CARL VON OSSIETZKY universität OLDENBURG

Fakultät II: Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik
Abteilung: Entwicklung korrekter Systeme

Transformational semantics of the combination π-*OZ* for mobile processes with data

Masterarbeit

- post version -

Name:	Muhammad Ekbal Ahmad
E-Mail:	muhammad.ekbal.ahmad@uni-oldenburg.de

Studiengang:	Fach-Master Informatik
Erstgutachter:	Prof. Dr. Ernst-Rüdiger Olderog
Zweitgutachter:	M.Sc. Manuel Gieseking
Datum:	20. Januar 2020

Contents

List of Figures

1 Introduction

In many cases of modern computing it is of interest to describe and model concurrency. Computers no longer just solve a problem by subsequently working off the single tasks of their own, but they decompose and concurrently calculate the problem even together in a network. The increase in the number of CPU cores and more heavily of GPU cores within one single computer convincingly demonstrates how fundamental concurrency is for modern computing. Moreover, the rapidly increasing spread of the Internet is one of the most common examples which shows the importance of networks.

This thesis is divided into five chapters. In Chapter 2 we briefly introduce sequences and properly investigate the π-calculus and its operational semantics (the *early transition system* [SW01]). Thereby, we investigate its properties and define the refinement based on the trace semantics. Finally, the conclusion in Chapter 3 gives a brief summary of our results and presents ideas for future work.

2 Preliminaries

At the heart of the refinement of *π-calculus* processes is the theory of *sequences*. Thus, in this chapter, we recall the model of sequences to gain a formal construct to handle ordered elements.

Furthermore, we introduce the π-calculus and investigate its behavior properly. In particular, we carefully explain the operational semantics of π-calculus processes, since its peculiarities induce the characteristics of the refinement and its properties. Moreover, we discuss why we choose this particular operational semantics for the following work in this thesis and compare it to other semantics.

The majority of those definitions and notions can, for example, be found in [Mil99, SW01].

As mathematical notations, we consider the natural numbers starting with zero ($\mathbb{N} = \{0, 1, 2, \ldots\}$) and use $\fatsemi$ as the composition of relations. Furthermore, we denote R^* as the reflexive and transitive closure of a relation R.

2.1 The π-calculus

The *π-calculus* is a process algebra that can be used to describe the behavior. This section introduces the pure polyadic version of the π-calculus as decriped in [Mil99].

2.1.1 Intuition

To explain the π-calculus inuitivly we will use the ion example as in [Mil99]. Let us imagine a positive and a negative ion. When those two ions merge, we get a new construct. The merge operation is called a *reaction*, since an ion acts and the other reacts. This reaction can be seen as communication between two processes. The two processes communicate to share some information. One process is the sender and the other is the receiver. By doing the reaction both processes evoulte to some thing new. The reaction, information sharing and evolution concepts are the core of the π-calculus. Using those concepts we can understand the title of

Milner's book *communicating and mobile processes: the* π-calculus [Mil99]. The word *communicating* refers to the *reaction* concept. The word *mobile* refers to the *information sharing and evolution* concepts, since the receiver process can use the received information to change its location as we will see in .. .
The π-calculus consists of:

- a set of names starting with capital case letters like $P, P_1, Q, ..etc$ used to refer to a process directly.
- a set of names starting with capital case letters like $A, B, C, ..etc$ used a process identifier. The process identifier will be used to define recursion with parameters.
- a set of names starting with lower case letter like $a, b, x, y, ..etc$ used as a channel and message name. This set is denoted by $\mathcal{N}$.
- operators like:
  - Parallel composition operator: " | ".
  - Sequential composition operator: " . ".
  - Choice operator: " + ".
  - Scope restriction operator: " $\underline{\texttt{new}}$ ".

So a simple example of a process can be: $\overline{x}\langle y\rangle.\mathbf{0}$ this process simply sends the message y via the channel x and stops. The full syntax of π-calculus process is given in Definition 2.1.1. In this thesis starting from this point, when we mention the word *names* we refer to $\mathcal{N}$. Furthermore, we shall often write $\vec{y}$ for a sequence $y_1, ..., y_n$ of names.

2.1.2 Syntax

Definition 2.1.1 (Process syntax) The syntax of a π-calculus process P is defined by:

$$P ::= \sum_{i\in I} \pi_i.P_i \;\Big|\; P_1 \mid P_2 \;\Big|\; \underline{\texttt{new}}\,\vec{y}\,P \;\Big|\; A\langle\vec{v}\rangle$$

where:

- $\sum_{i\in I} \pi_i.P_i$ is the guarded sum.

- $P_1 \mid P_2$ is the parallel composition of processes.
- $\underline{\texttt{new}}\,\vec{y}\,P$ is the restriction of the scope of the names $\vec{y}$ to the process P
- $A\langle\vec{v}\rangle$ is a process call. *

***Guarded sum*:**

Is the *choice* between multiple guarded processes. If the guard of one process took place, other guarded processes will be discarded. For example, the processes: $x().P_1 + y().P_2$ will evolve to the process P_1 if the guard $x()$ occurred.

Furthermore, The process **0** is called the *stop process* or *inaction* and stands for the process that can do nothing. It can be omitted.

***Guard*:**

Also called *action prefix* and denoted by π. It's syntax is defined by:

Definition 2.1.2 (Action prefix syntax)

$$\pi ::= \overline{x}\langle\vec{y}\rangle \;\Big|\; x(\vec{y}) \;\Big|\; \tau$$

where:

- $\overline{x}\langle\vec{y}\rangle$ [1] represents the action: send $\vec{y}$ via the channel x.
- $x(\vec{y})$ [2] represents the action: receive $\vec{y}$ via the channel x.
- τ represents an internal non observable action. *

The set of all *actions* is defined as $\texttt{Act} =_{\text{def}} \texttt{Out} \cup \texttt{In} \cup \{\tau\}$, where:

- $\texttt{Out}$ is the set of all *output actions*, defined as $\texttt{Out} =_{\text{def}} \{\overline{x}\langle\vec{y}\rangle \mid x \in \mathcal{N}\}$.
- $\texttt{In}$ the set of all *input actions*, defined as $\texttt{In} =_{\text{def}} \{x(\vec{y}) \mid x \in \mathcal{N}\}$.

[1] $\overline{x}\langle\rangle$ means: send a signal via x. $\overline{x}\langle y\rangle$ means: send the name y via x. $\overline{x}\langle\vec{y}\rangle$ means: send the sequence $\vec{y}$ via x.

[2] $x()$ means: receive a signal via x. $x(y)$ means: receive any name y via x. $x(\vec{y})$ means: receive any sequence $\vec{y}$ via x. "y here plays the role of parameter"

***Parallel composition*:**

The parallel composition operator | represent the concept of concurrency in the π-calculus, where two processes can evolve in concurrent.It represents an interleaving behavior of the concurrency. For example let: $P =_{\text{def}} P_1 \mid (P_2 \mid P_3)$ where: $P_1 =_{\text{def}} x(y).Q_1$, $P_2 =_{\text{def}} \overline{x}\langle y\rangle.Q_2$ and $P_3 =_{\text{def}} x(y).Q_3$. So $P =_{\text{def}} x(y).Q_1 \mid (\overline{x}\langle y\rangle.Q_2 \mid x(y).Q_3)$. Possible evolution cases of P are:

- $P_1 \mid (Q_2 \mid Q_3)$. P_2 sends y via x to P_3.
- $Q_1 \mid (Q_2 \mid P_3)$. P_2 sends y via x to P_1.

The example above illustrated the privacy nature of the parallel operator in the π-calculus. A process can via a channel communicate with only one process pro time.P_2 cannot communicate with both P_1, P_3 in the same time, while in Communicating Sequential Processes (CSP) a process can communicate with multiple processes in the same time via the same channel by sending multiple copies of the same message, in other words: in CSP the channel represents a Hub.

***Restriction*:**

The expression $\underline{\texttt{new}}\,\vec{y}\;P$ binds the names $\vec{y}$ to the process P. In other words: the visibility scope of the names $\vec{y}$ is restricted to the process P. It is similar to declaring a private variable in programming languages. Thus the names $\vec{y}$ are not visible outside P and P cannot use them to communicate with outside. For example, let $P =_{\text{def}} P_1 \mid P_2$ where: $P_1 =_{\text{def}} \underline{\texttt{new}}\,y\;\overline{y}\langle z\rangle.Q_1$ and $P_2 =_{\text{def}} y(z).Q_2$. The process P cannot evolute to $Q_1 \mid Q_2$, since the name y in P_1 is only visible inside it, i.e., from the P_2's point of view P_1 doesn't have a channel called y. This takes us to the definition of the Bound and free names.

Definition 2.1.3 (Free names) are all the restricted names in a process. ∗

Definition 2.1.4 (Bound names) are all the name that occur in a process except the bound names. ∗

For example, let $P_1 =_{\text{def}} \underline{\texttt{new}}\,x\;\overline{x}\langle y\rangle.P_2$ where $P_2 =_{\text{def}} \underline{\texttt{new}}\,z\;\overline{x}\langle z\rangle.P_3$. The name x is bound in P_1 but free in P_2.

***Process call*:**

Let P be a process and let A be a process identifier.To be able to use the process P recursively we use the process identifier A as follow: $A(\vec{w}) =_{\text{def}} P$. Thus , when we write $A\langle\vec{v}\rangle$ we are using the identifier A to call the process P with replacing the names $\vec{w}$ in P with the names $\vec{v}$. This replacement is called the α-conversion

For example, let $P =_{\text{def}} \overline{w}\langle y\rangle.\mathbf{0}$ and let $A(w) =_{\text{def}} P$ be the recursive definition of the process P, then the behavior of $A\langle v\rangle$is equlivant to $\overline{v}\langle y\rangle.\mathbf{0}$

2.1.3 Semantics

To understand the operational semantics of π-calculus we will use a labelled transition system LTS. Using this LTS we can investigate π-calculus process evolution. The definition of LTS is adapted from [Mil99] pages 39[3], 91[4], 132[5] with some changes.

Definition

Definition 2.1.5 (The LTS of π-calculus) The labelled transition system $(\mathcal{P}^{\pi}, \mathcal{T})$ of π-calculus processes over the action set `Act` has the process expressions $\mathcal{P}^{\pi}$ as its states, and its transitions $\mathcal{T}$ are those which can be inferred from the rules in Figure 2.1. The rule REACT is the most important one. It shows the process evolution when a reaction occurs. The reaction requires two complementary transitions $P \xrightarrow{\overline{x}\langle\vec{y}\rangle} P'$ and $Q \xrightarrow{x(\vec{z})} Q'$, we call them commitments. so the process P takes a commitment to take part in the reaction, and so does Q.

[3]Transition Rules: LTS for concurrent processes not for π-calculus processes.
[4]Reaction Rules: no labels and no LTS.
[5]Commitment Rules: abstractions and concretions are out of this thesis's scope.

$$\underline{OUT} : \overline{x}\langle\vec{y}\rangle.P \xrightarrow{\overline{x}\langle\vec{y}\rangle} P \qquad \underline{IN} : x(\vec{y}).P \xrightarrow{x(\vec{y})} P$$

$$\underline{TAU} : \tau.P \xrightarrow{\tau} P \qquad \underline{SUM} : \alpha.P + \sum_{i\in I} \pi_i.P_i \xrightarrow{\alpha} P$$

$$\underline{L\_PAR} : \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \underline{R\_PAR} : \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

$$\underline{RESTRICTION} : \frac{P \xrightarrow{\alpha} P'}{\underline{\texttt{new}}\, x\ P \xrightarrow{\alpha} \underline{\texttt{new}}\, x\ P'}\ if\ \alpha \notin \{\overline{x}, x\}$$

$$\underline{PROCESS\_CALL} : \frac{\{\vec{y}/\vec{z}\}\, P \xrightarrow{\alpha} P'}{A\langle\vec{y}\rangle \xrightarrow{\alpha} P'}\ if\ A(\vec{z}) =_{\text{def}} P$$

$$\underline{REACT} : \frac{P \xrightarrow{\overline{x}\langle\vec{y}\rangle} P' \quad Q \xrightarrow{x(\vec{z})} Q'}{P \mid Q \xrightarrow{\tau} P' \mid \{\vec{y}/\vec{z}\}\, Q'} \qquad *$$

Figure 2.1: The *transition rules* [Mil99].

An example of using the transition rules of this LTS to infer a transition is: Let $P =_{\text{def}} \underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle)$ where: $A_1(y) =_{\text{def}} \overline{y}\langle\rangle.A_2\langle y\rangle$ and $B_1(z) =_{\text{def}} z().B_2\langle z\rangle$. the process P can do the transition $\underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle) \xrightarrow{\tau} \underline{\texttt{new}}\, x\ (A_2\langle x\rangle \mid B_2\langle x\rangle)$, which is a reaction.The inference tree of this transition is shown in Figure 2.2. Thus, using the LTS we can enumerate sll possible transitions of a π-calculus process.

$$\dfrac{\dfrac{\dfrac{}{\overline{x}\langle\rangle.A_2\langle x\rangle \xrightarrow{\overline{x}\langle\rangle} A_2\langle x\rangle}\ \text{by OUT}}{A_1\langle x\rangle \xrightarrow{\overline{x}\langle\rangle} A_2\langle x\rangle}\ \text{by PROCESS CALL} \qquad \dfrac{\dfrac{}{x().B_2\langle x\rangle \xrightarrow{x()} B_2\langle x\rangle}\ \text{by IN}}{B_1\langle x\rangle \xrightarrow{x()} B_2\langle x\rangle}\ \text{by PROCESS CALL}}{A_1\langle x\rangle \mid B_1\langle x\rangle \xrightarrow{\tau} A_2\langle x\rangle \mid B_2\langle x\rangle}\ \text{by REACT}$$
$$\dfrac{A_1\langle x\rangle \mid B_1\langle x\rangle \xrightarrow{\tau} A_2\langle x\rangle \mid B_2\langle x\rangle}{\underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle) \xrightarrow{\tau} \underline{\texttt{new}}\, x\ (A_2\langle x\rangle \mid B_2\langle x\rangle)}\ \text{by RESTRICTION}$$

Figure 2.2: The *inference tree* [Mil99].

2.1.4 Visualization

To gain more understanding of the π-calculus we will use *Stargazer*[Star]. Stargazer is a visual simulator for π-calculus. Figure 2.3 shows the code listing of the process $P =_{\text{def}} \underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle)$ where: $A_1(y) =_{\text{def}} \overline{y}\langle\rangle.A_2\langle y\rangle$ and $B_1(z) =_{\text{def}} z().B_2\langle z\rangle$ in Stargazer syntax.

```
new x. (A1[x] | B1[x])

A1[y] := y<>.A2[y]
B1[z] := z().B2[z]
```

Figure 2.3: Stargazer code for the process P .

Stargazer can visualize the reaction $\underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle) \xrightarrow{\tau} \underline{\texttt{new}}\, x\ (A_2\langle x\rangle \mid B_2\langle x\rangle)$ as shown in Figure 2.4 and Figure 2.5.

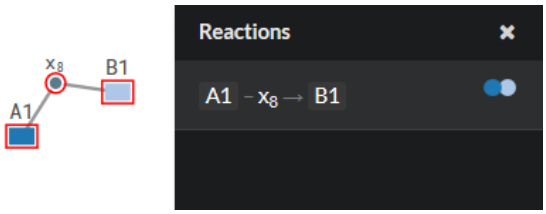


Figure 2.4: The process P before reaction occurrence.

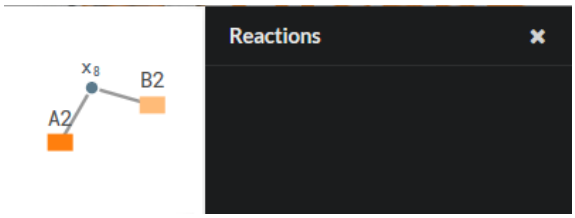


Figure 2.5: The process P after reaction occurrence.

2.1.5 Simulation

For comparing processes based on their behavior there is already a notion called *strong bisimulation* defined in [**?**]..

Definition 2.1.6 (Strong simulation) A relation $\mathcal{S} \subseteq \mathcal{P}^\pi \times \mathcal{P}^\pi$ is called a *strong simulation*, if $(P, Q) \in \mathcal{S}$ implies that

$$\text{if } P \xrightarrow{\alpha} P' \text{ then } Q' \in \mathcal{P}^\pi \text{ exists such that } Q \xrightarrow{\alpha} Q' \text{ and } (P', Q') \in \mathcal{S}. \qquad *$$

Thereby, the internal as well as the external behavior are taken into account. To abstract from the invisible actions, the *weak bisimulation* relates processes only according to their external behavior.

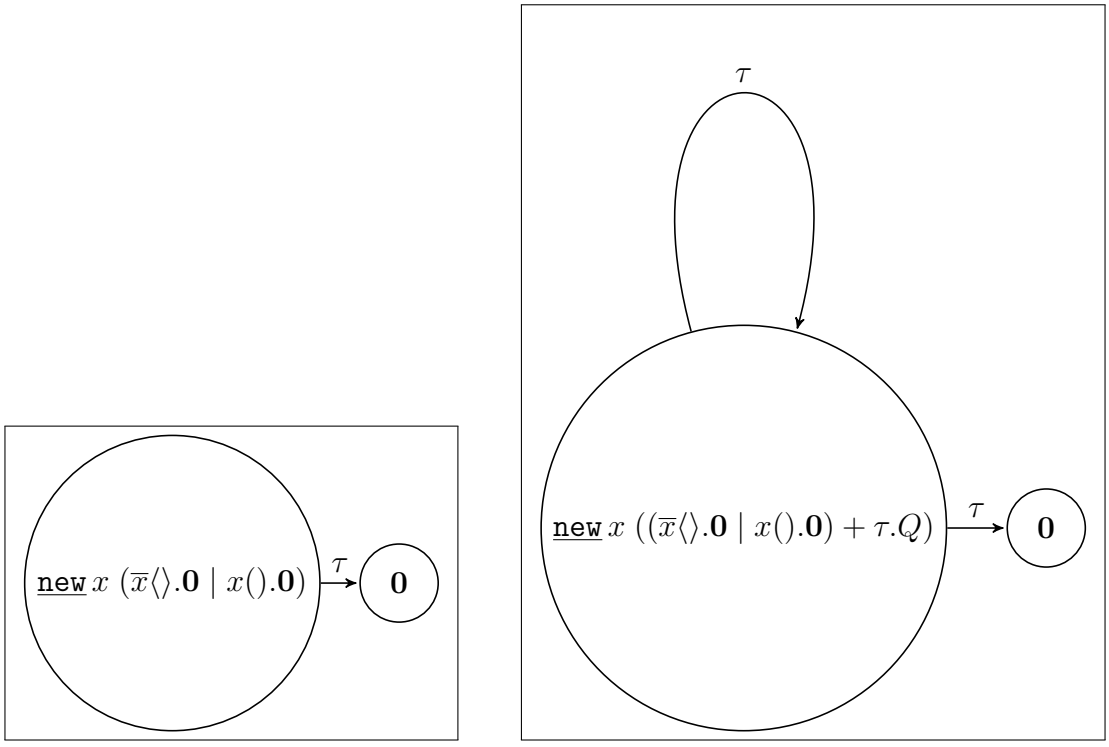
An example of checking the strong simulation is:
Let

- $P =_{\text{def}} \underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle)$
- $Q =_{\text{def}} \underline{\texttt{new}}\, x\ ((\overline{x}\langle\rangle.\mathbf{0} \mid x().\mathbf{0}) + \tau.Q)$

where:

- $A_1(y) =_{\text{def}} \overline{y}\langle\rangle.\mathbf{0}$
- $B_1(z) =_{\text{def}} z().\mathbf{0}$

Intuitively, The behavior of P and Q can be illustrated using transition graphs as shown in Figure 2.6. Q's transition graph is the same as P's, except one thing: Q has a loop with label τ. This loop is due to the τ transition in Q's definition. Hence, we can notice that Q can do all the transitions that P can, plus an extra transition *tau*. In other words Q simulates P, but P doesn't simulate Q.



(a) P (b) Q

Figure 2.6: Transition graphs

To check the strong simulation we will use *ABC (Another Bisimilarity Checker)*[ABC]. ABC is a tool that checks for simulation between π-calculus processes. Figure 2.8 shows the code listing of the process P and Q in ABC syntax.

```
agent P = (^x)( A_1 x | B_1 x)

agent A_1(y) = 'y.0
agent B_1(z) = z.0

agent Q = (^x)((A_1 x | B_1 x) + t.Q)

// check if Q strongly simulates P
lt P Q

// check if P strongly simulates Q
lt Q P
```

Figure 2.7: ABC code for P and Q.

```
Welcome to Another Bisimulation Checker
Opening file examples/viz.abc
Agent P is defined.
Agent A_1 is defined.
Agent B_1 is defined.
Agent Q is defined.
The two agents are strongly related (2).
Do you want to see the core of the simulation (yes/no) ? yes
{
  (
    0
    { }
    0
  )

  (
    (^x0)('x0.0 | x0.0)
    { }
    (^x0)(('x0.0 | x0.0) + t.Q)
  )
}
The two agents are not strongly related (2).
Do you want to see some traces (yes/no) ? yes
traces of

Q
P

-t->
-t->

(^x0)(('x0.0 | x0.0) + t.Q)
0

-t->
-t->

0
*
```

Figure 2.8: ABC code for P and Q.

2.2 The OZ

The OZ

3 Conclusion and future work

In this thesis ...

Bibliography

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, Cambridge, England, 1999.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, Cambridge, England, 2001.

[Star] E. D'Osualdo. *Stargazer: A π-calculus simulator*. http://www.emanueledosualdo.com/stargazer/

[ABC] S. Briais. *Another Bisimilarity Checker*. http://sbriais.free.fr/tools/abc/

Index

T

U

V

W

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Außerdem versichere ich, dass ich die allgemeinen Prinzipien wissenschaftlicher Arbeit und Veröffentlichung, wie sie in den Leitlinien guter wissenschaftlicher Praxis der Carl von Ossietzky Universität Oldenburg festgelegt sind, befolgt habe.

Oldenburg, den 20. Januar 2020

_____

(Muhammad Ekbal Ahmad)