Information Systems 2

5. Business Process Modelling I: Models

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Business Economics and Information Systems
& Institute for Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

1. Petri Nets

2. The Pi Calculus
Overview

– Petri nets are models for parallel computation.

– A Petri net represents a parallel system as graph of component states (places) and transitions between them.

– You can execute Petri nets online (jPNS) at http://robotics.ee.uwa.edu.au/pns/java/

There also is a more advanced open source Petri net editor (PIPE2):
http://pipe2.sourceforge.net/

– Petri Nets have been invented by the German mathematician Carl Adam Petri in 1962.

Definition

A **Petri net** is a directed graph \((P \cup T, F)\) over two (disjunct) sorts of nodes, called **places** \(P\) and **transitions** \(T\) respectively, where

– all roots and leaves are places and

– edges connect only places with transitions, and not places with places or transitions with transitions, i.e., \(F \subseteq (P \times T) \cup (T \times P)\).

Graphical representation:

places — circles

transitions — bars (or boxes)
The components of a Petri net have the following interpretation:

– places denote a **stopping point in a process** as, e.g., the attainment of a milestone; from the perspective of a transition, a place denotes a **condition**.
– transitions denote an **event** or **action**.

**Inputs and Outputs**

**inputs / preconditions** of a transition \( t \in T \): the places with edges into \( t \), i.e.,

\[
\bullet t := \text{fanin}(t) := \{ p \in P \mid (p, t) \in F \}
\]

**outputs / postconditions** of a transition \( t \in T \): the places with edges from \( t \), i.e.,

\[
t\bullet := \text{fanout}(t) := \{ p \in P \mid (t, p) \in F \}
\]
State of a Petri Net

The state of a Petri net is described by the **markings** of the places by **tokens**, i.e.,

\[ M : P \to \mathbb{N} \]

where \( M(p) \) denotes the number of tokens assigned to place \( p \in P \) at a given point in time.

**Graphical representation:**

- tokens — black dots

![Petri Net Diagram](image)

State Change of a Petri Net

A transition \( t \in T \) is said to be **enabled** if each of its inputs contains at least one token, i.e.,

\[ M(p) \geq 1 \quad \forall p \in \bullet t \]

An enabled transition \( t \in T \) may fire, i.e., change the state of the Petri net from a state \( M \) into a new state \( M^{\text{new}} \) by

- remove one token from each of its inputs and
- add one token to each of its outputs, i.e.,

\[ M^{\text{new}}(p) := M(p) - 1 \quad \forall p \in \bullet t, \]
\[ M^{\text{new}}(p) := M(p) + 1 \quad \forall p \in t\bullet \]

The new state is also denoted by \( t(M) := M^{\text{new}} \).

If several transitions are enabled, the next transition to fire is choosen at random.
Information Systems 2 / 1. Petri Nets

State Change of a Petri Net / Example

Lars Schmidt-Thieme, Information Systems and Machine Learning Lab (ISMLL), Institute BW/WI & Institute for Computer Science, University of Hildesheim
Course on Information Systems 2, summer term 2008

7/29
State Change of a Petri Net / Example

AND vs. OR

AND: both inputs are required.

OR: at least one input is required.
OR vs. XOR

OR: at least one input is required.

XOR: exactly one input is required.

Example (1/4)

Assume there is a robot with three states:
- P0 robot works outside special workplace
- P1 robot waits for access to special workplace
- P2 robot works inside special workplace

and three events:
- T0 finish work outside special workplace
- T1 enter special workplace
- T2 finish work in special workplace

that works repeatedly:
A system consisting of two such robots can be described as follows:

Now assume the special workplace cannot be used by both robots at the same time:

with additional place:

P3 special workplace available
Now assume a third robot assembles one component produced by the two robots each immediately and its input buffer can hold maximal 4 components.

with additional places:
- P4 buffer place for component 2 available
- P5 buffer place for component 1 available

Reachability

A given marking \( N \) of a Petri net is said to be \textbf{reachable from a marking} \( M \) if there exist transitions \( t_1, t_2, \ldots, t_n \in T \) with

\[
N = t_n(t_{n-1}(\ldots t_2(t_1(M))\ldots))
\]

Example:

1. The state

\[
P0 = 0, P1 = 0, P2 = 1, P0b = 1, P1b = 0, P2b = 0, P3 = 0
\]

denoting the first robot to work in the special workplace while the second works outside, is reachable for the net robots (3/4) from the initial marking

\[
P0 = 1, P1 = 0, P2 = 0, P0b = 1, P1b = 0, P2b = 0, P3 = 1
\]

by the transition sequence \( T0, T1 \).

2. The state

\[
P0 = 0, P1 = 0, P2 = 1, P0b = 0, P1b = 0, P2b = 1, P3 = 0
\]

denoting both robots to work in the special workplace, is not reachable from the initial state.
For $k \in \mathbb{N}$, a Petri net is called $k$-bounded for an initial marking $M$ if no state with a place containing more than $k$ tokens is reachable from $M$.

A Petri net is called save for an initial marking $M$, if it is 1-bounded for $M$. 

Two robots with states:
- P0 robot available
- P1 robot works on component

and events:
- T0 start working
- T1 finish work on component

working in sequence.

P2 input component for second robot available
The former example is not bounded as the first robot could produce arbitrary many tokens in P2 without the second robot ever consuming one.

Introducing a new place

P3  buffer place available

with initially 3 tokens renders the example 3-bounded.

Deadlock

A mutex can easily produce a `deadlock`, i.e., all processes waiting for the availability of the mutex.
1. Petri Nets

2. The Pi Calculus

Overview

- The \( \pi \)-calculus is another model for concurrent computation.

- The \( \pi \)-calculus is a formal language for defining concurrent communicating processes (usually called agents).

- The \( \pi \)-calculus models relies on message passing between concurrent processes.

- The \( \pi \)-calculus got his name to resemble the lambda calculus, the minimal model for functional programming (Church/Kleene 1930s). Here \( \pi \) (= greek p) as “parallel”.

- The \( \pi \)-calculus was invented by the Scottish mathematician Robin Milner in the 1990s.
Initial Example

\[
\begin{align*}
(b(a).S) & \mid (b(x).\bar{x}(d).P) \xrightarrow{\tau} S \mid \bar{a}(d).P
\end{align*}
\]

Agents

Let \( \mathcal{X} \) be a set of atomic elements, called names.

An agent is defined as follows:

\[
R ::= 0 \quad \text{do nothing}
\]

\[
x(y).P \quad \text{send data } y \text{ to channel } x, \text{ then proceed as } P
\]

\[
x(y).P \quad \text{receive data into } y \text{ from channel } x, \text{ then proceed as } P
\]

\[
P + Q \quad \text{proceed either as } P \text{ or as } Q
\]

\[
P \mid Q \quad \text{proceed as } P \text{ and as } Q \text{ in parallel}
\]

\[
(\nu x)P \quad \text{create fresh local name } x
\]

\[
!P \quad \text{arbitrary replication of } P, \text{ i.e., } P|P|P|\ldots
\]

where \( P, Q \) are agents and \( x, y \in \mathcal{X} \) are names.

To modularize complex agents, one usually allows definitions of abbreviations as

\[
A(x_1, x_2, \ldots, x_n) ::= P
\]

as well as using such definitions

\[
A(x_1, x_2, \ldots, x_n) \quad \text{proceed as defined by } A
\]

where \( P \) is an agent and \( A \) is a name.
Bound and Free Names

There are two ways to bind a name \( y \) in \( \pi \)-calculus:

- by receiving into a name: \( x(y).P \).
- by creating a name: \( (\nu y)P \).

All free / unbound names figure as named constants that agents must agree on:

<table>
<thead>
<tr>
<th>agent</th>
<th>free names</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \emptyset )</td>
</tr>
<tr>
<td>( \bar{x}(y).P )</td>
<td>free(( P )) ( \cup {x, y} )</td>
</tr>
<tr>
<td>( x(y).P )</td>
<td>free(( P )) ( \setminus {y} \cup {x} )</td>
</tr>
<tr>
<td>( P + Q )</td>
<td>free(( P )) ( \cup ) free(( Q ))</td>
</tr>
<tr>
<td>( P \mid Q )</td>
<td>free(( P )) ( \cup ) free(( Q ))</td>
</tr>
<tr>
<td>( (\nu x)P )</td>
<td>free(( P )) ( \setminus {x} )</td>
</tr>
<tr>
<td>( !P )</td>
<td>free(( P ))</td>
</tr>
</tbody>
</table>

Therefore one defines a notion of equivalent formulas (structurally equivalent).
The following agents are said to be **structurally congruent**:

\[
P \equiv Q \quad \text{if} \ P \text{ and } Q \text{ differ only in bound names}
\]

\[
P + Q \equiv Q + P \quad \text{+-symmetry}
\]

\[
P + 0 \equiv P \quad \text{+-neutrality of 0}
\]

\[
P \mid Q \equiv Q \mid P \quad \mid\text{-symmetry}
\]

\[
P \mid 0 \equiv P \quad \mid\text{-neutrality of 0}
\]

\[
!P \equiv P \mid !P \quad \text{!-expansion}
\]

\[
(\nu x)0 \equiv 0 \quad \text{restriction of null}
\]

\[
(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P \quad \nu\text{-communtativity}
\]

\[
(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q \quad \text{if } x \notin \text{free}(P)
\]

Formulas

\[
(a(x).0) \quad \text{and} \quad (\bar{a}(b).0)
\]

are abbreviated as

\[
a(x) \quad \text{and} \quad \bar{a}(b)
\]

respectively.
Structured Messages

Often one agent needs to pass a message that consists of several parts.

Just sending both parts sequentially, may lead to garbled messages. Example:

\[
(a(x).a(y)) | (\bar{a}(b_1).\bar{a}(c_1)) | (\bar{a}(b_2).\bar{a}(c_2))
\]

intends to sent either \((b_1, c_1)\) or \((b_2, c_2)\) and bind it to \((x, y)\), but it may happen that actually the second agent sends \(b_1\), then the third \(b_2\), so \((x, y)\) is bound to \((b_1, b_2)\).

Private channels can avoid this problem:

\[
\bar{a}(b_1, b_2, \ldots, b_n) := (\nu w)(\bar{a}(w).\bar{w}(b_1).\bar{w}(b_2).\ldots.\bar{w}(b_n))
\]

\[
a(x_1, x_2, \ldots, x_n) := a(w).w(b_1).w(b_2).\ldots.w(b_n)
\]

Now the example can we written as

\[
a(x, y) | \bar{a}(b_1, c_1) | \bar{a}(b_2, c_2)
\]

and just the private channel name \(w\) is exchanged via the public channel \(a\), the actual data \((b_1, c_1)\) is sent via the private channel \(w\).

An Example (1/3)

4 concurrent agents: car, two bases and centre.

8 named channels: talk \(t_1, t_2\), switch \(s_1, s_2\), give \(g_1, g_2\), alert \(a_1, a_2\).

First base uses channels \(t_1\) and \(s_1\) to communicate with car, \(g_1\) and \(a_1\) to communicate with centre.

Second base uses channels \(t_2\) and \(s_2\) to communicate with car, \(g_2\) and \(a_2\) to communicate with centre.
An Example (2/3)

\[
\text{System}_1 := (\nu t_1, t_2, s_1, s_2, g_1, g_2, a_1, a_2) \\
\quad (\text{Car}(t_1, s_1) | \text{Base}(t_1, s_1, g_1, a_1) | \text{IdleBase}(t_2, s_2, g_2, a_2) | \text{Centre}_1) \\
\quad \equiv \ldots | \ldots + g_1(t', s').\bar{s}_1(t', s').\text{IdleBase}(t_1, s_1, g_1, a_1) | \ldots | (\bar{g}_1(t_2, s_2).\bar{a}_2) . \text{Centre}_2
\]

\[
\rightarrow \ldots | (\bar{s}_1(t_2, s_2).\text{IdleBase}(t_1, s_1, g_1, a_1) | \ldots | (\bar{a}_2) . \text{Centre}_2) \\
\equiv (\ldots + s_1(t', s').\text{Car}(t', s').)(\bar{s}_1(t_2, s_2).\text{IdleBase}(t_1, s_1, g_1, a_1) | \ldots | (\bar{a}_2) . \text{Centre}_2)
\]

\[
\rightarrow \text{Car}(t_2, s_2) | \text{IdleBase}(t_1, s_1, g_1, a_1) | \ldots | (\bar{a}_2) . \text{Centre}_2 \\
\equiv \text{Car}(t_2, s_2) | \text{IdleBase}(t_1, s_1, g_1, a_1) | (\bar{a}_2) . \text{Base}(t_2, s_2, g_2, a_2) | (\bar{a}_2) . \text{Centre}_2
\]

\[
\rightarrow \text{Car}(t_2, s_2) | \text{IdleBase}(t_1, s_1, g_1, a_1) | \text{Base}(t_2, s_2, g_2, a_2) | \text{Centre}_2
\]

An Example (3/3)
References
