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: 1. März 2020

Contents

List of Figures

1 Introduction

every entity has a behavior and data. behavior actions can have effects on data. to model this idea we will break down our entity into two components: behavior component and data component. behavior component: represent the behavior that can an entity do during it's life cycle. data component: represents the data of an entity and the changes that can be made on it. we use pi calculus which is a specification language to model the behavior component. we use oz which is a specification language to model the data components. since pi and oz are two different languages used to descripe different aspects of entity, we need a way to put them togher to get the model a complete entity. this is done using a simple trick. the trick is: transforming the oz into a pi language. this way we will have : behavior component: in pi. data component: in pi too. this way we can let them play together to represent an entity which have two view: behavior and data.

2 Preliminaries

2.1 The π-calculus

The π-*calculus* is a process algebra that can be used to describe a 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 Section 2.1.5.
Intuitively, 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: " 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*:**

The 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 $\mathbf{0}$ is called the *stop process* or *inaction* and stands for the process that can do nothing. It can be omitted.

***Guard*:**

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

Definition 2.1.2 (Action prefix syntax)

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

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:

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

***Parallel composition*:**

The parallel composition operator | represents 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, i.e., the channel represents a binary synchronization. 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, i.e., in CSP the channel represents a multiple synchronization.

[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"

***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 (Bound names) are all the restricted names in a process. △

Definition 2.1.4 (Free 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 2.1.5 (LTS of π-calculus) The labelled transition system $(\mathcal{P}^\pi, \mathcal{T})$ of π-calculus processes over the action set $\texttt{Act}$ has the process expressions $\mathcal{P}^\pi$ as its

[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.

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.

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

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

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

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

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

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

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 $B_1(z) =_{\text{def}} z().B_2\langle z\rangle$. 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.

$$\dfrac{\dfrac{}{\overline{x}\langle\rangle.A_2\langle x\rangle \xrightarrow{\overline{x}\langle\rangle} A_2\langle x\rangle}\ \text{by OUT}}{A_1\langle x\rangle \xrightarrow{\overline{x}\langle\rangle} A_2\langle x\rangle}\ \text{by PROCESS CALL} \qquad \dfrac{\dfrac{}{x().B_2\langle x\rangle \xrightarrow{x()} B_2\langle x\rangle}\ \text{by IN}}{B_1\langle x\rangle \xrightarrow{x()} B_2\langle x\rangle}\ \text{by PROCESS CALL}$$

$$\dfrac{A_1\langle x\rangle \mid B_1\langle x\rangle \xrightarrow{\tau} A_2\langle x\rangle \mid B_2\langle x\rangle}{\underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle) \xrightarrow{\tau} \underline{\texttt{new}}\, x\ (A_2\langle x\rangle \mid B_2\langle x\rangle)}\ \text{by RESTRICTION}$$

by REACT

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.

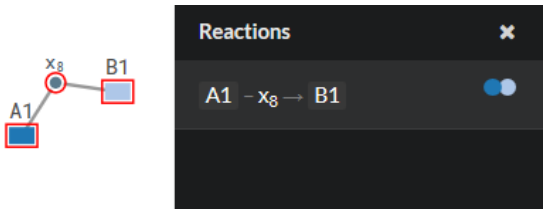


Figure 2.4: The process P before reaction occurrence.

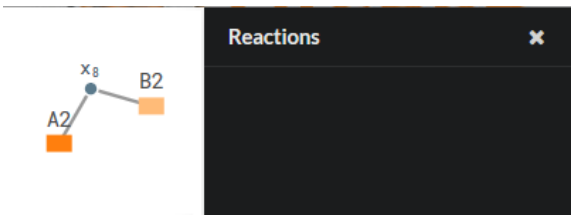


Figure 2.5: The process P after reaction occurrence.

2.1.5 Mobility

As mentioned previously, the word *mobile* refers to the *information sharing and evolution* concepts, since the receiver process can use the received information to change its location. Let us take an example to illustrate the mobility. Let: $\underline{\texttt{new}}\, x, y\ (A\langle x, y\rangle \mid B\langle x\rangle)$ where:

- $A(a, b) =_{\text{def}} \overline{a}\langle b\rangle.A\langle a, b\rangle$
- $B(c) =_{\text{def}} c(d).B\langle d\rangle$

Figure 2.6 shows the stargazer code listing of the process $\underline{\texttt{new}}\, x, y\ (A\langle x, y\rangle \mid B\langle x\rangle)$, Figure 2.7 shows it's visualization before the interaction occurrence, and Figure 2.8 shows it's visualization after the interaction occurrence.

```
new x,y. (A[x,y] | B[x])

A[a,b] := a<b>.A[a,b]
B[c] := c(d).B[d]
```

Figure 2.6: Stargazer code for the process $\underline{\texttt{new}}\, x, y\ (A\langle x, y\rangle \mid B\langle x\rangle)$.

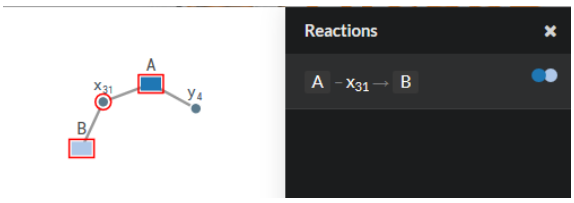


Figure 2.7: Before reaction occurrence.

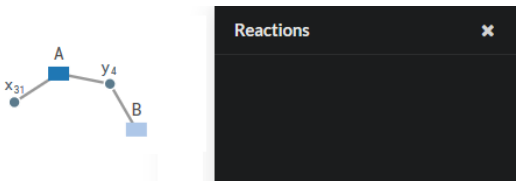


Figure 2.8: After reaction occurrence.

Intuitively, The mobility can be noticed in Figure 2.7 and Figure 2.7, since B changed it's position in the connection topology. The following explains the mobility through interaction step by step:

- Initially the process $A\langle x, y\rangle$ has the channels x, y and the process $B\langle x\rangle$ has the channel x. Thus, $A\langle x, y\rangle$ and $B\langle x\rangle$ are connected via channel x.
- $A\langle x, y\rangle$ has commitment $\overline{x}\langle y\rangle$, i.e., send the channel name y via the channel x.
- $B\langle x\rangle$ has commitment $x(d)$, i.e., receive a message d via x.
- That means: a reaction can occur between $A\langle x, y\rangle$ and $B\langle x\rangle$. This reaction is: $\underline{\texttt{new}}\, x, y\ (\overline{x}\langle y\rangle.A\langle x, y\rangle \mid x(d).B\langle d\rangle) \xrightarrow{\tau} \underline{\texttt{new}}\, x, y\ (A\langle x, y\rangle \mid B\langle y\rangle)$.
- Information sharing: the process $A\langle x, y\rangle$ sends the name y to $B\langle x\rangle$ when the interaction occurs.
- Evolution: when interaction occurs $B\langle x\rangle$ knows about the channel y and uses it as parameter for the the process call $B\langle y\rangle$ .i.e, The $B\langle y\rangle$ now has the channel y, and no more x.
- Finally, in other words:
  - before the reaction: B was connected to A via x as shown in Figure 2.7.
  - after the reaction: B is connected to A via y as shown in Figure 2.8.

2.1.6 Strong simulation

The *strong simulation* is comparison of processes based on their behavior. To understand this let us start with a simple example: Let $P =_{\text{def}} \tau.\tau.\mathbf{0}$ and $Q =_{\text{def}} \tau.\mathbf{0}$. We can notice that P can do two τ transitions, but Q can do only one. Thus Q doesn't strongly simulates P. The word *strongly* refers to the point that: the strong simulation comparison takes the internal transition τ into account. There is another kind of comparison called the *weak simulation*, which doesn't consider the internal transition τ, but this kind of comparison is not considered in this thesis. The formal definition of the *strong simulation* is given in Definition 2.1.6, which is adapted from [Gi14] page 32 with some changes.

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

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

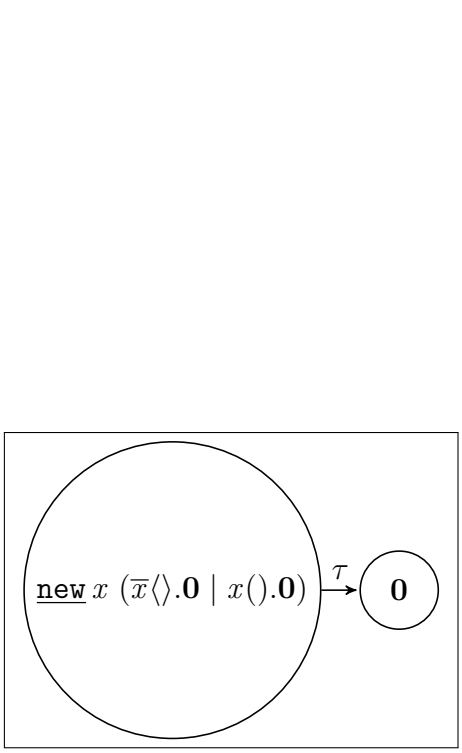
An example of checking the strong simulation is:
Let

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

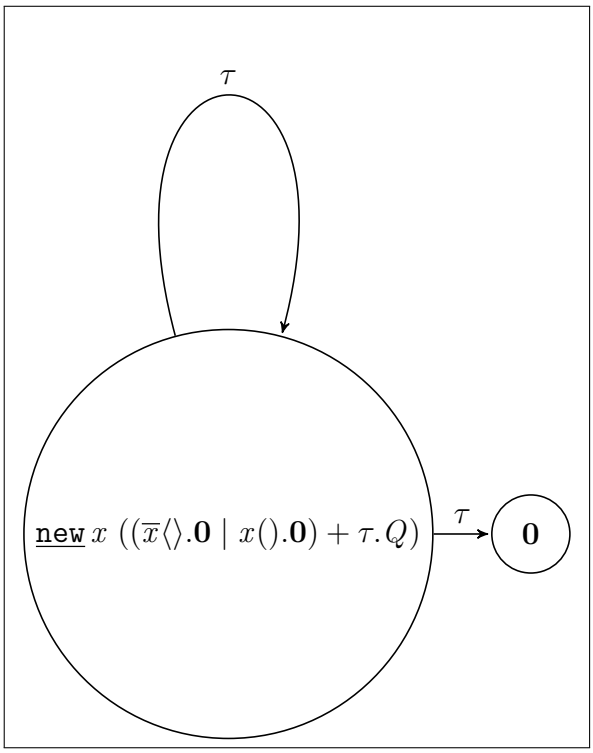
where:

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

Intuitively, The behavior of P and Q can be illustrated using transition graphs as shown in Figure 2.9. 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 τ. In other words Q simulates P, but P doesn't simulate Q.



(a) P



(b) Q

Figure 2.9: Transition graphs

To check the strong simulation we can use *ABC (Another Bisimilarity Checker)* [ABC]. ABC is a tool that checks simulation between π-calculus processes. Figure 2.10 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 2.10: ABC code for P and Q.

Figure 2.11 and Figure 2.12 shows the result of running Figure 2.10, where x0 stands for x, since ABC renames the channels and messages names internally.

In Figure 2.11 we see the result of the command *lt* P Q, which checks if Q strongly simulates P. The result is *yes* and the simulation relation is shown, where x0 stands for x. In Figure 2.11 we see the two pairs of the simulation relation, where:

- (0 { } 0) stands for the pair $(\mathbf{0}, \mathbf{0})$, which means: The state $\mathbf{0}$ of Q is as powerful as $\mathbf{0}$ of P.
- ((^x0)('x0.0 | x0.0) { } (^x0)(('x0.0 | x0.0) + t.Q)) stands for the pair $(\underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle), \underline{\texttt{new}}\, x\ ((A_1\langle x\rangle \mid B_1\langle x\rangle) + \tau.Q))$, which means: The state $\underline{\texttt{new}}\, x\ ((\overline{x}\langle\rangle.\mathbf{0} \mid x().\mathbf{0}) + \tau.Q)$ of Q is as powerful as $\underline{\texttt{new}}\, x\ (\overline{x}\langle\rangle.\mathbf{0} \mid x().\mathbf{0})$ of P.

Thus, Q strongly simulates the behavior of P and the simulation relation is $\mathcal{S} = \{(\mathbf{0}, \mathbf{0}), (\underline{\texttt{new}}\, x\ (A_1\langle x\rangle \mid B_1\langle x\rangle), \underline{\texttt{new}}\, x\ ((A_1\langle x\rangle \mid B_1\langle x\rangle) + \tau.Q))\}$.

```
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)
  )
}
```

Figure 2.11: ABC output: check if Q strongly simulates P.

In Figure 2.12 we can see the result of the command *lt Q P*, which checks if P strongly simulates Q. The result is *no*, since:

- when:
  - Q is in the state $\underline{\texttt{new}}\, x\ ((\overline{x}\langle\rangle.\mathbf{0} \mid x().\mathbf{0}) + \tau.Q)$.
  - P is in the state $\underline{\texttt{new}}\, x\ (\overline{x}\langle\rangle.\mathbf{0} \mid x().\mathbf{0})$.
- then:
  - Q can do a τ transition, which is the loop, to the state $\underline{\texttt{new}}\, x\ ((\overline{x}\langle\rangle.\mathbf{0} \mid x().\mathbf{0}) + \tau.Q)$.
  - P can do a τ transition, which is a reaction, to the state $\mathbf{0}$.
- then:
  - Q can do a τ transition, which is a reaction, to the state $\mathbf{0}$.
  - P cannot go ahead, denoted by " $*$ ", since it is in the state $\mathbf{0}$.

Thus, P doesn't strongly simulates the behavior of 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 2.12: ABC output: check if P strongly simulates Q.

2.2 The Object-Z

The Object-Z ,shortly OZ, is a specifications language used to describe the data part of an entity. This section introduces the Object-Z as decriped in [Ol18].

2.2.1 Intuition

To explain the OZ intuitively, we will start by examining the vending machine example, then we will explain how to build a set mathematically, finally we will explain the main concepts in OZ.

Vending Machine:

As a preperation, let us imagine that we have the task: specifying a vending machine.

- Let cv be the ammount of coffee, and tv be the amount of tea.
- Let *coffee* be the selling coffee operation, and *tea* be the selling tea operation.

the specifications are:

- It sell *coffee* and *tea*, and the maximum amount for each if them is 3.
- It's initial state is $cv = 3$ and $tv = 3$.
- When the operation *coffee* or *tea*,then the amount should be decreased by one.

Those specifications describe the data of the vending machine and the changes on them. Thus, the state space of the vending machine can be visualized as shown in Figure 2.13, where we see the initial state $VM(3, 3)$ and a state transition to $VM(2, 3)$. Later in **Main concepts of OZ** we will learn how to write the specifications using OZ language notations.

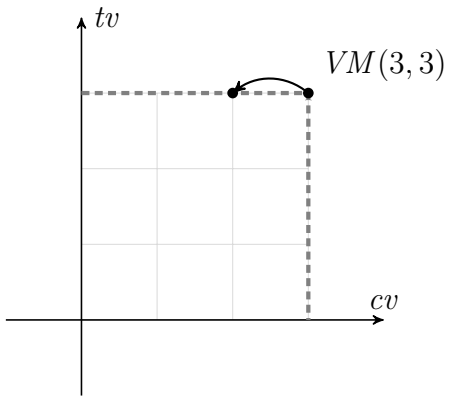


Figure 2.13: State Space.

***Set building*:**

A Set is a collection of things. For example: $\{5, 7, 11\}$ is a set. But we can also build a set by describing what is in it. Here is a simple example of building a set: $\{x \mid (x \in \mathbb{Z}) \land (x \geq 0)\}$, it says: *the set of all x's, such that x is integer and greater than 0.* The last example can be written in a nicer way as follow: $\{x : \mathbb{Z} \mid x \geq 0\}$. T have a more control about the elements in a set, we can write: $\{x : \mathbb{Z} \mid x \geq 0 \bullet x\}$,it says the same as the last example. The *expretion* after $\bullet$ decides how should be the elements of the set be transformed. another example is: $\{x : \mathbb{Z} \mid x \geq 0 \bullet x^2\}$,it says: *the set of all squared x's, such that x is integer and greater than 0.* Thus, to build a set: we can use the following notation: $\{Deklaration \mid predicate \bullet expression\}$.

***Main concepts of OZ*:**

The main concepts of OZ are:

- *Schema*: It can been seen as a set [SIJ88].
- *Class*: It can been seen as a grouping of a *state schema*, *initial state schema* and *operation schemas* [TDC04]. It represents the object oriented approach

To illustrate those main concepts, consider the vending machine example,denoted by VM:

- *Class*: To model the vending machine we need to define a class VM. Syntactically, in OZ a class definition is a named box as shown in Figure 2.14.

Figure 2.14: VM *class.*

- *State space*: The state space of our vending machine can be seen as a set of all valid states. The set of all valid states is:
  - In Mathematics: $State\_Space = \{cv, tv : \mathbb{Z} \mid (0 \leq cv \leq 3) \land (0 \leq tv \leq 3) \bullet (cv, tv)\} = \{(0,0), \ldots, (3,3)\}$.
  - In OZ: The set *State_Space* can be described using a *state schema*, which is a box without name added to the class box as shown in Figure 2.15.

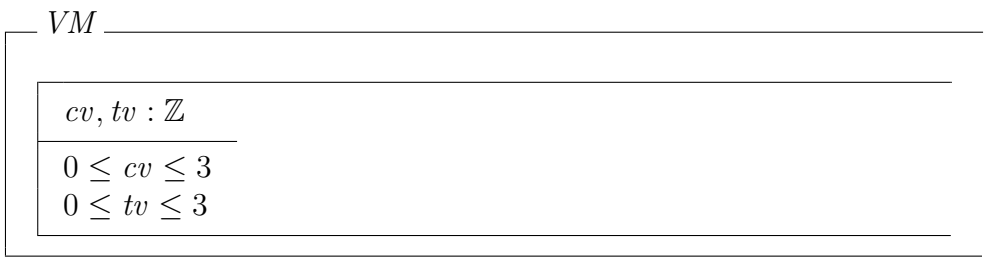


Figure 2.15: *VM* class: adding the *state schema.*

- *Initial state*: Our vending machine has an initial state with $cv = 3$ and $tv = 3$. The set of all possible initial states, that respects those conditions is:
  - In Mathematics: $Initial\_States = \{cv, tv : \mathbb{Z} \mid (0 \leq cv \leq 3) \land (0 \leq tv \leq 3) \land (cv = 3) \land (tv = 3) \bullet (cv, tv)\} = \{(3,3)\}$.
  - In OZ: the set *Initial_States* can be described using a *initial state schema*, which is a box named *INIT* added to the class box as shown in Figure 2.16.

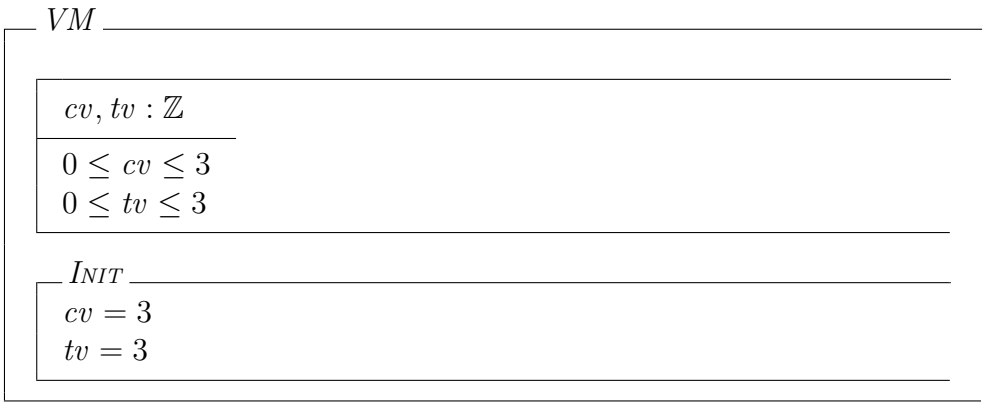


Figure 2.16: *VM* class: adding the *initial state schema*.

- *State transition*: When the vending machine sells a coffee, the amount of coffee should be decreased by one. This is a state transition. the set of all possible state transitions when the selling coffee operation occurs is:
  - In Mathematics: $coffee = \{cv, tv, cv', tv' : \mathbb{Z} \mid (0 \leq cv \leq 3) \land (0 \leq tv \leq 3) \land (0 \leq cv' \leq 3) \land (0 \leq tv' \leq 3) \land (tv' = tv) \land (cv' = cv - 1) \bullet ((cv, tv), (cv', tv'))\} = \{((3,3),(2,3)), \ldots, ((1,0),(0,0))\}$, where (cv, tv) represents the *pre state* and (cv', tv') represents the *post state* of a state transition.
  - In OZ: the set *coffee* can be described using an *operation schema*, which is a box named with the operation name added to the class box as shown in Figure 2.17.

VM

$$cv, tv : \mathbb{Z}$$

$$0 \leq cv \leq 3$$
$$0 \leq tv \leq 3$$

Init

$$cv = 3$$
$$tv = 3$$

coffee

$$cv, tv, cv', tv' : \mathbb{Z}$$

$$0 \leq cv \leq 3$$
$$0 \leq tv \leq 3$$
$$0 \leq cv' \leq 3$$
$$0 \leq tv' \leq 3$$
$$tv' = tv$$
$$cv' = cv - 1$$

Figure 2.17: *VM* class: adding the *operation schema*.

OZ offers a more nice way to write the operation schema using Δ-list. In OZ:

- Operation schema has a Δ-list of state variables whose values may change. By convention, no Δ-list means no attribute changes value.
- Operation schema implicitly includes the state schema and a primed version of it.

Thus, since the schema operation *coffee* specifies changes on the *coffee* value only, we can write it as shown in Figure 2.18.

VM

$cv, tv : \mathbb{Z}$

$0 \leq cv \leq 3$
$0 \leq tv \leq 3$

Init

$cv = 3$
$tv = 3$

coffee

$\Delta(cv)$

$cv' = cv - 1$

Figure 2.18: *VM* class: *operation schema using delta operator*.

Finally, we can model the *tea* operation schema as we did with *coffee* to get the class *VM* as shown in Figure 2.19, which represents all the specification we had of the vending machine written in OZ language.

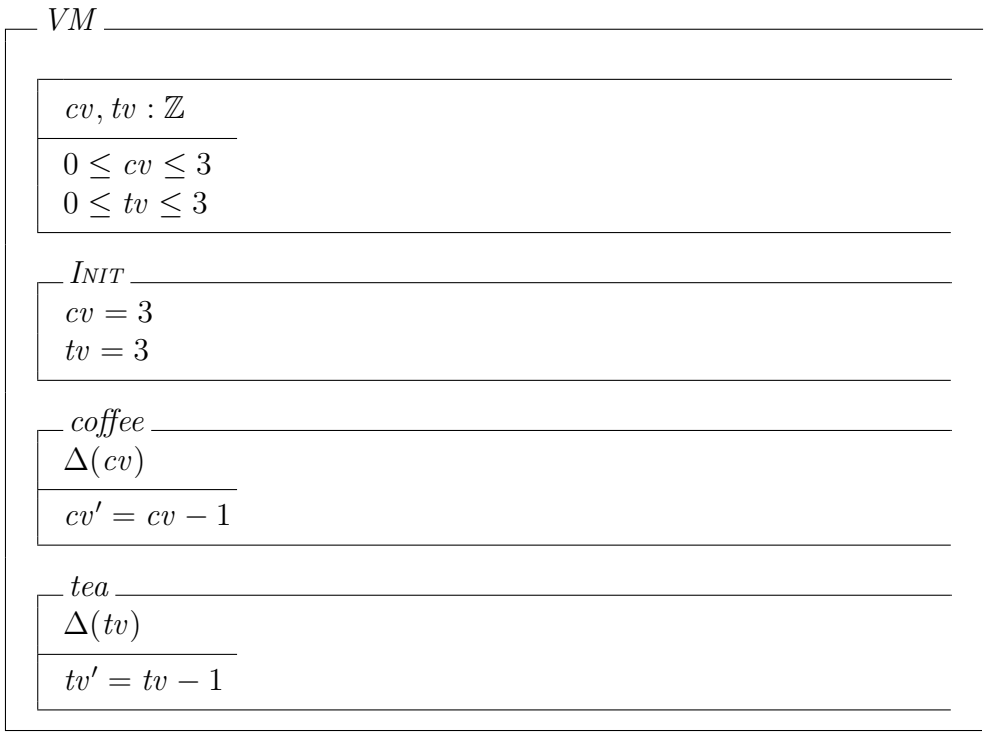


Figure 2.19: *VM* class: *with all specifications.*

***Operation's input and output parameters*:**

Some operations can have input and output parameters, just like method in programming language, where the method's parameters represent the input, and the returned values represent the output. To illustrate the idea let us extend our vending machine .The new *VM* can talk to a shop sending a message to it. So it has a new operation *talk* and a state variable m representing the message to be be sent.

The set of all possible state transitions when the *talk* operation occurs is:

- In Mathematics: $talk = \{cv, tv, m, cv', tv', m', x : \mathbb{Z} \mid (0 \leq cv \leq 3) \land (0 \leq tv \leq 3) \land (0 \leq cv' \leq 3) \land (0 \leq tv' \leq 3) \land (tv' = tv) \land (cv' = cv) \land (m' = m) \land (x = m) \bullet ((cv, tv, m), (cv', tv', m'))\} = \{((3,3,1),(3,3,1)), \ldots, ((0,0,1),(0,0,1))\}$.
- In OZ: the set *talk* can be described using an *operation schema*, as shown in Figure 2.20. We can notice that the operation doesn't change any state variable's value, it just says that the value of the output parameter x, written as $x!$, must be equal to the value of the state variable m. For input parameter use ? symbol.

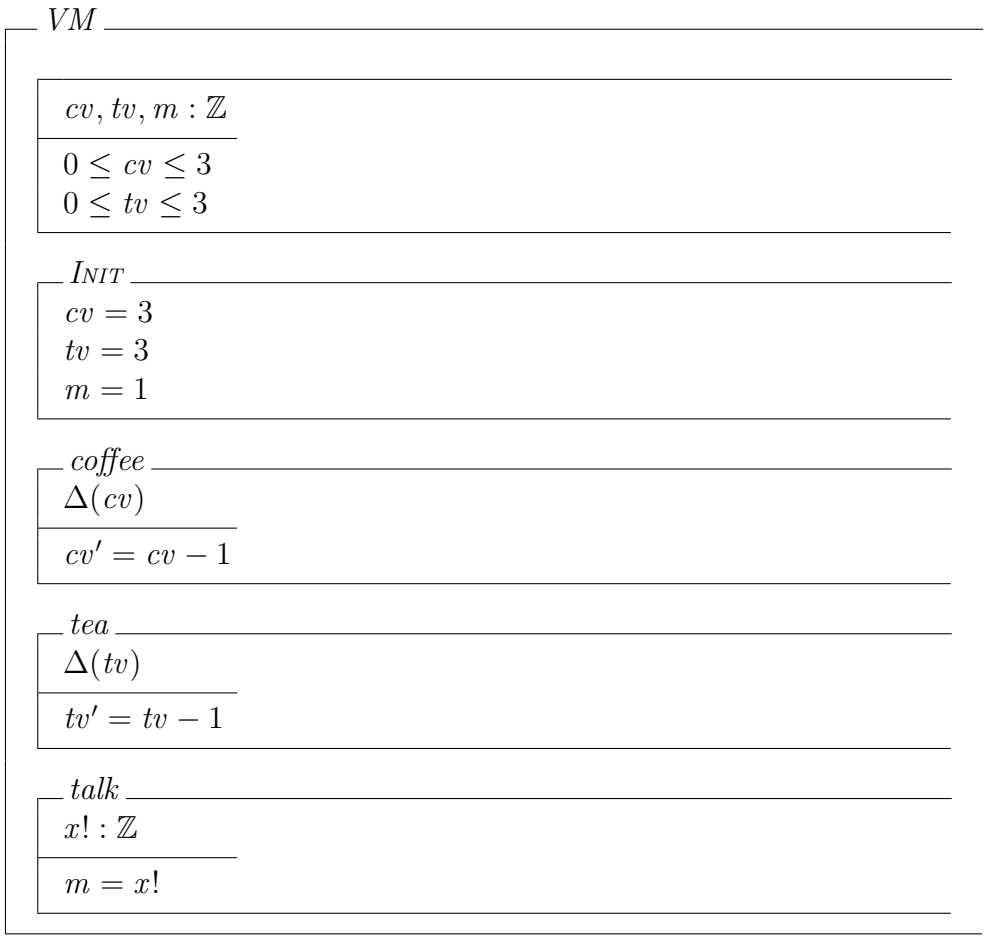


Figure 2.20: *VM* class: *talk operation with output parameter.*

***Instance reference*:**

Since OZ is an object oriented approach, every instance of a class needs a reference to refer to it. In OZ the reference can be seen simply as sate constant. Furthermore, operations can can share its instance identity through output or input the reference name *self* as shown in Figure 2.21 in the operation *talk*.

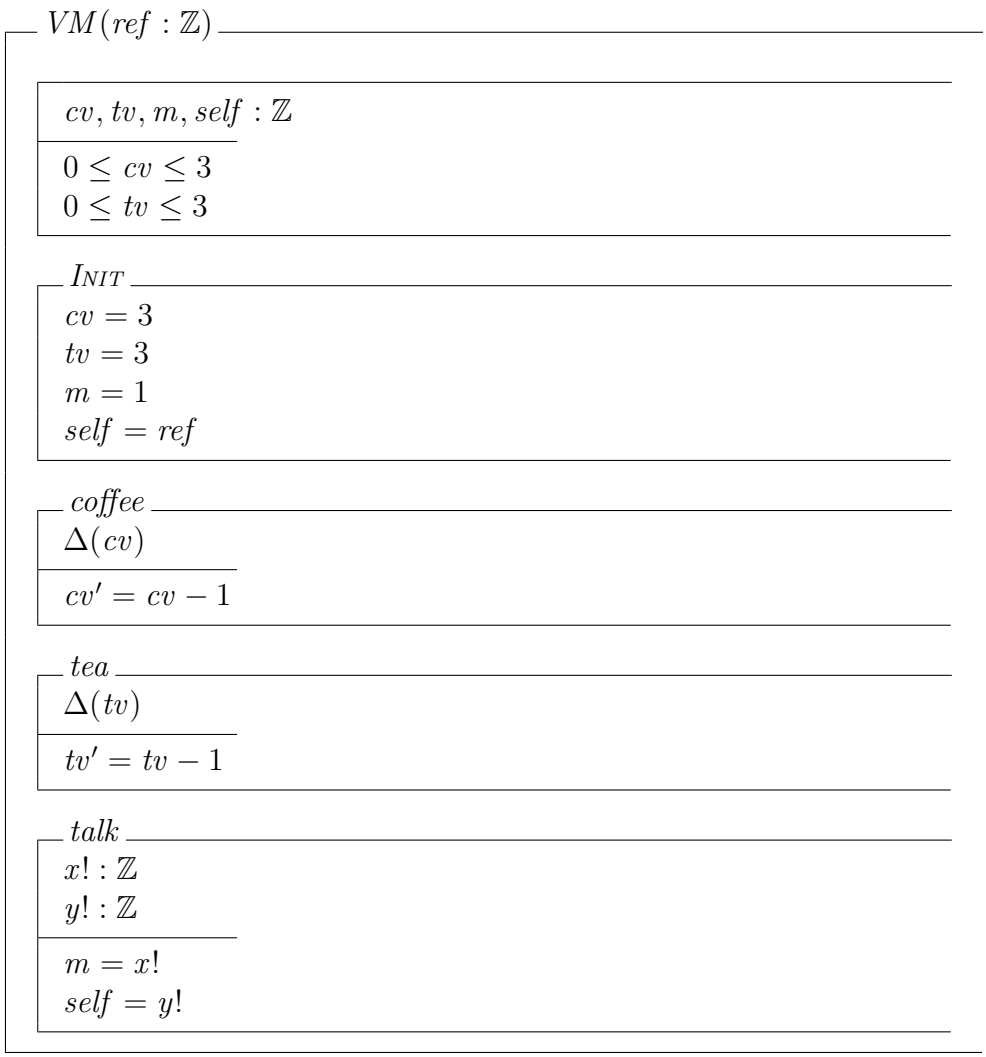


Figure 2.21: *VM* class: *instance reference.*

2.2.2 Semantics

To understand the operational semantics of OZ we will use a labelled transition system LTS. Using this LTS we can investigate the state evolution of a OZ object . The definition of this LTS is adapted from Definition 2.1.5 with some changes.

Definition 2.2.1 (LTS of OZ) The labelled transition system $(\mathcal{S}^{OZ}, \mathcal{T})$ of OZ states over the set of operations, has the valid states $\mathcal{S}^{OZ}$ as it's states, and it's transitions $\mathcal{T}$ are those which can be inferred from the rule in Figure 2.22. The rule *OPER* shows the state evolution .i.e, state transition.

$$\underline{OPER} : PRE\_STATE \xrightarrow{operation} POST\_STATE \qquad \triangle$$

Figure 2.22: The *OZ transition rules*

An example of using the transition rule of this LTS is: drawing the transition graph of vending machine shown in Figure 2.19. The transition graph is shown in Figure 2.23, where we show only a small part of it. Thus, using the LTS we can enumerate all possible states transitions of an OZ state.

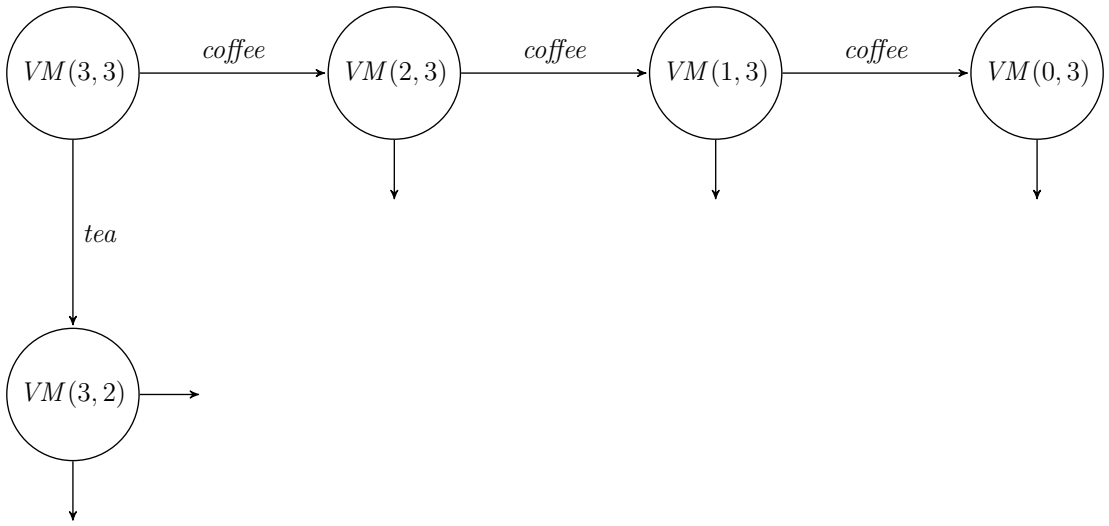
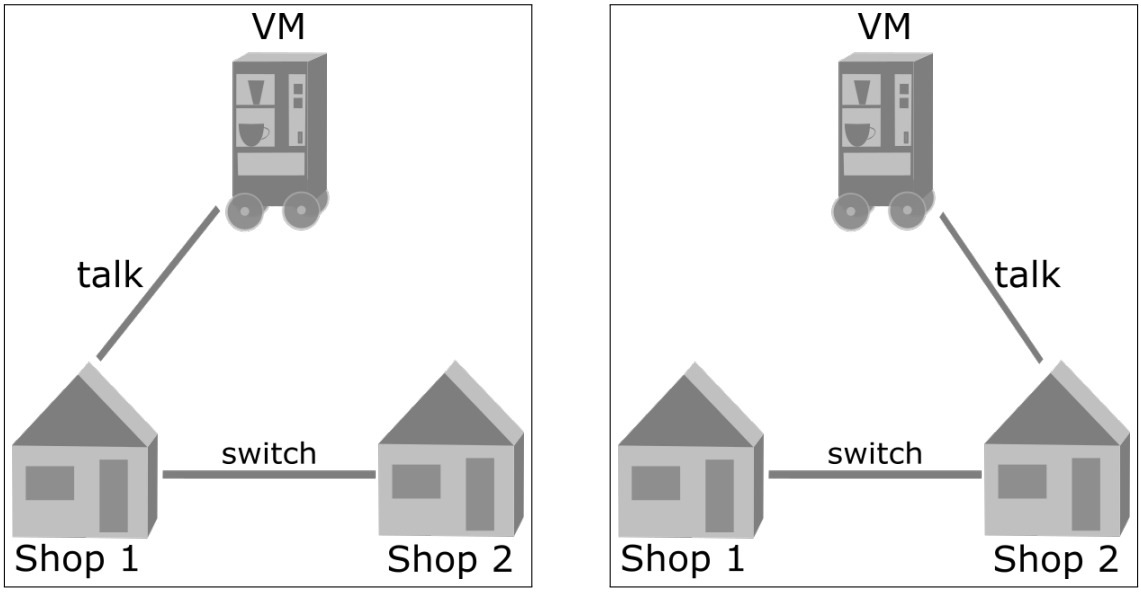


Figure 2.23: *VM Transition graph*

2.2.3 Unfixed operation

OZ can be used to model fixed communication typologies. Let us illustrate the problem using the example shown in Figure 2.24. the vending machina can communicate with a shop through a channel *talk*. so the *VM* can invoke the operation



(a) Before switch

(b) After switch

Figure 2.24: Unfixed operation

method overloading

Shop($ref : \mathbb{Z}$)

$m, vmId, self : \mathbb{Z}$
$t : chan[\mathbb{Z} \times \mathbb{Z}]$

Init

$self = ref$

switch

$\Delta(t)$
$z? : \mathbb{Z}$

$t' = z?$

switch

$z! : \mathbb{Z}$

$t = z!$

switch

$x! : \mathbb{Z}$

$t \neq nil$
$t = x!$

$\Delta(t)$
$x? : \mathbb{Z}$

$t' = x?$

talk

$\Delta(m, vmId)$
$y? : \mathbb{Z}$
$z? : \mathbb{Z}$

$m' = y?$
$vmId' = z?$

Figure 2.25: *VM* class: *instance reference.*

3 Transformational semantics of OZ

3.1 Mapping values

mapping

3.2 Mapping Data Types

mapping

3.3 Mapping state variables

mapping

3.4 Mapping class

mapping

3.5 Mapping state schema

mapping

3.6 Mapping initial state schema

mapping

3.7 Mapping operation schema

mapping

4 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.

[Gi14] M. Gieseking. *Refinement of π-calculus processes*. Master thesis, Carl von Ossietzky Universität Oldenburg, 2014.

[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/

[Ol18] E. Olderog. *Kombination von Spezifikationstechniken*. Vorlesungsskript, Carl von Ossietzky Universität Oldenburg, 2018/19.

[SIJ88] S. King and I. Sorensen and J. woodcock. *Z: GRAMMAR AND CONCRETE AND ABSTRACT SYNTAXES (Version 2.0)*. Oxford University Computing Laboratory, England, 1988.

[TDC04] T. Kenji and D. Jin and C. Gabriel. *Relating π-calculus to Object-Z*. Engineering of Complex Computer Systems, IEEE International Conference on 97-106, 2004.

Index

R

S

T

U

V

W

Acknoldegment

Special thanks goes to:

- Prof. Dr. Ernst-Rüdiger Olderog for suggesting the topic of the thesis and for the continuous support.
- M.Sc Manuel Gieseking for providing the latex template.
- Dr. Emanuele D'Osualdo for support with π-calculus.

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 1. März 2020

_____

(Muhammad Ekbal Ahmad)