

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>

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 2013 1/29

Information Systems 2

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.

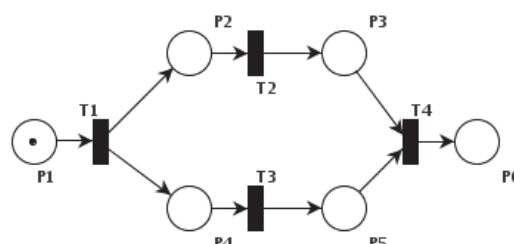
Definition

A **Petri net** is a directed graph $(P \dot{\cup} T, F)$ over two (disjoint) 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:

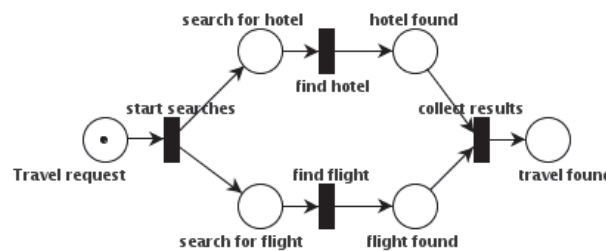
places — circles
transitions — bars (or boxes)



Interpretation

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**.



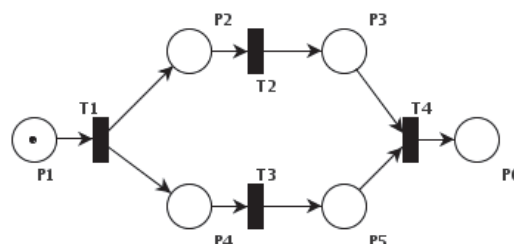
Inputs and Outputs

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

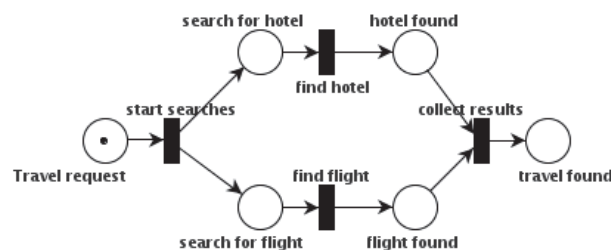
The state of a Petri net is described by the **markings** of the places by **tokens**, i.e.,

$$M : P \rightarrow \mathbb{N}$$

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



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^{new} by

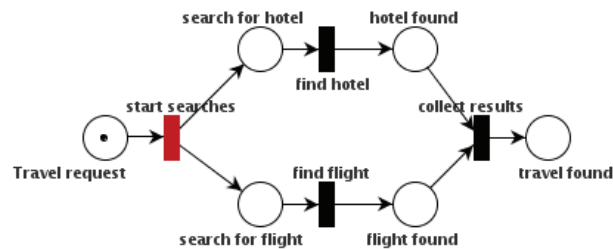
- remove one token from each of its inputs and
- add one token to each of its outputs, i.e.,

$$\begin{aligned} M^{\text{new}}(p) &:= M(p) - 1 \quad \forall p \in \bullet t, \\ M^{\text{new}}(p) &:= M(p) + 1 \quad \forall p \in t\bullet \end{aligned}$$

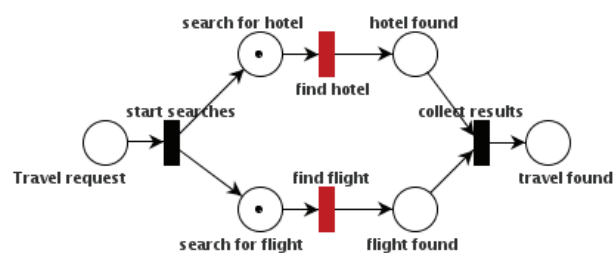
The new state is also denoted by $t(M) := M^{\text{new}}$.

If several transitions are enabled, the next transition to fire is chosen at random.

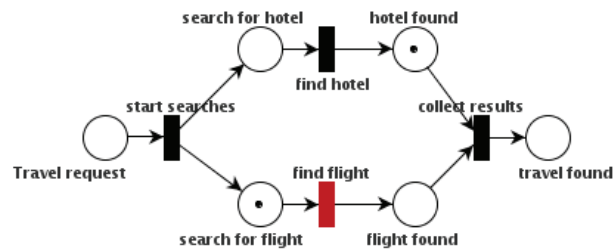
State Change of a Petri Net / Example



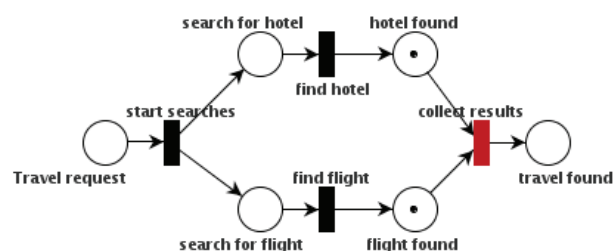
State Change of a Petri Net / Example



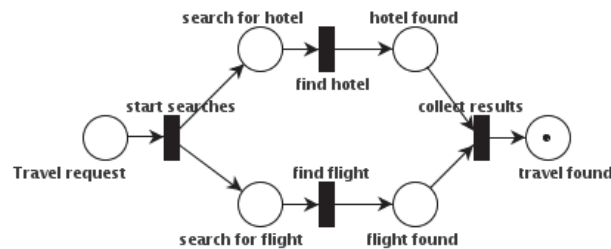
State Change of a Petri Net / Example



State Change of a Petri Net / Example



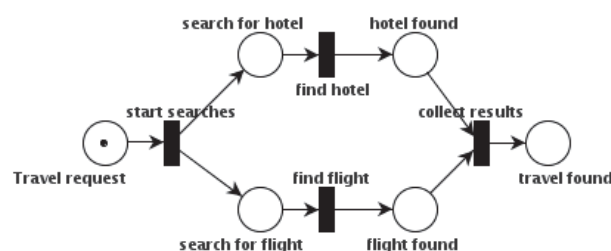
State Change of a Petri Net / Example



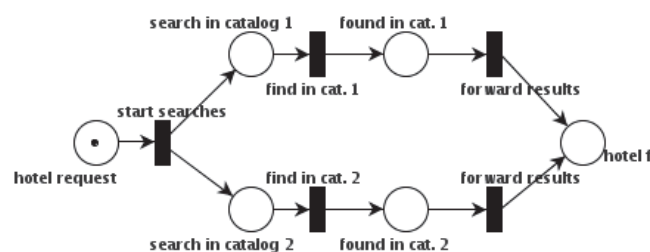
Information Systems 2 / 1. Petri Nets

AND vs. OR

AND: both inputs are required.

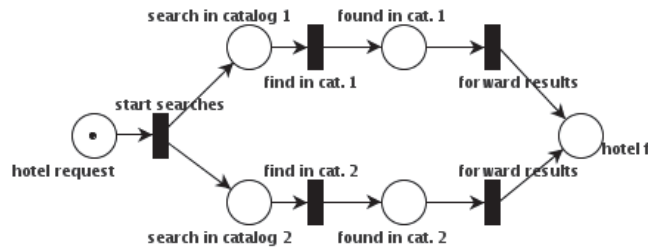


OR: at least one input is required.

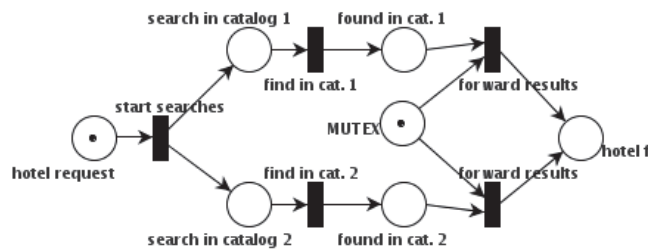


OR vs. XOR

OR: at least one input is required.



XOR: exactly one input is required.



Example (1/4)

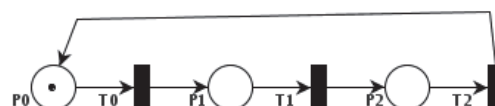
Assume there is a robot with three states:

- P0 robot works outside special workplace
- P1 robot waits for access to special workplace
- P2 robot works inside special workplace

and three events:

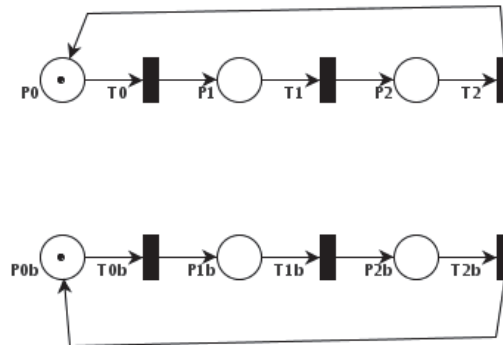
- T0 finish work outside special workplace
- T1 enter special workplace
- T2 finish work in special workplace

that works repeatedly:



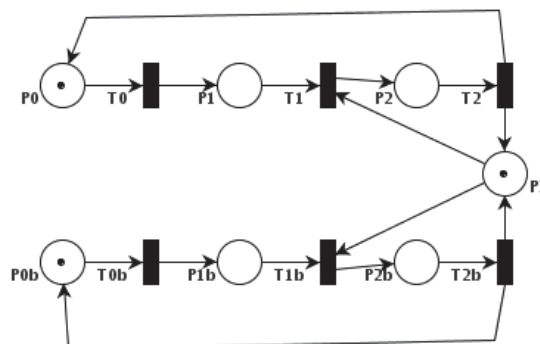
Example (2/4)

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



Example (3/4)

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

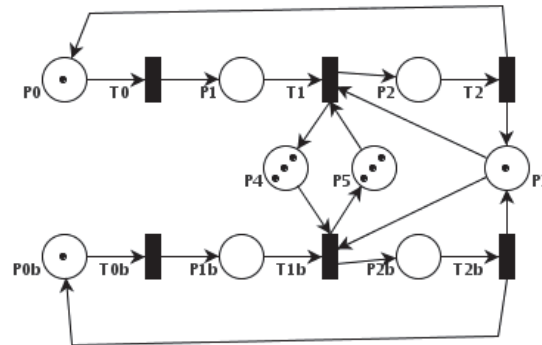


with additional place:

P3 special workplace available

Example (4/4)

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 **reachable from a marking M** if there exist transitions $t_1, t_2, \dots, t_n \in T$ with

$$N = t_n(t_{n-1}(\dots t_2(t_1(M)) \dots))$$

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.

Boundedness and Saveness

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 .

Boundedness and Saveness / Example (1/2)

Two robots with states:

P0 robot available

P1 robot works on component

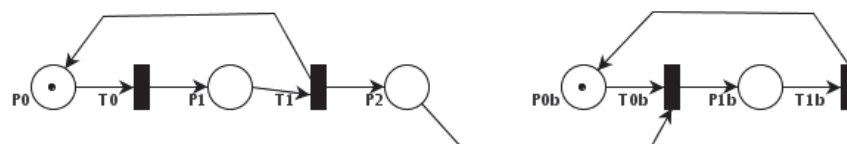
and events:

T0 start working

T1 finish work on component

working in sequence.

P2 input component for second robot available



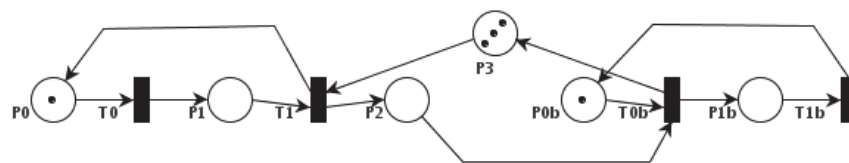
Boundedness and Saveness / Example (2/2)

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

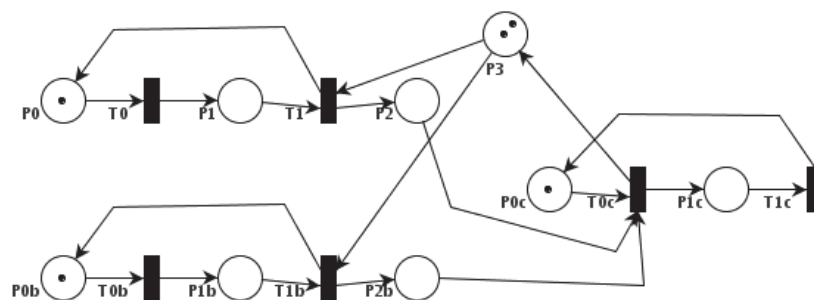
P3 buffer place available

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.



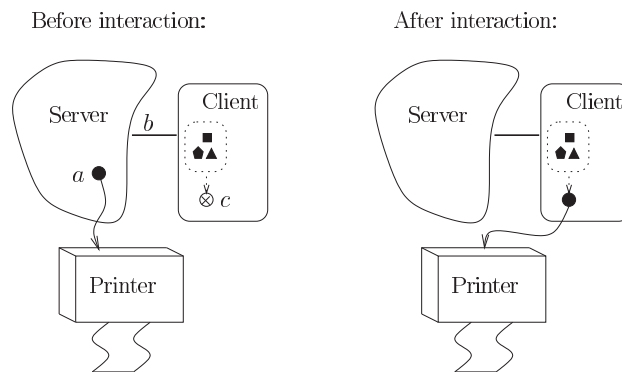
1. Petri Nets

2. The Pi Calculus

Overview

- The π -calculus is a another model for concurrent computation.
- The π -calculus is a formal language for defining concurrent communicating processes (usually called agents).
- The π -calculus relies on message passing between concurrent processes.
- The π -calculus got his name to resemble the lambda calculus, the minimal model for functional programming (Church/Kleene 1930s).
Here π (= greek p) as “parallel”.
- The π -calculus was invented by the Scottish mathematician Robin Milner in the 1990s.

Initial Example



[Par01]

$$(\bar{b}\langle a \rangle.S) \mid (b(x).\bar{x}\langle d \rangle.P) \xrightarrow{\tau} S \mid \bar{a}\langle d \rangle.P$$

Agents

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

An **agent** is defined as follows:

- $R ::= 0$ do nothing
- $\bar{x}\langle y \rangle.P$ send data y to channel x , then proceed as P
- $x(y).P$ receive data into y from channel x , then proceed as P
- $P + Q$ proceed either as P or as Q
- $P \mid Q$ proceed as P and as Q in parallel
- $(\nu x)P$ create fresh local name x
- $!P$ arbitrary replication of P , i.e., $P \mid P \mid P \mid \dots$

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

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

$$A(x_1, x_2, \dots, x_n) := P$$

as well as using such definitions

$$A(x_1, x_2, \dots, x_n) \text{ proceed as defined by } A$$

where P is an agent and A is a name

Bound and Free Names

There are two ways to bind a name y in π -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:

agent	free names
0	\emptyset
$\bar{x}\langle y \rangle.P$	$\text{free}(P) \cup \{x, y\}$
$x(y).P$	$\text{free}(P) \setminus \{y\} \cup \{x\}$
$P + Q$	$\text{free}(P) \cup \text{free}(Q)$
$P \mid Q$	$\text{free}(P) \cup \text{free}(Q)$
$(\nu x)P$	$\text{free}(P) \setminus \{x\}$
$!P$	$\text{free}(P)$

Structural Congruence / Example

The same agent can be expressed by different formulas:

$$(\bar{b}\langle a \rangle.S) \mid (b(x).\bar{x}\langle d \rangle.P)$$

$$(b(x).\bar{x}\langle d \rangle.P) \mid (\bar{b}\langle a \rangle.S)$$

$$(b(y).\bar{y}\langle d \rangle.P) \mid (\bar{b}\langle a \rangle.S)$$

Therefore one defines a notion of equivalent formulas (structurally equivalent).

Structural Congruence

The following agents are said to be **structurally congruent**:

$P \equiv Q$	if P and Q differ only in bound names
$P + Q \equiv Q + P$	+symmetry
$P + 0 \equiv P$	+neutrality of 0
$P \mid Q \equiv Q \mid P$	symmetry
$P \mid 0 \equiv P$	neutrality of 0
$!P \equiv P \mid !P$!expansion
$(\nu x)0 \equiv 0$	restriction of null
$(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$	ν -commutativity
$(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$	if $x \notin \text{free}(P)$

Formulas

$$(a(x).0) \quad \text{and} \quad (\bar{a}\langle b \rangle.0)$$

are abbreviated as

$$a(x) \quad \text{and} \quad \bar{a}\langle b \rangle$$

respectively.

Reduction

A **reduction** $P \rightarrow Q$ describes that P results in Q by parallel computation.

Reduction rules:

communication:

$$(\dots + \bar{x}\langle z \rangle.P) \mid (\dots + x(y).Q) \longrightarrow P \mid Q[z/y]$$

reduction under composition:

$$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}$$

reduction under restriction:

$$\frac{P \longrightarrow Q}{(\nu x)P \longrightarrow (\nu x)Q}$$

same reduction for structurally equivalent agents:

$$\frac{P \longrightarrow Q \quad P \equiv P' \quad Q \equiv Q'}{P' \longrightarrow Q'}$$

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:

$$(a(x).a(y)) \mid (\bar{a}\langle b_1 \rangle.\bar{a}\langle c_1 \rangle) \mid (\bar{a}\langle b_2 \rangle.\bar{a}\langle c_2 \rangle)$$

intends to send 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:

$$\begin{aligned} \bar{a}\langle b_1, b_2, \dots, b_n \rangle &:= (\nu w)(\bar{a}\langle w \rangle.\bar{w}\langle b_1 \rangle.\bar{w}\langle b_2 \rangle.\dots.\bar{w}\langle b_n \rangle) \\ a(x_1, x_2, \dots, x_n) &:= a(w).w(b_1).w(b_2).\dots.w(b_n) \end{aligned}$$

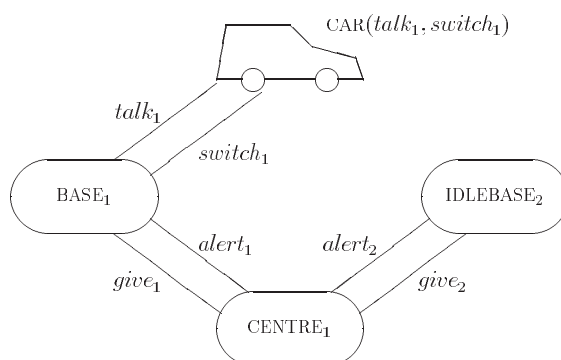
Now the example can be written as

$$a(x, y) \mid \bar{a}\langle b_1, c_1 \rangle \mid \bar{a}\langle b_2, c_2 \rangle$$

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)



[Mil93]

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)

$$\begin{aligned}
\text{System}_1 &:= (\nu t_1, t_2, s_1, s_2, g_1, g_2, a_1, a_2) \\
&\quad (\text{Car}(t_1, s_1) \mid \text{Base}(t_1, s_1, g_1, a_1) \mid \text{IdleBase}(t_2, s_2, g_2, a_2) \mid \text{Centre}_1) \\
\text{Car}(t, s) &:= t().\text{Car}(t, s) + s(t', s').\text{Car}(t', s') \\
\text{Base}(t, s, g, a) &:= t().\text{Base}(t, s, g, a) + g(t', s').\bar{s}\langle t', s' \rangle.\text{IdleBase}(t, s, g, a) \\
\text{IdleBase}(t, s, g, a) &:= a().\text{Base}(t, s, g, a) \\
\text{Centre}_1 &:= \bar{g}_1\langle t_2, s_2 \rangle.\bar{a}_2\langle \rangle.\text{Centre}_2 \\
\text{Centre}_2 &:= \bar{g}_2\langle t_1, s_1 \rangle.\bar{a}_1\langle \rangle.\text{Centre}_1
\end{aligned}$$

An Example (3/3)

$$\begin{aligned}
\text{System}_1 &:= (\nu t_1, t_2, s_1, s_2, g_1, g_2, a_1, a_2) \\
&\quad (\text{Car}(t_1, s_1) \mid \text{Base}(t_1, s_1, g_1, a_1) \mid \text{IdleBase}(t_2, s_2, g_2, a_2) \mid \text{Centre}_1) \\
&\equiv \dots \mid (\dots + \textcolor{red}{g}_1(t', s').\bar{s}_1\langle t', s' \rangle.\text{IdleBase}(t_1, s_1, g_1, a_1) \mid \dots \mid (\bar{\textcolor{red}{g}}_1\langle t_2, s_2 \rangle.\bar{a}_2\langle \rangle.\text{Centre}_2) \\
&\rightarrow \dots \mid (\bar{s}_1\langle \textcolor{red}{t}_2, \textcolor{red}{s}_2 \rangle.\text{IdleBase}(t_1, s_1, g_1, a_1) \mid \dots \mid (\bar{a}_2\langle \rangle.\text{Centre}_2) \\
&\equiv (\dots + \textcolor{red}{s}_1(t', s').\text{Car}(t', s')) \mid (\bar{s}_1\langle \textcolor{red}{t}_2, \textcolor{red}{s}_2 \rangle.\text{IdleBase}(t_1, s_1, g_1, a_1) \mid \dots \mid (\bar{a}_2\langle \rangle.\text{Centre}_2) \\
&\rightarrow \text{Car}(\textcolor{red}{t}_2, \textcolor{red}{s}_2) \mid \text{IdleBase}(t_1, s_1, g_1, a_1) \mid \dots \mid (\bar{a}_2\langle \rangle.\text{Centre}_2) \\
&\equiv \text{Car}(\textcolor{red}{t}_2, \textcolor{red}{s}_2) \mid \text{IdleBase}(t_1, s_1, g_1, a_1) \mid (\textcolor{red}{a}_2().\text{Base}(t_2, s_2, g_2, a_2)) \mid (\bar{a}_2\langle \rangle.\text{Centre}_2) \\
&\rightarrow \text{Car}(\textcolor{red}{t}_2, \textcolor{red}{s}_2) \mid \text{IdleBase}(t_1, s_1, g_1, a_1) \mid \text{Base}(t_2, s_2, g_2, a_2) \mid \text{Centre}_2
\end{aligned}$$

References

- [Mil93] Robin Milner. The polyadic pi-calculus: A tutorial. In F. L. Hamer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*. Springer, 1993.
- [Par01] Joachim Parrow. An introduction to the π -calculus. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*. Elsevier, 2001.