we have **k** tokens on a place in $\mathcal{PN}[\![P_1 \mid \ldots \mid P_n]\!]$ then there are

k intersecting orbits

$$orb(P_{i_1}) \cap \ldots \cap orb(P_{i_k}) \neq \emptyset$$

The Boundedness Result

Observation Rephrased

If we have **k** tokens on a place in $\mathcal{PN}[\![P_1 \mid \ldots \mid P_n]\!]$ then there are

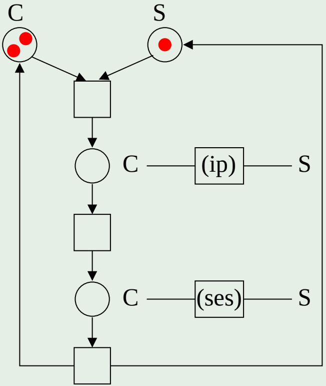
k intersecting orbits

$$orb(P_{i_1}) \cap \ldots \cap orb(P_{i_k}) \neq \emptyset$$

Theorem

The maximal number of intersecting orbits in $P_1 \mid \ldots \mid P_n$ gives a bound for $\mathcal{PN}[\![P_1 \mid \ldots \mid P_n]\!]$.

Example



is bounded by 2, since
$orb(C\lfloor url \rfloor) \cap orb(C\lfloor url \rfloor) \neq \emptyset$

Improves the trivial bound of 3

From FCPs to Safe Nets

Definition

An FCP $P_{\mathcal{FC}} = P_1 \mid \ldots \mid P_n$ is called a safe process, if all orbits are disjoint.

- The theorem guarantees that safe processes are mapped to safe Petri nets
- The function *Safe* maps every FCP $P_{\mathcal{FC}}$ to a safe process $\mathit{Safe}(P_{\mathcal{FC}})$

The Function *Safe*

- Idea: Rename all process identifiers used by P_i
- Duplicate the corresponding equations
- Remove the original equations

Example

$$P_{\mathcal{FC}} = C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor$$
$$C(url) := \nu ip.\overline{url}\langle ip \rangle.ip(s).s(x).C\lfloor url \rfloor$$
$$S(url) := url(y).\nu ses.\overline{y}\langle ses \rangle.\overline{ses}\langle ses \rangle.S\lfloor url \rfloor.$$

The Function *Safe*

Example

$$P_{\mathcal{FC}} = \boxed{C}\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor$$

$$\boxed{C}(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).\boxed{C}\lfloor url \rfloor$$

$$S(url) := url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url \rfloor.$$

$$Safe(P_{\mathcal{FC}}) = \boxed{C^1}\lfloor url \rfloor \mid C^2\lfloor url \rfloor \mid S^3\lfloor url \rfloor$$

$$\boxed{C^1}(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).\boxed{C^1}\lfloor url \rfloor$$

The Function *Safe*

Example

$$P_{\mathcal{FC}} = C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor$$

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

$$S(url) := url(y).\nu ses.\overline{y}\langle ses\rangle.\overline{ses}\langle ses\rangle.S\lfloor url \rfloor.$$

$$Safe(P_{\mathcal{FC}}) = C^1\lfloor url \rfloor \mid C^2\lfloor url \rfloor \mid S^3\lfloor url \rfloor$$

$$C^1(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C^1\lfloor url \rfloor$$

$$C^2(url) := \nu ip.\overline{url}\langle ip\rangle.ip(s).s(x).C^2\lfloor url \rfloor$$

The Function *Safe*

Example

$$
\begin{aligned}
P_{\mathcal{FC}} &= C\lfloor url \rfloor \mid C\lfloor url \rfloor \mid S\lfloor url \rfloor \\
C(url) &:= \nu ip.\overline{url}\langle ip \rangle.ip(s).s(x).C\lfloor url \rfloor \\
S(url) &:= url(y).\nu ses.\overline{y}\langle ses \rangle.\overline{ses}\langle ses \rangle.S\lfloor url \rfloor.
\end{aligned}
$$

$$
\begin{aligned}
Safe(P_{\mathcal{FC}}) &= C^1\lfloor url \rfloor \mid C^2\lfloor url \rfloor \mid S^3\lfloor url \rfloor \\
C^1(url) &:= \nu ip.\overline{url}\langle ip \rangle.ip(s).s(x).C^1\lfloor url \rfloor \\
C^2(url) &:= \nu ip.\overline{url}\langle ip \rangle.ip(s).s(x).C^2\lfloor url \rfloor \\
S^3(url) &:= url(y).\nu ses.\overline{y}\langle ses \rangle.\overline{ses}\langle ses \rangle.S^3\lfloor url \rfloor.
\end{aligned}
$$

Bisimilarity

Theorem

$P_{\mathcal{FC}}$ and $Safe(P_{\mathcal{FC}})$ are bisimilar, just remove the superscripts

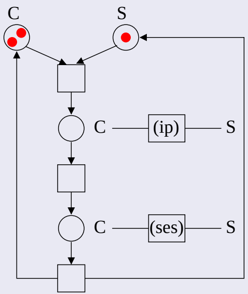
P and $\mathcal{PN}[\![P]\!]$ are isomorphic

Corollary

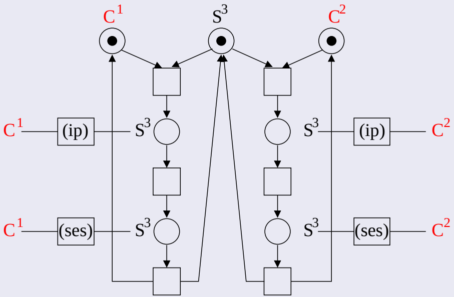
$P_{\mathcal{FC}}$ and $\mathcal{PN}[\![Safe(P_{\mathcal{FC}})]\!]$ are bisimilar. The reachable processes can be reconstructed from the markings.

Bisimilarity

$\mathcal{PN}[\![P_{\mathcal{FC}}]\!]$

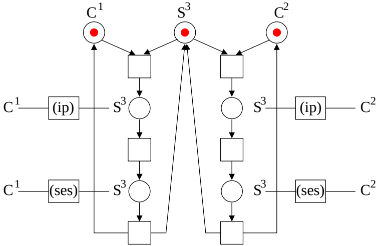


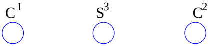
$\mathcal{PN}[\![Safe(P_{\mathcal{FC}})]\!]$



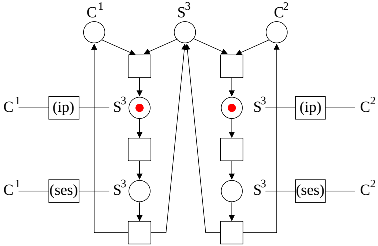
- Unfoldings represent the reachable states of a Petri net implicitly
- Technically: acyclic net that records the causal dependence of transitions
- Intuitively: just fire the transitions and record the places they use

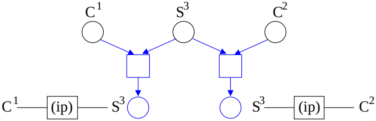
The Finite and Complete Prefix



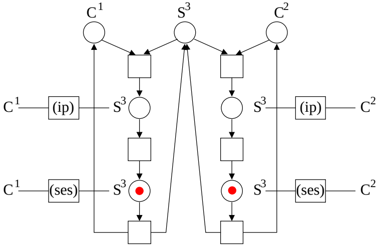


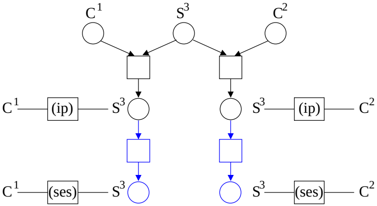
The Finite and Complete Prefix



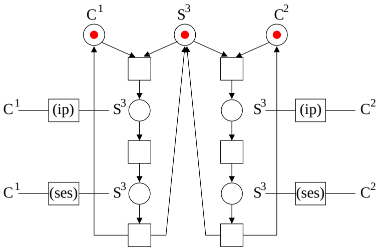


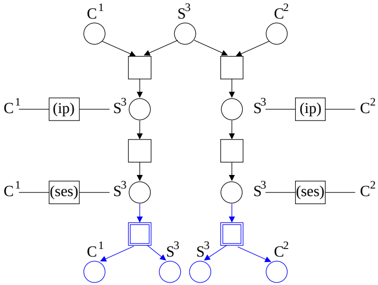
The Finite and Complete Prefix





The Finite and Complete Prefix

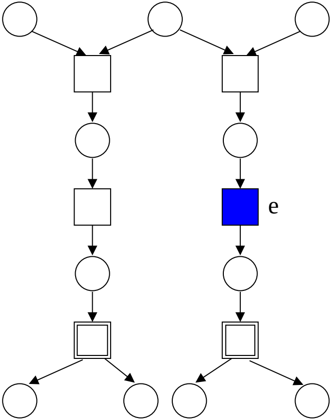




Model Checking Unfoldings with SAT

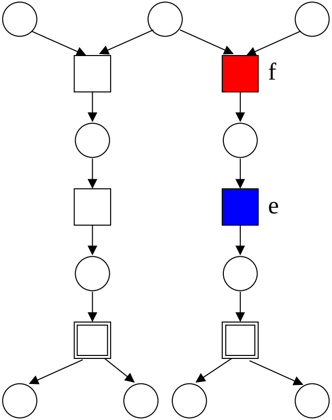
- A Boolean expression φ is built from the prefix so that
  - φ is unsatisfiable iff the property holds
  - Every satisfying assignment gives a violation trace
- φ has the form **Conf** $\wedge$ **Viol**

Conf: Causality



If *e* is executed, then its causal predecessors are executed

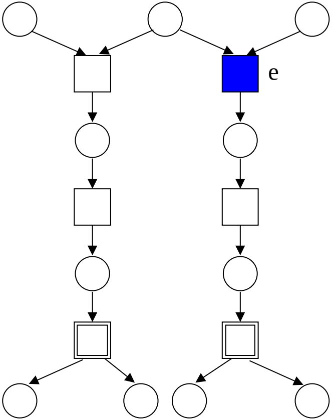
Conf: Causality



If e is executed, then its causal predecessors are executed

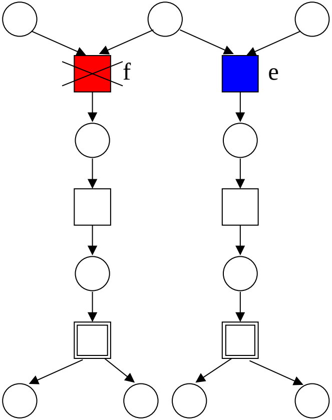
$$\bigwedge_{e \in E \setminus E_{Cut}} \bigwedge_{f \in {}^{\bullet\bullet}e} e \Rightarrow f$$

Conf: Conflicts



If *e* is executed, then events in conflict are not executed

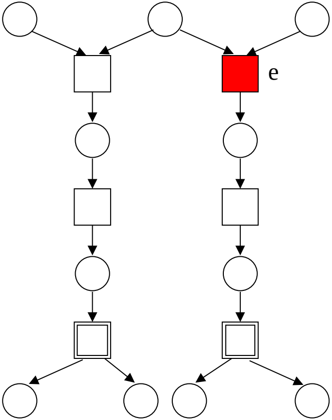
Conf: Conflicts



If e is executed, then events in conflict are not executed

$$\bigwedge_{e \in E \setminus E_{Cut}} \bigwedge_{f \in ({}^{\bullet}e)^{\bullet} \setminus \{e\}} e \Rightarrow \neg f$$

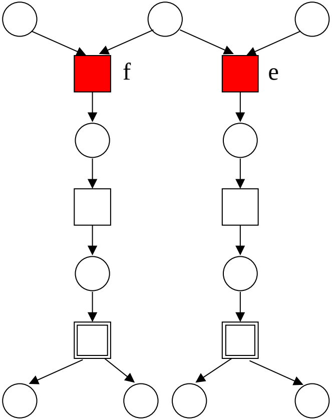
Viol: Deadlock



If *e* cannot be executed,

- either some event in conflict or *e* itself has been executed

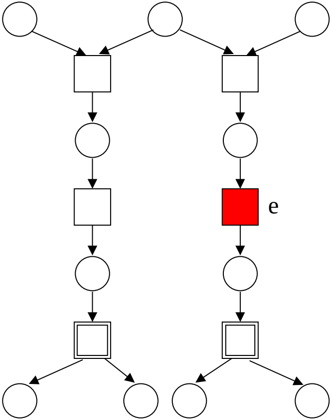
Viol: Deadlock



If *e* cannot be executed,

- either some event in conflict or *e* itself has been executed

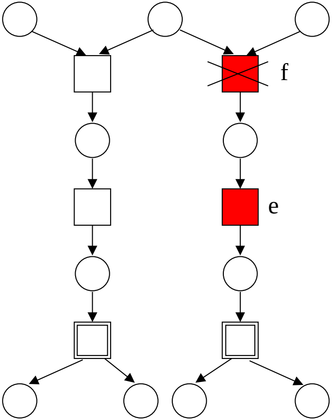
Viol: Deadlock



If *e* cannot be executed,

- either some event in conflict or *e* itself has been executed
- or the predecessor of *e* has not fired

Viol: Deadlock



If e cannot be executed,

- either some event in conflict or e itself has been executed
- or some predecessor of e has not fired

$$\bigwedge_{e \in E} \left(\bigvee_{f \in ({}^{\bullet}e)^{\bullet} \setminus E_{Cut}} f \vee \bigvee_{f \in {}^{\bullet\bullet}e} \neg f \right)$$

Experimental Results

- Comparison with *MWB* [VM94], *HAL* [FGMP03], and [DKK06] translation+Unfolding-based model checking
- Three case studies:
  - Client/Server with scalable number of clients and sessions
  - Education system from [KKN06], high-degree of concurrency, scalable in the number of students
  - Realistic model of GSM network [OP92]
- Property: deadlock-freeness

Experimental Results

- *MWB* and *HAL* do not scale well
- [KKN06]+Unfoldings scales but requires recursion-free processes
- Handled nets of up to 282 places, 1722 transitions
- Building the unfolding took up to 25 minutes, SAT solving 84 seconds
- Others failed when our nets had still 100 places and 300 transitions

Experimental Results

- In absolute numbers:
  - From 4 students (others) to 6 students (our technique)
  - From 2 clients and 2 sessions to 5 sessions and 5 clients
  - Handled the GSM network within < 1 second, *MWB* fails, *HAL* uses 18 seconds

Conclusion and Thanks

Take home message: there is an efficient way of checking FCPs with net unfoldings

Everything is implemeted:
`http://petruchio.informatik.uni-oldenburg.de`

Thanks for your attention

References I

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

R. Devillers, H. Klaudel, and M. Koutny.
A Petri net semantics of the finite π-Calculus terms.
Fundamenta Informaticae, 70(3):203–226, 2006.

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.

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

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

References II

B. Victor and F. Moller.
The mobility workbench: A tool for the π-Calculus.
In *Proc. CAV'94*, volume 818 of *Lecture Notes in Computer Science*, pages 428–440. Springer-Verlag, 1994.