CARL VON OSSIETZKY universität OLDENBURG

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

Refinement of π-calculus processes

Masterarbeit

Name:	Manuel Gieseking
E-Mail:	manuel.gieseking@informatik.uni-oldenburg.de

Studiengang:	Fach-Master Informatik
Erstgutachter:	Prof. Dr. Ernst-Rüdiger Olderog
Zweitgutachter:	Dipl.-Inform. Sven Linker
Datum:	29. März 2014

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.

For concurrently calculating the decomposed tasks, these tasks have to be distributed to the participants. Moreover, the members have to communicate with each other, for example, for sending interrupts and sharing results. Generally, one can say, the more use of concurrency the more communication has to be done. Hence, concurrently communicating systems are rather the norm than an exception. Thus, investigating concurrency and communication are a continuing concern within modern computing.

For the formal specification of such systems *process algebras* have been widely accepted, especially Tony Hoare's Communicating Sequential Processes (CSP) [Hoa78] and Robin Milner's Calculus of Communicating Systems (CCS) [Mil80]. Thereby, the components of the system – for example, clients and servers – are called processes and are described as algebraic structures. One limitation of the expressiveness of the process algebras mentioned above is the static definition of the processes. A structural modification of the process during a computation is not possible. This is taken into account in the π-calculus defined by Robin Milner, Joachim Parrow, and David Walker in [MPW92].

The π-calculus is an extension of CCS and adds the facility of expressing changes in the process structure inspired by the work of Uffe Engberg and Mogens Nielsen in [EN86]. This modification of the process structure is called *mobility*. Thus, with the π-calculus we can describe and investigate concurrently communicating mobile processes. In the π-calculus mobility, can be understood as modifications of the

linking structure of the processes.

A standard example of mobility in the π-calculus is the connection of mobile phones to base stations. Consider, for example, two people P_1 and P_2 carrying mobile phones. At the beginning of the setting both are separately connected to two different static base stations S_1 and S_2. Thereby, P_1 is connected to S_1 and P_2 is connected to the base station S_2. Then, we can think of two possible forms of movement. On the one hand there is the *physical mobility*. That is, a person has changed its location within the real word, which is not within the focus of the π-calculus. On the other hand, the communication links between the base stations and the people has changed, called *virtual mobility*. Consider, for example, P_1 is leaving the reception area of S_1 while reaching the area of S_2. Then a new communication link between P_1 and S_2 is established and the contact to S_1 is disconnected. Thus, P_1 and P_2 are now both connected to S_2 and none of them to S_1. Hence, the connections between the people and the base stations at the beginning of the setting are significantly different to the linking structure at the end. The π-calculus treats the virtual mobility by passing channels within a communication.

Even though many variations and modifications of Milner's original π-calculus exist and it has been investigated in many ways, the general focus is mainly on bisimulations between processes to determine their equivalence [PS96]. That is, observing their behaviour and determine whether they behave identically. Thus, proving correctness requires a symmetric equivalence relation with the program and the specification within. Apart from that, it would be interesting to investigate the process' behavior like it is done in CSP with its set-theoretical denotation for the investigation whether a process satisfies its specification. That is, we define a condition (the specification) and check if another process (the program) meets this specification by comparing their behavior collected in sets. To reach this, the sequences of events (*traces*) are collected in a set and the correctness is proved by set inclusion. Thereby, there is no need for a symmetry. This approach is often called *refinement*. Even though there is also a relation called simulation which does not need to be symmetrical, "[...] the time has come to unify the two modelling styles [(CSP and CCS)], to enable practicing engineers to exploit a combination of their complementary advantages"[1] like Tony Hoare wrote in 2006. This statement is also applicable to CSP and the π-calculus.

Hence, the contribution of this thesis is to develop a notion of refinement for

[1] [Hoa06], page 209.

π-calculus processes similar to the approach for CSP in [Ros98] to achieve the advantages of this modeling style. Thereby, we primarily concentrate on the *trace semantics* and neglect the *failures* and *failures-divergence semantics*. Furthermore, we investigate the properties of this semantics to show its usability and point out its limitations.

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, a new structural form of processes – the *extended standard form* – is introduced and it is shown that every process can be transformed into a structural congruent one in this standard form. We proceed in Chapter 3 by defining a big-steps semantics which bases on the early transition system by observing an arbitrary number of steps within the early semantics. Subsequently, in Chapter 4, we introduce the trace semantics which is based on the big-steps semantics of the previous chapter. Thereby, we investigate its properties and define the refinement based on the trace semantics. Finally, the conclusion in Chapter 5 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, Mey09, CC03].

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 Sequences

In this section we recall the definition of sequences and some of its properties. This well-known information can be found in standard textbooks.

Definition 2.1.1 (Sequence) A *sequence* is a partial function which maps natural numbers to values of a given set. We define the set of all finite sequences for a given set X by

$$\mathtt{seq}(X) =_{\text{def}} \{f : \mathbb{N} \to X \mid \exists n \in \mathbb{N} : \mathtt{dom}(f) = \{1, \ldots, n\} \vee \mathtt{dom}(f) = \emptyset\},$$

with $\mathtt{dom}(f)$ denotes the domain of a function f.

The collection of all sequences we denote by $\mathtt{seqs}$. As an abbreviation we write $\langle a_1, \ldots, a_n \rangle$ for the function $f : \mathbb{N} \to \{a_1, \ldots, a_n\}$ with $i \mapsto a_i$ for all $i \in \{1, \ldots, n\}$. Furthermore, we also use $\alpha \in s$ instead of $(\cdot, \alpha) \in s$, for a sequence $s \in \mathtt{seq}(X)$ and $\alpha \in X$.

The special case that $\mathtt{dom}(f) = \emptyset$ holds, so no element is mapped for a function f, is called an *empty sequence* and we denote it by $\langle\rangle$. The collection of all sequences apart from the empty sequence is denoted by $\mathtt{seqs}_1$. Similarly, $\mathtt{seq}_1(X)$ describes all sequences over a given set X except of the empty sequence. Furthermore, we often use the index notation for a sequence $s = \langle a_1, \ldots, a_n \rangle$ to address the single elements: $s_i = s(i)$. $*$

To ease the usage of sequences, we list some operations handling them and their properties.

Definition 2.1.2 (Operations on sequences) We define the *length of a sequence* by the cardinality of its domain:

$$\# : \begin{cases} \mathtt{seqs} & \to \mathbb{N} \\ s & \mapsto |\mathtt{dom}(s)| \,. \end{cases} \tag{length}$$

For the *concatenation* of two sequences we use the $\frown$ operator, defined by

$$\frown : \begin{cases} \mathtt{seqs} \times \mathtt{seqs} & \to \mathtt{seqs} \\ (s,t) & \mapsto s \cup \{(i + \#(s), x) \mid (i,x) \in t\} \,. \end{cases} \tag{concatenation}$$

The *iteration* of a sequence is inductively defined by

$$\begin{array}{lll} s^0 & =_{\text{def}} & \langle\rangle \\ s^{n+1} & =_{\text{def}} & s^n \frown s. \end{array} \tag{iteration}$$

We lift both, the concatenation for two sets $S, T \subseteq \mathtt{seqs}$ by

$$S \frown T =_{\text{def}} \{s \frown t \mid s \in S, t \in T\} \tag{concatenation}$$

and the iteration by

$$\begin{array}{lll} S^0 & =_{\text{def}} & \{\langle\rangle\} \\ S^{n+1} & =_{\text{def}} & S^n \frown S \end{array} \tag{iteration}$$

to sets of sequences.

Furthermore, we define the *Kleene star* as $S^* =_{\text{def}} \bigcup_{n \in \mathbb{N}} S^n$, as well as the *prefix operator*:

$$\mathtt{pref}(S) =_{\text{def}} \{t \in \mathtt{seqs} \mid \exists s \in S, u \in \mathtt{seqs} : s = t \frown u\} \tag{prefix}$$

for a set $S \subseteq \texttt{seqs}$. $*$

With the definition of the concatenation of sets of sequences, we get that the union and the concatenation of sets of sequences are distributive.

Lemma 2.1.1 (Distributivity) *Given $S, T, R \in \texttt{seqs}$, then*

$$\begin{aligned}(S \cup T)^\frown R &= (S^\frown R) \cup (T^\frown R)\\ S^\frown (T \cup R) &= (S^\frown T) \cup (S^\frown R)\end{aligned}$$

holds. $\diamondsuit$

The proof follow directly from the definition of the concatenation and union of sets of sequences.

2.2 The π-calculus

The *π-calculus* belongs to the family of *process algebras* and is used to model communicating and concurrent systems of mobile processes with changing structure. The roots of process algebras can be found in Tony Hoare's CSP from the late 70's and Milner's CCS invented around 1980 [Mil80]. Moreover, the π-calculus can be seen as a continuation and extension of CCS and is inspired by the work of Engberg and Nielsen in [EN86], where they extended CCS with mobility [MPW92].

The first published paper treating the π-calculus is called "A calculus of mobile processes" [MPW92] written by Robin Milner, Joachim Parrow, and David Walker in the late 80's [Mil99]. While in CCS only messages can be sent over a channel during a communication, in this paper they developed the capability to send a channel over a channel. In particular, messages and channels have the same type.

Hence, the main advantage of the π-calculus is its capability to send names over a channel during a communication which can later be used as channels themselves. This provides the ability to change the connection structure of a system.

In this thesis we choose a π-calculus variant which uses parameterized recursion instead of the replication operator. Thereby, we have no loss in expressiveness, since there is a transformation from recursion to replication and vice versa [Mil99, SW01]. But since the definition of the examples in the topic of computer science with recursion appeared to be more intuitive to us and recursion is present in CCS, we decided to use a variant with recursion rather than replication.

Furthermore, we define a variant without the match prefix, since "[m]atch, and especially mismatch, prefixes are seldom useful for describing systems"[1], but inflating the proofs and definitions.

Apart from that, the definition of the syntax as well as the definition of the semantics and the treatment of the π-calculus refers to [SW01]. Especially the *guarded choice* – which means every process in a choice has to perform a prefix action first – differs from Milner's, Parrow's and Walker's syntax in [MPW92]. This restriction is well-accepted in literature [Mey09], since non-guarded choice "is considered of minor practical importance"[2] and "complicates the theory"[3]. However, using guarded choice "does not delimit the computational expressiveness of the calculus"[4]. We slightly differ from the standard definition of the guarded choice, since for the transformation of a process to our extended standard form we need to allow the existence of some intermediate processes. But this adaption is of minor influence for the remaining theory.

Finally, we use the *monadic* π-calculus – meaning just a single name can be sent (respectively received) by an output (respectively input) action –, since Milner also gave a transformation from the *polyadic* to the monadic version, which proves their equal expressiveness [Mil99] and the monadic variant simplifies the handling of processes in this thesis.

2.2.1 Syntax

Let $\mathcal{N}$ be the countably infinite set of *names*, with typical representatives as lower case letters like $a, b, c, x, y, z, \ldots \in \mathcal{N}$ and let $\overline{\mathcal{N}} =_{\text{def}} \{\overline{a} \mid a \in \mathcal{N}\}$ be the set of *co-names*. In communications the names of $\mathcal{N}$ are used as *channels* as well as as *messages / objects*.

With the help of these names we can define *prefixes* of π-calculus processes. The possible prefixes π of a π-calculus process are

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

with $x, y \in \mathcal{N}$. The prefix $\overline{x}\langle y\rangle$ stands for the *output prefix*, which means the process can send the name y over the channel x. For receiving a value y along the channel x,

[1] [SW01], page 13.
[2] [Mey09], page 15.
[3] [SW01], page 13.
[4] [Mey09], page 15.

the *input prefix* $x(y)$ is used. After receiving, every occurrence of y in the remaining process is replaced by the received name. The third prefix is called the *silent prefix* and means that the process performs an internal step.

Processes do not simply consist of prefixes, they also can for example be composed in parallel or appear in an alternative. The full syntax of π-calculus processes is given in Definition 2.2.1. To define a recursion with parameters, there is the need for *process identifiers* typically denoted by upper case letters $A, B, C, \ldots$ and an abbreviation for a finite list of names. Thus, we can write $\vec{a}$ for a parameter list $a_1, a_2, \ldots, a_n$ for some $n \in \mathbb{N} \setminus \{0\}$ and $a_1, \ldots, a_n \in \mathcal{N}$. Additionally, we treat $\vec{a}$ as a set $\{a_1, a_2, \ldots, a_n\}$, whenever it is needed and no confusion arises. Moreover, in the *recursive definition* $A(\vec{w}) =_{\text{def}} P$ with the process identifier A and a process P, the elements of $\vec{w}$ must be pairwise distinct.

Definition 2.2.1 (Syntax) The syntax of π-calculus *processes* – typically denoted by $P, Q, R, \ldots$ – is defined inductively:

$$M ::= \mathbf{0} \;\Big|\; \pi.P \;\Big|\; M_1 + M_2 \;\Big|\; \underline{\texttt{new}}\, z\; M$$
$$P ::= M \;\Big|\; P_1 \mid P_2 \;\Big|\; \underline{\texttt{new}}\, z\; P \;\Big|\; A\langle\vec{v}\rangle.$$

The set of all π-calculus processes is denoted by $\mathcal{P}^\pi$. We call M a *summation* or *sum* and the set of all summations is denoted by $\mathcal{P}^\pi_M$. We further assume that there is a recursive definition $A(\vec{w}) =_{\text{def}} Q$ for every $A\langle\vec{v}\rangle$ and define that every process itself relies on a finite number of process identifiers. ∗

To gain an intuition of the intended interpretation of the syntax, at first we give a brief informal description of the behavior of the possible types of processes, before we define an operational semantics in Section 2.2.4.

The process $\mathbf{0}$ is called the *stop process* or *inaction* and stands for the process without behavior. Often we will – as a convention – omit a pending $\mathbf{0}$ and accordingly write, for example, $\overline{x}\langle y\rangle \mid (x(z) + \tau)$ instead of $\overline{x}\langle y\rangle.\mathbf{0} \mid (x(z).\mathbf{0} + \tau.\mathbf{0})$.

Let π be one of the three defined prefixes and $P \in \mathcal{P}^\pi$, then we call $\pi.P$ a *prefix process*. The prefixes represent the communication possibilities of a process. As described we can send or receive values over a given channel or perform an internal step. From the semantical view we call it an *action*, if such a prefix is performed. Every action resulting from handling a prefix apart of the silent prefix will be called an *observable action*, since this behavior is visible to the environment. Thus, a process $P' =_{\text{def}} \overline{x}\langle y\rangle.P$ has to perform an *output action* $\overline{x}\langle y\rangle$ before it behaves like

P. Furthermore, we call x the *subject* and y the *object* or *parameter* of a prefix $\overline{x}\langle y\rangle$ or $x(y)$, as well as of the corresponding actions. The third prefix performs a *silent action* or – in contrast to the other prefixes – an *unobservable action* and is used to describe an *internal action*. Thus, $\pi.P$ performs the π prefix and after that it behaves like P or in case that π is an input prefix it behaves like a process P', where every occurrence of the object of the prefix is replaced by the received name. For example the process $a(x).\overline{x}\langle y\rangle.\mathbf{0}$ can evolve with the visible *input action* $a\,b$ to the process $\overline{b}\langle y\rangle.\mathbf{0}$. Depending on the prefix π we also call a process $\pi.P$ an *input process*, *output process* or τ *process*.

A *choice process*, or simply *choice*, is written by $M_1 + M_2$ with $M_1, M_2 \in \mathcal{P}^{\pi}_{M}$. From the definition we know that every participant of the choice must either be an inaction, a prefix or a sum with a restriction. Since a sum with a restriction is also possible within a choice, this variant is a slightly adapted version of a *guarded choice*. A choice process $M_1 + M_2$ with $M_1, M_2 \in \mathcal{P}^{\pi}_{M}$ has the possibility to behave like the process M_1 or like M_2. But if one process is chosen, the behavior of the other is no longer taken into account. Thus, for instance, the process $\overline{a}\langle b\rangle.\overline{b}\langle c\rangle.\mathbf{0} + \overline{x}\langle y\rangle.\overline{y}\langle z\rangle.\mathbf{0}$ can evolve to the process $\overline{b}\langle c\rangle.\mathbf{0}$ or to $\overline{y}\langle z\rangle.\mathbf{0}$.

For a *parallel composition* $P_1 \mid P_2$ with $P_1, P_2 \in \mathcal{P}^{\pi}$ the behavior is an interleaving of the behavior of the process P_1 and P_2, with the added possibility of a communication between the two parts over a common channel. Consider, for example, $P_1 =_{\text{def}} x(y).\overline{y}\langle z\rangle$ and $P_2 =_{\text{def}} \overline{x}\langle a\rangle.\overline{b}\langle b\rangle$, then $P =_{\text{def}} P_1 \mid P_2$ can evolve invisibly to $\overline{a}\langle z\rangle \mid \overline{b}\langle b\rangle$, since P_1 and P_2 communicate over the common channel x by sending the name a. Furthermore, P_1 and P_2 can act independently, so if, for instance, P_2 performs the visible action $\overline{x}\langle a\rangle$, P can also perform the action $\overline{x}\langle a\rangle$ and thus evolve to $x(y).\overline{y}\langle z\rangle \mid \overline{b}\langle b\rangle$.

The *restriction* $\underline{\texttt{new}}\, z\ P$ for a process $P \in \mathcal{P}^{\pi}$ and a name $z \in \mathcal{N}$, binds the name z to the process P. Thus, the scope of z is restricted to P. This concept can be seen similarly to the notion of private variables in the context of programming languages. We call z *restricted*, since the name z may only be used within P and not for a communication with the environment. For instance $\underline{\texttt{new}}\, z\,(z(x) \mid \overline{z}\langle y\rangle)$ can only invisibly communicate over z, whereby $\underline{\texttt{new}}\, z\,(\overline{x}\langle y\rangle.\overline{z}\langle a\rangle)$ can perform the visible output action $\overline{x}\langle y\rangle$, since z is not a part of the action. But there is one exception from this rule. It is also possible to widen the scope of a restricted name z – called *scope extrusion* –, if the restricted name is passed via a nonrestricted channel. For instance consider $P =_{\text{def}} \underline{\texttt{new}}\, z\,(\overline{x}\langle z\rangle.z(y))$ and $Q =_{\text{def}} x(a).\overline{a}\langle b\rangle$, then $P \mid Q$ can

communicate over channel x so that afterwards the former just to P known channel z is also known, but restricted, in the whole remaining process $\underline{\texttt{new}}\, z\,(z(y) \mid \overline{z}\langle b\rangle)$.

Finally, $A\langle\vec{v}\rangle$ with a process $P \in \mathcal{P}^\pi$ and a recursive definition $A(\vec{w}) =_{\text{def}} P$ is a *process call*, or simply *call*. The behavior of the call $A\langle\vec{v}\rangle$ is the same as the behavior of the process P', where P' results of P by the simultaneous replacement of every occurrence of the elements of $\vec{w}$ by the elements of $\vec{v}$. Hence, $\vec{v}$ and $\vec{w}$ have to be of the same length.

As a convention, we agree that the unary operators bind stronger than the binary ones, and also the sum binds more tightly than the parallel composition. Additionally, the prefix operator binds stronger than restriction. Thus, we can, for example, write $\underline{\texttt{new}}\, y\, \overline{x}\langle y\rangle.P \mid a(b).Q + M_1$ with $P, Q \in \mathcal{P}^\pi$ and $M_1 \in \mathcal{P}^\pi_M$ instead of $\underline{\texttt{new}}\, y\,(\overline{x}\langle y\rangle.P) \mid ((a(b).Q) + M_1)$. Furthermore, we agree that we can abbreviate a sequence of restrictions, for example $\underline{\texttt{new}}\, a_1\,(\ldots \underline{\texttt{new}}\, a_n\,(P)\ldots)$, by $\underline{\texttt{new}}\, a_1, \ldots, a_n\,(P)$.

We present two interesting syntactical subclasses of the π-calculus: the subclass of *restriction-free* processes and the subclass of *recursion-free* ones, which are already defined, for example, in [Mey09].

Definition 2.2.2 (Restriction-free) A process $P' \in \mathcal{P}^\pi$, which is constructed from the syntax of Definition 2.2.1 without using the $\underline{\texttt{new}}\, z\ M$ with $M \in \mathcal{P}^\pi_M$ and $\underline{\texttt{new}}\, z\ P$ with $P \in \mathcal{P}^\pi$ parts, is called *restriction-free*. The set of all restriction-free processes is denoted by $\mathcal{P}^\pi_{\texttt{resf}}$. $*$

Thus, there is no restriction operator within a restriction-free process. Analogously, we define the recursion-free process by not using the call operator from Definition 2.2.1, while building the process.

Definition 2.2.3 (Recursion-free) A process $P \in \mathcal{P}^\pi$, which is constructed from the syntax of Definition 2.2.1 without using the $A\langle\vec{v}\rangle$ part, is called *recursion-free*. The set of all recursion-free processes is denoted by $\mathcal{P}^\pi_{\texttt{recf}}$. $*$

These subclasses helps us in Section 4.2.2 to investigate the compositionality of our in Section 4.1 defined semantics.

2.2.2 Names and substitution

As already seen in the informal description of the syntax of the π-calculus, the restriction operator as well as the input prefix binds a name to a process. Intuitively, the *bound names* of a process are those, which occur as an object in an input

prefix and those who are restricted within a process. The other names, which additionally appear in a process, are called *free names*. Formally, we introduce with Definition 2.2.4 two functions to calculate the bound respectively free names of a process.

Definition 2.2.4 (Bound and free names of a process) Given $P, Q \in \mathcal{P}^\pi$ and $M, N \in \mathcal{P}^\pi_M$. Furthermore, let $a, b \in \mathcal{N}$. The function $\mathtt{bn} : \mathcal{P}^\pi \to \mathbb{P}(\mathcal{N})$ collects all *bound names* of a process and similarly the function $\mathtt{fn} : \mathcal{P}^\pi \to \mathbb{P}(\mathcal{N})$ computes the *free names* of a process. Both are inductively defined as follows:

$\mathtt{bn}(\mathbf{0})$	$=_{\text{def}}$	$\emptyset$	$\mathtt{fn}(\mathbf{0})$	$=_{\text{def}}$	$\emptyset$
$\mathtt{bn}(\tau.P)$	$=_{\text{def}}$	$\mathtt{bn}(P)$	$\mathtt{fn}(\tau.P)$	$=_{\text{def}}$	$\mathtt{fn}(P)$
$\mathtt{bn}(\overline{a}\langle b\rangle.P)$	$=_{\text{def}}$	$\mathtt{bn}(P)$	$\mathtt{fn}(\overline{a}\langle b\rangle.P)$	$=_{\text{def}}$	$\{a, b\} \cup \mathtt{fn}(P)$
$\mathtt{bn}(a(b).P)$	$=_{\text{def}}$	$\{b\} \cup \mathtt{bn}(P)$	$\mathtt{fn}(a(b).P)$	$=_{\text{def}}$	$\{a\} \cup (\mathtt{fn}(P) \setminus \{b\})$
$\mathtt{bn}(M + N)$	$=_{\text{def}}$	$\mathtt{bn}(M) \cup \mathtt{bn}(N)$	$\mathtt{fn}(M + N)$	$=_{\text{def}}$	$\mathtt{fn}(M) \cup \mathtt{fn}(N)$
$\mathtt{bn}(P \mid Q)$	$=_{\text{def}}$	$\mathtt{bn}(P) \cup \mathtt{bn}(Q)$	$\mathtt{fn}(P \mid Q)$	$=_{\text{def}}$	$\mathtt{fn}(P) \cup \mathtt{fn}(Q)$
$\mathtt{bn}(\underline{\mathtt{new}}\, a\ P)$	$=_{\text{def}}$	$\{a\} \cup \mathtt{bn}(P)$	$\mathtt{fn}(\underline{\mathtt{new}}\, a\ P)$	$=_{\text{def}}$	$\mathtt{fn}(P) \setminus \{a\}$
$\mathtt{bn}(A\langle\vec{v}\rangle)$	$=_{\text{def}}$	$\emptyset$	$\mathtt{fn}(A\langle\vec{v}\rangle)$	$=_{\text{def}}$	$\vec{v}$.

Furthermore, we call $\mathtt{n} : \mathcal{P}^\pi \to \mathbb{P}(\mathcal{N})$ with $\mathtt{n}(P) =_{\text{def}} \mathtt{fn}(P) \cup \mathtt{bn}(P)$ for all $P \in \mathcal{P}^\pi$ the *names* of a process P. ∗

According to that, we stipulate that for a recursive definition $A(\vec{w}) =_{\text{def}} P$, the free names of P are included in the parameter list; thus, $\mathtt{fn}(P) \subseteq \vec{w}$. This simplifies the handling of a recursive call and so the definition of the free names of a recursive call fits to this convention.

For the informal description of the semantics of the input process and the process call, we already used an idea of the replacement of names in a process. We now give a formal definition of a function, which replaces the free names of a process by some other names.

Definition 2.2.5 (Substitution) We define a *substitution* $\sigma : \mathcal{N} \to \mathcal{N}$, as the simultaneous replacement of names, that is, σ maps a finite number of names to other names and behaves as the identity otherwise. We write $\{^{a_1,\ldots,a_n}/_{b_1,\ldots,b_n}\}$ for some $n \in \mathbb{N}$ and $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathcal{N}$ for the substitution

$$\sigma(x) = \begin{cases} a_i & \text{if } x = b_i \text{ and } i \in \{1, \ldots, n\} \\ x & \text{else} \end{cases}$$

as an abbreviation. Furthermore, we define the *names* of such a substitution with $\mathtt{n}(\{^{a_1,\ldots,a_n}/_{b_1,\ldots,b_n}\}) =_{\text{def}} \{a_1,\ldots,a_n,b_1,\ldots,b_n\}$, the *support* with $\mathtt{supp}(\sigma) =_{\text{def}} \{x \in \mathcal{N} \mid \sigma(x) \neq x\}$ and the *co-support* of a such a substitution by $\mathtt{cosupp}(\sigma) =_{\text{def}} \{\sigma(x) \in \mathcal{N} \mid x \in \mathtt{supp}(\sigma)\}$. Hence, $\mathtt{n}(\sigma) = \mathtt{supp}(\sigma) \cup \mathtt{cosupp}(\sigma)$.

If the substitution σ just interchanges two names, for example, $\sigma = \{^{a,b}/_{b,a}\}$ we call σ a *transposition* and abbreviate it by $\{a \leftrightarrow b\}$. Furthermore, we define the application of a substitution σ on a set of names $X \subseteq \mathcal{N}$ by $X\sigma =_{\text{def}} \{\sigma(x) \mid x \in X\}$.

The *application* of a substitution σ to a process $P \in \mathcal{P}^\pi$, with $\mathtt{n}(\sigma) \cap \mathtt{bn}(P) = \emptyset$, is written as $P\sigma \in \mathcal{P}^\pi$ and is inductively defined

$$\begin{array}{llll}
\mathbf{0}\sigma & =_{\text{def}} \mathbf{0} & (M_1 + M_2)\,\sigma & =_{\text{def}} M_1\sigma + M_2\sigma \\
(\tau.P_1)\,\sigma & =_{\text{def}} \tau.\,(P_1\sigma) & (P_1 \mid P_2)\,\sigma & =_{\text{def}} P_1\sigma \mid P_2\sigma \\
(x(y).P_1)\,\sigma & =_{\text{def}} \sigma(x)(y).\,(P_1\sigma) & (\underline{\mathtt{new}}\,a\;P_1)\,\sigma & =_{\text{def}} \underline{\mathtt{new}}\,a\,(P_1\sigma) \\
(\overline{x}\langle y\rangle.P_1)\,\sigma & =_{\text{def}} \overline{\sigma(x)}\langle\sigma(y)\rangle.\,(P_1\sigma) & A\langle\vec{v}\rangle\sigma & =_{\text{def}} A\langle\vec{v}\sigma\rangle
\end{array}$$

for $x, y, a \in \mathcal{N}$, $\vec{v} \subseteq \mathcal{N}$, $P_1, P_2 \in \mathcal{P}^\pi$ and $M_1, M_2 \in \mathcal{P}^\pi_M$. ∗

For the definition of the *α-convertibility*, a relation collecting processes which can be obtained from each other by only a finite number of replacements of bound names, we refer to [CC03]. Therefore, we first define the meaning of a *congruence relation* on processes.

Definition 2.2.6 (Congruence relation) A relation $\equiv \subseteq \mathcal{P}^\pi \times \mathcal{P}^\pi$ is an *equivalence relation*, if

$$\forall P \in \mathcal{P}^\pi\colon P \equiv P \qquad \text{(Reflexivity)}$$
$$\forall P, Q \in \mathcal{P}^\pi\colon P \equiv Q \Rightarrow Q \equiv P \qquad \text{(Symmetry)}$$
$$\forall P, Q, R \in \mathcal{P}^\pi\colon P \equiv Q \wedge Q \equiv R \Rightarrow P \equiv R \qquad \text{(Transitivity)}$$

holds. If such a relation is preserved under each defined operator, it is called a *process congruence*. In the notation according to [Mey09], an equivalence relation is a process congruence if

$$\forall P, Q, R \in \mathcal{P}^\pi\colon P \equiv Q \Rightarrow P \mid R \equiv Q \mid R \qquad \text{(Cong Par R)}$$
$$\forall P, Q, R \in \mathcal{P}^\pi\colon P \equiv Q \Rightarrow R \mid P \equiv R \mid Q \qquad \text{(Cong Par L)}$$
$$\forall P, Q \in \mathcal{P}^\pi, M \in \mathcal{P}^\pi_M, \pi \text{ prefix}\colon P \equiv Q \Rightarrow \pi.P + M \equiv \pi.Q + M \qquad \text{(Cong Sum1 R)}$$
$$\forall P, Q \in \mathcal{P}^\pi, M \in \mathcal{P}^\pi_M, \pi \text{ prefix}\colon P \equiv Q \Rightarrow M + \pi.P \equiv M + \pi.Q \qquad \text{(Cong Sum1 L)}$$

$$\forall M, M_1, M_2 \in \mathcal{P}^\pi_M, a \in \mathcal{N}:$$
$$M_1 \equiv M_2 \Rightarrow \underline{\texttt{new}}\, a\ M_1 + M \equiv \underline{\texttt{new}}\, a\ M_2 + M \qquad \text{(Cong Sum2 R)}$$
$$M_1 \equiv M_2 \Rightarrow M + \underline{\texttt{new}}\, a\ M_1 \equiv M + \underline{\texttt{new}}\, a\ M_2 \qquad \text{(Cong Sum2 L)}$$
$$\forall P, Q \in \mathcal{P}^\pi, a \in \mathcal{N}\colon P \equiv Q \Rightarrow \underline{\texttt{new}}\, a\ P \equiv \underline{\texttt{new}}\, a\ Q \qquad \text{(Cong Res)}$$

holds. ∗

Since the choice operator is only defined on prefix processes or restrictions of sums, in general $P + M$ and $\underline{\texttt{new}}\, a\ P + M$ is not a process. Thus, every possible choice process is covered by the (Cong Sum1 R) respectively (Cong Sum1 L) and (Cong Sum2 R) respectively (Cong Sum2 L) properties. Furthermore, note that with $M = \mathbf{0}$ the prefix operator is also covered by the first two sum implications and the restriction operator by the second two sum implications together with the last implication of the congruence.

By adding two additional rules, we get a relation which is preserved under the renaming of bound names within a process.

Definition 2.2.7 (α-convertibility) The least congruence relation on processes, denoted by $=_\alpha \subseteq \mathcal{P}^\pi \times \mathcal{P}^\pi$, with

$$\forall P \in \mathcal{P}^\pi, a \in \mathcal{N}, p \notin \mathtt{n}(P) : \underline{\texttt{new}}\, a\ P =_\alpha \underline{\texttt{new}}\, p\, (P\,\{^p/_a\}) \qquad \text{(Alpha Res)}$$
$$\forall P \in \mathcal{P}^\pi, x, y \in \mathcal{N}, p \notin (\mathtt{n}(P) \cup \{x\}) : x(y).P =_\alpha x(p).\,(P\,\{^p/_y\}) \qquad \text{(Alpha Inp)}$$

is called the *α-convertibility relation*. For all $(P, Q) \in =_\alpha$ we say P and Q are *α-convertible* and the equivalence classes are noted by $[P] \in \mathcal{P}^\pi_\alpha$ for all $P \in \mathcal{P}^\pi$, where $\mathcal{P}^\pi_\alpha$ denotes the set of all equivalence classes modulo α-convertibility. ∗

Intuitively, we can rename the bound names within a process without changing anything of its behavior or nature. This concept can similarly be seen as the idea of local variables in programming languages.

Since there is the possibility to change the bound names anytime we want without changing the process' behavior, we introduce the convention that the bound names of an investigated set of processes and substitutions are unique.

Convention 2.2.1 (Uniqueness of bound names) *Given $P_1, \ldots, P_n \in \mathcal{P}^\pi$ a collection of π-calculus processes and a collection $\sigma_1, \ldots, \sigma_m$ of substitutions for $n, m \in \mathbb{N}$, we stipulate that*

1. $\forall i, j \in \{1, \ldots, n\}$ *with* $i \neq j : \mathtt{bn}(P_i) \cap \mathtt{bn}(P_j) = \emptyset$,

2. $\bigcup_{i \in \{1,\ldots,n\}} \mathtt{bn}(P_i) \cap \bigcup_{i \in \{1,\ldots,n\}} \mathtt{fn}(P_i) = \emptyset$,

3. *For all* $i \in \{1, \ldots, n\}$ *and* $\circ \in \{|, +\}$ *the following implications hold*

$$\begin{array}{rcll} P_i & = & P \circ Q & \Rightarrow \mathtt{bn}(P) \cap \mathtt{bn}(Q) = \emptyset \\ P_i & = & a(x).P & \Rightarrow x \notin \mathtt{bn}(P) \\ P_i & = & \underline{\mathtt{new}}\, z\ P & \Rightarrow z \notin \mathtt{bn}(P) \end{array}$$

   with $P, Q \in \mathcal{P}^\pi$ *and* $a, x, z \in \mathcal{N}$,

4. $\mathtt{bn}(P_i) \cap \mathtt{n}(\sigma_j) = \emptyset$ *for all* $i \in \{1, \ldots, n\}$ *and* $j \in \{1, \ldots, m\}$

holds. ∘

The first condition ensures that the bound names among the processes are different, the second states that the bound names of a process and its free names and all the free names of the other processes are different, the third one ensures that the bound names differ within a process, and the last one states that the bound names of a process are different from the names of the substitutions under consideration.

Hence, any time we receive a process which violates the convention, we silently rename the bound names so that we receive a proper process.

2.2.3 Structural congruence and extended standard form

The α-convertibility relation identifies all processes P and Q where P can be obtained from Q by a renaming of bound names. We now extend this relation, to gain the so-called *structural congruence*.

Definition 2.2.8 (Structural congruence) The smallest congruence, denoted by $\equiv \subseteq \mathcal{P}^\pi \times \mathcal{P}^\pi$, which satisfies the axioms of Figure 2.1, is called *structural congruence.* ∗

The (RES-PAR) axiom is normally known as *scope extrusion.* With the added (RES-PRE) and (RES-SUM) axioms, we now have the ability to extrude the scope in two more ways. Besides, note that within these three rules the side condition that the name a is not allowed to appear free in the other part, is by Convention 2.2.1 the same as the name a is not allowed to appear in the other part at all.

$$P \equiv Q, \text{ if } P =_\alpha Q \qquad \text{(ALPHA)}$$

$$M + \mathbf{0} \equiv M \qquad \text{(SUM-NEU)}$$
$$M_1 + M_2 \equiv M_2 + M_1 \qquad \text{(SUM-COM)}$$
$$M_1 + (M_2 + M_3) \equiv (M_1 + M_2) + M_3 \qquad \text{(SUM-ASS)}$$

$$P \mid \mathbf{0} \equiv P \qquad \text{(PAR-NEU)}$$
$$P_1 \mid P_2 \equiv P_2 \mid P_1 \qquad \text{(PAR-COM)}$$
$$P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \qquad \text{(PAR-ASS)}$$

$$\underline{\texttt{new}}\, a\, \mathbf{0} \equiv \mathbf{0} \qquad \text{(RES-ABS)}$$
$$\underline{\texttt{new}}\, a\, (\underline{\texttt{new}}\, b\, P) \equiv \underline{\texttt{new}}\, b\, (\underline{\texttt{new}}\, a\, P) \qquad \text{(RES-TRA)}$$
$$\underline{\texttt{new}}\, a\, (\pi.P) \equiv \pi.\, (\underline{\texttt{new}}\, a\, P)\,, \text{ if } a \notin \texttt{fn}(\pi) \qquad \text{(RES-PRE)}$$
$$\underline{\texttt{new}}\, a\, (M_1 + M_2) \equiv M_1 + (\underline{\texttt{new}}\, a\, M_2)\,, \text{ if } a \notin \texttt{fn}(M_1) \qquad \text{(RES-SUM)}$$
$$\underline{\texttt{new}}\, a\, (P_1 \mid P_2) \equiv P_1 \mid (\underline{\texttt{new}}\, a\, P_2)\,, \text{ if } a \notin \texttt{fn}(P_1) \qquad \text{(RES-PAR)}$$

Figure 2.1: Axioms of the structural congruence on processes.

Milner's well-known *standard form* [Mil99] expands the scope of a restriction as much as possible. We added in Figure 2.1 the (RES-PRE) and (RES-SUM) axioms to Milner's structural congruence so that it is now possible to move *all* restrictions right at the front of a process.

Definition 2.2.9 (Extended standard form) A process $P \in \mathcal{P}^\pi$ with

$$P =_{\text{def}} \underline{\texttt{new}}\, \vec{z}\, (P_1 \mid \ldots \mid P_n)$$

and $\vec{z} \subseteq \mathcal{N}$ and $n \in \mathbb{N}$ is said to be in *extended standard form*, if $P_1, \ldots, P_n \in \mathcal{P}^\pi_{\texttt{resf}}$. If $\vec{z}$ is an empty sequence, we omit the restriction operator and if $n = 0$ then the form of the process is $P = \underline{\texttt{new}}\, \vec{z}\, \mathbf{0}$. The set of all processes in extended standard form is denoted by $\mathcal{P}^\pi_{\texttt{esf}}$. $*$

Thus, if we consider a process $P =_{\text{def}} a(x).\overline{x}\langle a\rangle + z(x).\underline{\texttt{new}}\, y\, \overline{x}\langle y\rangle$, we know that referring to Milner, P is in standard form. But with the (RES-PRE) and (RES-SUM) axioms we can get rid of the restriction operator inside the process and move it to the front. Hence, $P \equiv a(x).\overline{x}\langle a\rangle + \underline{\texttt{new}}\, y\, (z(x).\overline{x}\langle y\rangle) \equiv \underline{\texttt{new}}\, y\, (a(x).\overline{x}\langle a\rangle + z(x).\overline{x}\langle y\rangle)$. Thereby, we see the necessity of the slightly adapted version of the guarded choice

within the syntax of the π-calculus. For the intermediate step it is necessary that the parts of a sum can also start with a restriction and not only with a prefix. In general, the intermediate steps during a transformation of a process in extended standard form is the only reason for extending the syntax.

Moreover, it is possible to transform every process with the axioms of Figure 2.1 to a structurally congruent process in extended standard form.

Lemma 2.2.1 (Extended standard form) *Every process $P \in \mathcal{P}^\pi$ can be transformed in a process $P' \in \mathcal{P}^\pi_{\mathtt{esf}}$, with $P \equiv P'$.* ♢

Proof: Let $P \in \mathcal{P}^\pi$. We proceed by induction over the structure of processes.

Base case $P = \mathbf{0}$: By the (RES-ABS) axiom of Figure 2.1, we know $P \equiv \underline{\mathtt{new}}\, a\, (\mathbf{0})$ holds for an arbitrary $a \in \mathcal{N}$. And with the definition of the extended standard form $\underline{\mathtt{new}}\, a\, (\mathbf{0}) \in \mathcal{P}^\pi_{\mathtt{esf}}$ holds.

Base case $P = A\langle \vec{v} \rangle$: With $\vec{z}$ the empty sequence and since then the restriction operator is omitted according to Definition 2.2.9, we know $P = \underline{\mathtt{new}}\, \vec{z}\; A\langle \vec{v} \rangle$. Hence, since the structural congruence is reflexive $P \equiv P$ and since $A\langle \vec{v} \rangle \in \mathcal{P}^\pi_{\mathtt{resf}}$, we know $P \in \mathcal{P}^\pi_{\mathtt{esf}}$.

Induction hypothesis: For all structurally simpler $Q, R \in \mathcal{P}^\pi$ there exist processes $Q', R' \in \mathcal{P}^\pi_{\mathtt{esf}}$ with $Q \equiv Q'$ and $R \equiv R'$.

Induction step: Let $Q, R \in \mathcal{P}^\pi$.

Case $P = Q \mid R$: From the induction hypothesis we know that there are processes $Q', R' \in \mathcal{P}^\pi_{\mathtt{esf}}$ with $Q \equiv Q'$ and $R \equiv R'$. So there are sequences $\vec{z_1}, \vec{z_2} \subseteq \mathcal{N}$ and processes $Q_1, \ldots, Q_n, R_1, \ldots, R_m \in \mathcal{P}^\pi_{\mathtt{resf}}$ with $n, m \in \mathbb{N}$, such that $Q' =_{\text{def}} \underline{\mathtt{new}}\, \vec{z_1}\, (Q_1 \mid \ldots \mid Q_n)$ and $R' =_{\text{def}} \underline{\mathtt{new}}\, \vec{z_2}\, (R_1 \mid \ldots \mid R_m)$ holds. Since there is just a finite number of free names in a collection of processes, we can choose names $\vec{z_1'}, \vec{z_2'} \subseteq \mathcal{N}$, with $\left|\vec{z_1'}\right| = |\vec{z_1}|$ and $\left|\vec{z_2'}\right| = |\vec{z_2}|$ such that for all names $x \in \vec{z_1'}$ it holds that

$$x \notin (\mathtt{fn}(Q_1 \mid \ldots \mid Q_n) \cup \mathtt{fn}(R_1 \mid \ldots \mid R_m) \cup z_2')$$

and similarly $x' \notin (\mathtt{fn}(Q_1 \mid \ldots \mid Q_n) \cup \mathtt{fn}(R_1 \mid \ldots \mid R_m) \cup z_1')$ holds for all names $x' \in \vec{z_2'}$. Since the structural congruence is preserved under α-conversion, we know that $Q' \equiv \underline{\mathtt{new}}\, \vec{z_1'} \left((Q_1 \mid \ldots \mid Q_n) \left\{ {}^{\vec{z_1'}}/_{\vec{z_1}} \right\} \right)$ and $R' \equiv$

$\underline{\texttt{new}}\, \vec{z_2'}\left((R_1 \mid \ldots \mid R_m)\left\{\vec{z_2}/\vec{z_2'}\right\}\right)$ holds. From the definition of the substitution on processes, we know that there are processes $Q_1', \ldots, Q_n' \in \mathcal{P}^{\pi}_{\texttt{resf}}$ and $R_1', \ldots, R_m' \in \mathcal{P}^{\pi}_{\texttt{resf}}$ with $Q_i' =_{\text{def}} Q_i\left\{\vec{z_1}/\vec{z_1'}\right\}$ for all $i \in \{1, \ldots, n\}$ and $R_j' =_{\text{def}} R_j\left\{\vec{z_2}/\vec{z_2'}\right\}$ for all $j \in \{1, \ldots, m\}$. Hence, there are processes $Q'', R'' \in \mathcal{P}^{\pi}_{\texttt{esf}}$ with $Q'' =_{\text{def}} \underline{\texttt{new}}\, \vec{z_1'}\,(Q_1' \mid \ldots \mid Q_n')$ and $R'' =_{\text{def}} \underline{\texttt{new}}\, \vec{z_2'}\,(R_1' \mid \ldots \mid R_m')$. Since the structural congruence is transitive, we know that $Q \equiv Q''$ and $R \equiv R''$ holds. Furthermore, we know with the (Cong Par R) implication of Definition 2.2.6 that $Q \mid R \equiv Q'' \mid R$ and with (Cong Par L) that $Q'' \mid R \equiv Q'' \mid R''$ holds. Hence, with the transitivity of the structural congruence $Q \mid R \equiv Q'' \mid R''$ holds. Thus, consider $Q'' \mid R'' = \underline{\texttt{new}}\, \vec{z_1'}\,(Q_1' \mid \ldots \mid Q_n') \mid \underline{\texttt{new}}\, \vec{z_2'}\,(R_1' \mid \ldots \mid R_m')$. Since there is no $x \in \vec{z_2'}$ with $x \in \texttt{fn}(Q'')$, we know from the (RES-PAR) axiom of Figure 2.1 that $Q'' \mid R'' \equiv \underline{\texttt{new}}\, \vec{z_2'}\,(Q'' \mid (R_1' \mid \ldots \mid R_m'))$. Moreover, with (PAR-COM) and (RES-PAR), we also know $Q'' \mid R'' \equiv \underline{\texttt{new}}\, \vec{z_1'}\left(\underline{\texttt{new}}\, \vec{z_2'}\,((Q_1' \mid \ldots \mid Q_n') \mid (R_1' \mid \ldots \mid R_m'))\right)$, since the definition of the congruence relation yields that the axioms are also applicable in the context of a restriction. Hence, we found a process $P' \in \mathcal{P}^{\pi}_{\texttt{esf}}$ with $P' =_{\text{def}} \underline{\texttt{new}}\, \vec{z_3}\,(Q_1' \mid \ldots \mid Q_n' \mid R_1' \mid \ldots \mid R_m')$ with $\vec{z_3} = \vec{z_1'} \cup \vec{z_2'}$ and $P \equiv P'$.

Case $P = \pi.Q$**:** Again the induction hypothesis yields that there are names $z \subseteq \mathcal{N}$ and processes $Q_1, \ldots, Q_n \in \mathcal{P}^{\pi}_{\texttt{resf}}$, with $n \in \mathbb{N}$, such that $Q \equiv \underline{\texttt{new}}\, \vec{z}\,(Q_1 \mid \ldots \mid Q_n)$. Additionally, we can find fresh names $\vec{z'} \subseteq \mathcal{N}$, with $\left|\vec{z'}\right| = |\vec{z}|$ such that for all $x \in \vec{z'}$, we know that $x \notin \texttt{fn}(\pi)$ and $x \notin \texttt{fn}(Q_1 \mid \ldots \mid Q_n)$ holds. And with the (ALPHA) axiom of Figure 2.1 we know $Q \equiv \underline{\texttt{new}}\, \vec{z'}\left((Q_1 \mid \ldots \mid Q_n)\left\{\vec{z'}/\vec{z}\right\}\right)$ holds. Thus, with $Q_i' =_{\text{def}} Q_i\left\{\vec{z'}/\vec{z}\right\}$ for all $i \in \{1, \ldots, n\}$, we know $Q \equiv \underline{\texttt{new}}\, \vec{z'}\,(Q_1' \mid \ldots \mid Q_n')$ and so with the (Cong Sum1 L) implication of Definition 2.2.6, we know $P = \pi.Q \equiv P'$, with $P' =_{\text{def}} \pi.\underline{\texttt{new}}\, \vec{z'}\,(Q_1' \mid \ldots \mid Q_n')$. Furthermore, with the (RES-PRE) axiom of Figure 2.1 we know $P' \equiv P''$, with $P'' =_{\text{def}} \underline{\texttt{new}}\, \vec{z'}\,(\pi.\,(Q_1' \mid \ldots \mid Q_n'))$. Hence, by transitivity of the structural congruence, there is a process $P'' \in \mathcal{P}^{\pi}_{\texttt{esf}}$ with $P \equiv P''$.

Case $P = \underline{\texttt{new}}\, a\, Q$**:** Since there is a $Q' \in \mathcal{P}^{\pi}_{\texttt{esf}}$ with $Q \equiv Q'$ due to the induction hypothesis, we know with Cong Res of Definition 2.2.6 that $P \equiv \underline{\texttt{new}}\, a\, Q'$ holds. Furthermore, we know $\underline{\texttt{new}}\, a\, Q'$ itself is in extended standard form, since $Q' \in \mathcal{P}^{\pi}_{\texttt{esf}}$ holds.

Case $P = Q + R$**:** The induction hypothesis yields that there are processes

$Q', R' \in \mathcal{P}^{\pi}_{\texttt{esf}}$ with $Q \equiv Q'$ and $R \equiv R'$ and so there exists $\vec{z_1}, \vec{z_2} \subseteq \mathcal{N}$ and $Q_1, \ldots, Q_n, R_1, \ldots, R_m \in \mathcal{P}^{\pi}_{\texttt{resf}}$ with $m, n \in \mathbb{N}$ such that $Q' = \underline{\texttt{new}}\, \vec{z_1}\, (Q_1 \mid \ldots \mid Q_n)$ and $R' = \underline{\texttt{new}}\, \vec{z_2}\, (R_1 \mid \ldots \mid R_m)$. From the definition of the syntax of the π-calculus processes we know $Q, R \in \mathcal{P}^{\pi}_{M}$, otherwise P would not be a process. Hence, Q and R are each either an inaction, an input process, a choice process, or a restricted sum. Furthermore, we know that for all $M \in \mathcal{P}^{\pi}_{M}$ and $P \in \mathcal{P}^{\pi}$ that $M \equiv P$ implies $P \in \mathcal{P}^{\pi}_{M}$, since the only rules of the structural congruence, which influence the structure of a process significantly to possibly destroy the sum structure, are the (RES-PRE), (RES-SUM) and (RES-PAR) axioms of Figure 2.1. And since the (RES-PRE) and (RES-SUM) axioms preserve the sum structure and (RES-PAR) is not applicable for sums, we know that $Q', R' \in \mathcal{P}^{\pi}_{M}$. Thus, there are $M_1, M_2 \in \mathcal{P}^{\pi}_{M} \cap \mathcal{P}^{\pi}_{\texttt{resf}}$ such that $Q' = \underline{\texttt{new}}\, \vec{z_1}\ M_1$ and $R' = \underline{\texttt{new}}\, \vec{z_2}\ M_2$.

Analogously to the case of the parallel composition, we know that we can use α-conversion to ensure that all bound names are different from each other and from all the other names and preserve the structural congruence. Thus, there are fresh names $\vec{z_1'}, \vec{z_2'} \subseteq \mathcal{N}$ which do not appear free in any process under consideration with $Q' \equiv \underline{\texttt{new}}\, \vec{z_1'}\ M_1'$ and $R' \equiv \underline{\texttt{new}}\, \vec{z_2'}\ M_2'$ for some $M_1', M_2' \in \mathcal{P}^{\pi}_{M}$ with $M_1' =_{\text{def}} M_1 \left\{ \vec{z_1'}/\vec{z_1} \right\}$ and $M_2' =_{\text{def}} M_2 \left\{ \vec{z_2'}/\vec{z_2} \right\}$. The transitivity of the structural congruence yields that $Q \equiv \underline{\texttt{new}}\, \vec{z_1'}\ M_1'$ and $R \equiv \underline{\texttt{new}}\, \vec{z_2'}\ M_2'$ holds. Since $\underline{\texttt{new}}\, \vec{z_1'}\ M_1', \underline{\texttt{new}}\, \vec{z_2'}\ M_2' \in \mathcal{P}^{\pi}_{M}$ and since $\equiv$ is a congruence and accordingly is preserved for every context, we know that $P \equiv \underline{\texttt{new}}\, \vec{z_1'}\, (M_1') + \underline{\texttt{new}}\, \vec{z_2'}\, (M_2')$. Thus, with the (RES-SUM) axiom of Figure 2.1 we know $P \equiv \underline{\texttt{new}}\, \vec{z_2'} \left(\underline{\texttt{new}}\, \vec{z_1'}\, (M_1') + M_2 \right)$ holds. The (SUM-COM) and the (RES-SUM) axiom of Figure 2.1 and the (Cong Sum2 L) implication of Definition 2.2.6 yields that $P \equiv P'$ with $P' =_{\text{def}} \underline{\texttt{new}}\, \vec{z_2'} \left(\underline{\texttt{new}}\, \vec{z_1'}\, (M_2' + M_1') \right)$ holds and so $P' \in \mathcal{P}^{\pi}_{\texttt{esf}}$. ■

Thus, we know that we are able to transform every process in a structurally congruent process which has all the restrictions right at the front of the process.

2.2.4 Operational semantics

We already explained the intuition of the semantics of the π-calculus in Section 2.2.1. In this section we formalize this intuition by presenting a definition of an *operational*

semantics of the π-calculus. This definition yields a labeled transition system called the *early transition system* as presented in [SW01] in Table 1.5 on page 38. For this thesis, we applied only minor changes to the original presentation. In particular, we replace the replication rules by the $E-CALL$ rule, which is also defined in [SW01], and omit the matching rule due to the definition of our syntax. Furthermore, the labels of the transitions are adapted to the notion of this thesis.

Definition

For the labeling of the transitions, we define the set of all *output actions* as $\mathtt{Out} =_{\text{def}} \{\overline{x}\langle y\rangle \mid x, y \in \mathcal{N}\}$, the set of all *input actions* as $\mathtt{In} =_{\text{def}} \{x\, y \mid x, y \in \mathcal{N}\}$ and the set of all *bound output actions* as $\mathtt{Bout} =_{\text{def}} \{\overline{x}(y) \mid \exists x, y \in \mathcal{N} : x \neq y\}$. The meaning of the bound output will be explained shortly. Thus, the set of all *actions* is defined as $\mathtt{Act} =_{\text{def}} \mathtt{In} \cup \mathtt{Out} \cup \mathtt{Bout} \cup \{\tau\}$ with $\alpha, \beta, \ldots \in \mathtt{Act}$ as its standard representatives.

Furthermore, we expand the definition of the *free* and *bound names* of processes in Figure 2.2 such that they are also applicable to actions. Additionally, Figure 2.2 shows the application of a *substitution on actions* and the *conjugation*.

α	denotation	$\mathtt{n}(\alpha)$	$\mathtt{bn}(\alpha)$	$\mathtt{fn}(\alpha)$	$\sigma(\alpha)$	$\overline{\alpha}$
τ	internal	$\emptyset$	$\emptyset$	$\emptyset$	τ	τ
$a\, x$	input	$\{a, x\}$	$\emptyset$	$\{a, x\}$	$\sigma(a)\, \sigma(x)$	$\overline{a}\langle x\rangle$
$\overline{a}\langle x\rangle$	output	$\{a, x\}$	$\emptyset$	$\{a, x\}$	$\overline{\sigma(a)}\langle\sigma(x)\rangle$	$a\, x$
$\overline{a}(x)$	bound output	$\{a, x\}$	$\{x\}$	$\{a\}$	$\overline{\sigma(a)}(x)$	$a\, x$

Figure 2.2: Free and bound names of actions.

It seems to be a bit unintuitive that the set of bound names of an input action $a\, x$ is empty, since in the view of processes the name x is bound. But with the intuition that the actions present the behavior a process has performed, we see that the name x has already been sent over the channel a and is not a placeholder any longer. This fits to the definition of the operational semantics in Figure 2.3, where we see that a bound name in an input prefix is instantiated directly when the input transition is inferred. This principle is called an *early instantiation*.

To abstract from the replacement of bound names, we define the transition relation on the equivalence classes modulo α-convertibility of processes with the actions as its labels.

Definition 2.2.10 The relation $\left\{\xrightarrow{\alpha} \mid \ \alpha \in \mathtt{Act}\right\} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ is called the *early transition system* and is defined by the rules in Figure 2.3 apart from the omission of the rules $E - SUM_R$, $E - PAR_R$, $E - COM_R$ and $E - CLOSE_R$ for a shorter presentation. They can be obtained from their related rules by interchanging the roles of P and Q. $*$

$$\underline{E\text{–}TAU}: \ \frac{}{[\tau.P] \xrightarrow{\tau} [P]} \qquad \underline{E\text{–}CALL}: \ \frac{}{[A\langle\vec{v}\rangle] \xrightarrow{\tau} [P\,\{\vec{v}/\vec{w}\}]} \ A(\vec{w}) =_{\text{def}} P$$

$$\underline{E\text{–}OUT}: \ \frac{}{[\overline{x}\langle y\rangle.P] \xrightarrow{\overline{x}\langle y\rangle} [P]} \qquad \underline{E\text{–}IN}: \ \frac{}{[x(z).P] \xrightarrow{x\,y} [P\,\{y/z\}]}$$

$$\underline{E\text{–}SUM_L}: \ \frac{[P] \xrightarrow{\alpha} [P']}{[P+Q] \xrightarrow{\alpha} [P']} \qquad \underline{E\text{–}RES}: \ \frac{[P] \xrightarrow{\alpha} [P']}{[\underline{\mathtt{new}}\,z\ P] \xrightarrow{\alpha} [\underline{\mathtt{new}}\,z\ P']} \ z \notin n(\alpha)$$

$$\underline{E\text{–}PAR_L}: \ \frac{[P] \xrightarrow{\alpha} [P']}{[P \mid Q] \xrightarrow{\alpha} [P' \mid Q]} \ \mathtt{bn}(\alpha) \cap \mathtt{fn}(Q) = \emptyset$$

$$\underline{E\text{–}OPEN}: \ \frac{[P] \xrightarrow{\overline{x}\langle z\rangle} [P']}{[\underline{\mathtt{new}}\,z\ P] \xrightarrow{\overline{x}(z)} [P']} \ z \neq x \qquad \underline{E\text{–}COM_L}: \ \frac{[P] \xrightarrow{\overline{x}\langle y\rangle} [P'] \quad [Q] \xrightarrow{x\,y} [Q']}{[P \mid Q] \xrightarrow{\tau} [P' \mid Q']}$$

$$\underline{E\text{–}CLOSE_L}: \ \frac{[P] \xrightarrow{\overline{x}(z)} [P'] \quad [Q] \xrightarrow{x\,z} [Q']}{[P \mid Q] \xrightarrow{\tau} [\underline{\mathtt{new}}\,z\,(P' \mid Q')]} \ z \notin \mathtt{fn}(Q)$$

Figure 2.3: The *early transition system* [SW01].

The labeled transition system fulfills two main tasks. On the one hand, it defines the activity within the process and on the other hand it exhibits the process' potential to interact with its environment.

Thereby, the internal behavior is represented by τ transitions, that is $[P] \xrightarrow{\tau} [Q]$ expresses that P and all processes α-convertible to P can invisibly evolve to Q and to all processes which are α-convertible to Q. For the external behavior there is a transition labeled with $a\,x$ for an input possibility, $\overline{a}\langle x\rangle$ for an output potential and the so-called bound output $\overline{a}(x)$, to handle the transfer of a name x which is bound under the restriction. The intuition of a visible transition of a process is that we

can compose another process with a corresponding visible transition in parallel and the composed processes can communicate with each other.

Note that we abbreviate multiple transitions like for example $\left[a(b).\overline{b}\langle c\rangle\right] \xrightarrow{a\,d} \left[\overline{d}\langle c\rangle\right]$ and $\left[\overline{d}\langle c\rangle\right] \xrightarrow{\overline{d}\langle c\rangle} [\mathbf{0}]$ by $\left[a(b).\overline{b}\langle c\rangle\right] \xrightarrow{a\,d} \left[\overline{d}\langle c\rangle\right] \xrightarrow{\overline{d}\langle c\rangle} [\mathbf{0}]$.

A main challenge while defining an operational semantics for π-calculus processes is the treatment of bound names within a process and especially the treatment of the transmission of bound names. Therefore, we have to take a closer look upon the $E-RES$, $E-PAR_L$ (respectively $E-PAR_R$), $E-OPEN$ and $E-CLOSE_L$ (respectively $E-CLOSE_R$) rules. The other rules straightforwardly fit to the intuition described in Section 2.2.1.

Thus, with the $E-SUM_L$ (respectively $E-SUM_R$) rule an alternative either evolves to the process arisen from the left or from the right part of the sum, whereby the other part vanishes. Besides, there are only the $E-TAU$, $E-OUT$ and $E-IN$ axioms to handle a prefix in the described manner. Note that there is no condition within the $E-IN$ rule for the reception of a name. Thus, *any* name can be send to an input process. Due to that, it is enough to consider the case where the object of the input is equal to the object of the output transition within the premise of the $E-COM_L$ (respectively $E-COM_R$) rule for a communication between two processes. For instance, consider $P =_{\text{def}} \overline{a}\langle b\rangle$ and $Q =_{\text{def}} a(x).\overline{x}\langle x\rangle$. Since $[P] \xrightarrow{\overline{a}\langle b\rangle} [\mathbf{0}]$ and there is also a transition $\xrightarrow{a\,b}$ such that $[Q] \xrightarrow{a\,b} \left[\overline{b}\langle b\rangle\right]$, the $E-COM_L$ rule infers $[\overline{a}\langle b\rangle \mid a(x).\overline{x}\langle x\rangle] \xrightarrow{\tau} \left[\mathbf{0} \mid \overline{b}\langle b\rangle\right]$. Furthermore, note that in Figure 2.3 there is no inference rule for the inaction. Hence, an inaction has no transition and thus, no behavior.

By investigating the $E-CALL$ rule, we notice that a function call is resolved by a τ transition, so every function call "costs" one transition. This may seem a bit constructed for the modeling of problems outside the area of computer science, but in the topic of programming it is sensible that a computer takes an internal step to compute a function call.

From the $E-RES$ rule we know that a restriction does not delimit the behavior of a process as long as the bound channel is not used to compute the behavior. With the side condition $z \notin \mathtt{n}(\alpha)$ within the $E-RES$ rule, we prevent a private channel to be visible for the environment. Consider, for example, $P =_{\text{def}} \underline{\mathtt{new}}\, a\,(a(x).\overline{x}\langle c\rangle \mid \overline{a}\langle b\rangle.b(d))$. The bound channel a can only be used to communicate within its scope – for instance $[P] \xrightarrow{\tau} \left[\underline{\mathtt{new}}\, a\left(\overline{b}\langle c\rangle \mid b(d)\right)\right]$ – but there is no visible transition starting in $[P]$. Otherwise, each of such transitions would uncover

the bound channel to the environment.

By Convention 2.2.1 we know that we do not have to consider the case that the bound name is the object of an input prefix. But note that nevertheless for $P =_{\text{def}} \underline{\texttt{new}}\, a\, (x(y).\overline{y}\langle a\rangle)$ there is a transition labeled with $x\, a$ inferred by the $E - RES$ rule and starting in $[P]$, since, for example, $\underline{\texttt{new}}\, b\, (x(y).\overline{y}\langle b\rangle) \in [P]$ holds. But we see that the name used as object of the transition's label can in general not be the bound name of the resulting process: $[P] \xrightarrow{x\, a} [\underline{\texttt{new}}\, c\, \overline{a}\langle c\rangle]$, but $\underline{\texttt{new}}\, a\, \overline{a}\langle a\rangle \notin [\underline{\texttt{new}}\, c\, \overline{a}\langle c\rangle]$.

Intuitively, the open and close rules resemble the structural law of scope extrusion for parallel processes. Consider, for example, $P =_{\text{def}} \underline{\texttt{new}}\, a\, \overline{x}\langle a\rangle.a(y)$ and $Q =_{\text{def}} x(b).\overline{b}\langle x\rangle$. We know from Figure 2.1 that $\underline{\texttt{new}}\, a\, (\overline{x}\langle a\rangle.a(y)) \mid Q \equiv \underline{\texttt{new}}\, a\, (\overline{x}\langle a\rangle.a(y) \mid Q)$ holds, since $a \notin \texttt{fn}(Q)$. With the $E - IN$, $E - OUT$, $E - COM_L$ and $E - RES$ rules we know $[\underline{\texttt{new}}\, a\, (\overline{x}\langle a\rangle.a(y) \mid Q)] \xrightarrow{\tau} [\underline{\texttt{new}}\, a\, (a(y) \mid \overline{a}\langle x\rangle)]$ and we reach the same for $[P \mid Q]$ by using the $E{-}OUT$, $E{-}OPEN$, $E{-}IN$ and $E{-}CLOSE_L$ rules, since $a \notin \texttt{fn}(Q)$. Thus, $[P \mid Q] \xrightarrow{\tau} [\underline{\texttt{new}}\, a\, (a(y) \mid \overline{a}\langle x\rangle)]$. Consequently, we see on the one hand that the side condition within the $E - CLOSE_L$ (respectively $E - CLOSE_R$) rule is essential to prevent communication between processes without possible scope extrusion and on the other hand that there is a need for a visible transition for a process sending a bound name. Concluding, we see the necessity of the omission of the restriction within the conclusion of the $E - OPEN$ rule and the reintroduction within the $E - CLOSE_L$ (respectively $E - CLOSE_R$) rule. Otherwise, no scope extrusion would be possible. But if we consider, for example, a similar handling of the restriction operator as in the $E - RES$ and $E - COM_L$ (respectively $E - COM_R$) rules, $[P \mid Q]$ would reach $[\underline{\texttt{new}}\, b\, (b(y)) \mid \overline{a}\langle x\rangle]$ which has a significant different behavior.

With the intuition that the visible transitions describe the communication possibilities for a parallel composed process, we gain another hint for the omission and reintroduction of the restriction. We already saw for the processes P and Q from above that $[P \mid Q] \xrightarrow{\tau} [\underline{\texttt{new}}\, a\, (a(y) \mid \overline{a}\langle x\rangle)]$ and so $[P \mid Q] \xrightarrow{\tau} [\underline{\texttt{new}}\, a\, (a(y) \mid \overline{a}\langle x\rangle)] \xrightarrow{\tau} [\underline{\texttt{new}}\, a\, (\mathbf{0} \mid \mathbf{0})]$ holds. Since $[Q] \xrightarrow{x\, a} [\overline{a}\langle x\rangle] \xrightarrow{\overline{a}\langle x\rangle} [\mathbf{0}]$, it fits that $[P] \xrightarrow{\overline{x}(a)} [a(y)] \xrightarrow{a\, x} [\mathbf{0}]$ holds. By not omitting the restriction, the process would stuck in a deadlock ($[\underline{\texttt{new}}\, a\, a(y)]$) after conducting the first transition. Furthermore, we know that we do not create unwanted behavior with this omission, since the only rules which handle a bound output are the $E - SUM_L$ (respectively $E - SUM_R$), $E - CLOSE_L$ (respectively $E - CLOSE_R$) and the $E - PAR_L$ (respectively $E - PAR_R$) rules. Whereby, for the summation rules this policy does not have any relevance, since

only one part will be chosen and the other is rendered void. Within the closing rules the resulting process is again restricted by the same bound name. Thus, this is just the scope extrusion and all unintended behavior is prevented by its side condition. Within the $E-PAR_L$ and the $E-PAR_R$ rule, which handle the interleaving behavior of both processes, the unwanted behavior is also excluded by the side condition. For example, consider $P =_{\text{def}} \underline{\texttt{new}}\, a\ \overline{b}\langle a\rangle.a(b)$ and so $[P] \xrightarrow{\overline{b}(a)} [a(b)]$. Without the side condition we could reach $[a(b) \mid \overline{a}\langle b\rangle]$ with the bound output action $\overline{b}(a)$ from $\left[\underline{\texttt{new}}\, d\left(\overline{b}\langle d\rangle.d(b)\right) \mid \overline{a}\langle b\rangle\right]$. Thus, we would identify a bound name with a free one.

Names and substitution

A closer look upon the $E - OPEN$ rule yields that if there is a bound output transition, we can apply the $E-OPEN$ rule for every process within the equivalence class. Hence, for every process constructed through a replacement of the restricted name of the process with a fresh one, there is also a bound transition with the new name bound in its label. This context is written down in Lemma 2.2.2 which is proven by Sangiorgi and Walker in [SW01].

Lemma 2.2.2 (Bound transition [SW01]) *Given names $a, b \in \mathcal{N}$ and processes $P, Q \in \mathcal{P}^\pi$, then*

$$[P] \xrightarrow{\overline{a}(b)} [Q] \text{ and } z \notin \texttt{fn}(\underline{\texttt{new}}\, b\ P) \text{ implies } [P] \xrightarrow{\overline{a}(z)} [Q\,\{z/b\}]$$

holds. ♢

Since we now know that we can replace a bound name in an action by every name which is not free in the process the transition is starting from, we can extend Convention 2.2.1 for actions.

Convention 2.2.2 (Uniqueness of bound names and transitions) *We extend Convention 2.2.1 to actions. Thus, we additionally stipulate that the bound names of some actions under consideration are different from the free names of the processes, the free names of the other actions and the names of the substitutions under consideration. Thereby, we need the limitation that for $[P] \xrightarrow{\overline{a}(z)} [Q]$ the name z, which is bound within the action and may be bound in P, is also allowed to be free in Q. Otherwise, scope extrusion would not be possible.* ∘

We collect some more facts proved in [SW01], which simplify the treatment of transitions and will be lifted to a chain of transitions in Chapter 3. We start by

investing a connection between transitions and the free names of its processes.

Lemma 2.2.3 (Transitions and free names [SW01]) *Given names $a, b \in \mathcal{N}$, an action $\alpha \in \mathtt{Act}$ and processes $P, Q \in \mathcal{P}^\pi$ with $[P] \xrightarrow{\alpha} [Q]$, then*

$$\begin{array}{llll} \alpha = \tau & \textit{implies} & \mathtt{fn}(Q) \subseteq \mathtt{fn}(P), & \text{(TAU)} \\ \alpha = a\,b & \textit{implies} & a \in \mathtt{fn}(P) \textit{ and } \mathtt{fn}(Q) \subseteq \mathtt{fn}(P) \cup \{b\}, & \text{(INP)} \\ \alpha = \overline{a}\langle b \rangle & \textit{implies} & a, b \in \mathtt{fn}(P) \textit{ and } \mathtt{fn}(Q) \subseteq \mathtt{fn}(P), & \text{(OUT)} \\ \alpha = \overline{a}(b) & \textit{implies} & a \in \mathtt{fn}(P) \textit{ and } \mathtt{fn}(Q) \subseteq \mathtt{fn}(P) \cup \{b\}, & \text{(BOUT)} \end{array}$$

holds. ♢

Furthermore, we know that a transition between two processes implies that there is also a transition where an arbitrary substitution is applied to all of its components.

Lemma 2.2.4 (Substitution on transitions (Part I) [SW01]) *Given processes $P, Q \in \mathcal{P}^\pi$ and a substitution σ, then*

$$\textit{if } [P] \xrightarrow{\alpha} [Q] \textit{ then } [P\sigma] \xrightarrow{\sigma(\alpha)} [Q\sigma]$$

holds. ♢

Proof: It is proved by an induction over the inference of $[P] \xrightarrow{\alpha} [Q]$. All cases apart from the $E - CALL$ case are proved in [SW01]. Thus, let $P, Q \in \mathcal{P}^\pi$, σ a substitution and $[P] \xrightarrow{\alpha} [Q]$ inferred by the $E - CALL$ rule. So we know that $\alpha = \tau$ and there is a recursive definition $A(\vec{w}) =_{\text{def}} P'$ and some parameter $\vec{v} \subseteq \mathcal{N}$ such that $P = A\langle \vec{v} \rangle$ and $Q = P'\{\vec{v}/\vec{w}\}$. The definition of the application of a substitution to a recursive call yields that $A\langle \vec{v} \rangle \sigma = A\langle \vec{v}\sigma \rangle$. Hence, with the $E - CALL$ rule we know $[A\langle \vec{v} \rangle \sigma] \xrightarrow{\tau} [P'\{\vec{v}\sigma/\vec{w}\}]$. Since $\mathtt{fn}(P') \subseteq \vec{w}$ as stipulated, we know that $\mathtt{fn}(P'\{\vec{v}/\vec{w}\}) \subseteq \vec{v}$ and so its the same if we apply the substitution to the parameters or to the resulting process. Hence, $[A\langle \vec{v} \rangle \sigma] \xrightarrow{\tau} [(P'\{\vec{v}/\vec{w}\})\,\sigma]$. ■

Note that Convention 2.2.2 is important for this lemma. Consider, for example, $P =_{\text{def}} \mathtt{new}\, a\ \overline{x}\langle a \rangle.\overline{x}\langle a \rangle$. Then $[P] \xrightarrow{\overline{x}(b)} [\overline{x}\langle b \rangle]$ and for a substitution $\sigma =_{\text{def}} \{z/b\}$ Lemma 2.2.4 would yield that $[P] = [P\sigma] \xrightarrow{\overline{x}(b)} [\overline{x}\langle b \rangle \sigma] = [\overline{x}\langle z \rangle]$, since $\sigma(\overline{x}(b)) = \overline{x}(b)$. But such a transition leading to $[\overline{x}\langle z \rangle]$ does not exist. Without Convention 2.2.2 we would need a side condition that in such cases b is neither allowed to be within the free names of $P\sigma$ nor within the names of the substitution.

The converse of Lemma 2.2.4 does not hold. That is, for $[P\sigma] \xrightarrow{\beta} [Q']$ we are not always able to find an action $\alpha \in \mathtt{Act}$ and a process $Q \in \mathcal{P}^\pi$ such that $[P] \xrightarrow{\alpha} [Q]$ with $\sigma(\alpha) = \beta$ and $[Q\sigma] = [Q']$.

Consider, for example, $P =_{\text{def}} a(x) \mid \overline{b}\langle y\rangle$ and $\sigma =_{\text{def}} \{^a/_b\}$. Then, on the one hand $P\sigma = a(x).\mathbf{0} \mid \overline{a}\langle y\rangle.\mathbf{0}$ holds and so $[P\sigma] \xrightarrow{\tau} [Q\sigma]$ with $Q =_{\text{def}} \mathbf{0} \mid \mathbf{0}$. But on the other hand there is no τ transition starting in $[P]$. This is a problem as long as the application of a substitution inserts a name, which already occurs free in the process and this free name is not also replaced by the application. In this situation new possibilities for communication are established.

Furthermore, consider $P =_{\text{def}} y(a).\overline{x}\langle a\rangle$ and $\sigma =_{\text{def}} \{^b/_x\}$. Hence, we know that $[P\sigma] = \left[y(a).\overline{b}\langle a\rangle\right] \xrightarrow{y\,x} \left[\overline{b}\langle x\rangle\right]$ holds, but there is no way to find an action α and a process Q such that $[P] \xrightarrow{\alpha} [Q]$ with $\{^b/_x\}(\alpha) = y\,x$ and $[Q\sigma] = \left[\overline{b}\langle x\rangle\right]$ holds, since x has to be free in $\{^b/_x\}(\alpha)$ and $[Q\sigma]$. This is a problem every time an input process receives a name from $\mathtt{supp}(\sigma) \setminus \mathtt{cosupp}(\sigma)$, since then this name will be replaced in the label of the transition by the application of the substitution and there is no possibility to reintegrate it, if it is not in the co-support of σ.

The first problem can be handled by adding the side condition that the substitution is *injective* on the free names of the process under consideration. This means, for all $x, y \in \mathtt{fn}(P)$ if $x \neq y$ then $\sigma(x) \neq \sigma(y)$. Therewith, all the cases where new communications can be established due to the application of the substitution will be excluded.

The second problem is no problem in the view of the behavior of the process, since neither more communication is created nor destroyed from the substitution. The application of the substitution has just a problematic effect to the label of the transitions and the resulting process. We only can not find a suitable action for the transition, or a suitable process, since the application of the substitution replaces the needed name. Therefore, we can find a new substitution which preserves the behavior of the given substitution (behaves equal on the free names) but solves the problem with the input names, by adding those names to the substitution for reintegrating them. Thus, for example, a single substitution is getting a transposition.

This leads us to the restricted converse of Lemma 2.2.4.

Lemma 2.2.5 (Substitution on transitions (Part II) [SW01]) *Given a process $P \in \mathcal{P}^\pi$ and a substitution σ, which is injective on $\mathtt{fn}(P)$. Then, there is a bijection $\rho : (\mathtt{fn}(P)\sigma \setminus \mathtt{fn}(P)) \to (\mathtt{fn}(P) \setminus \mathtt{fn}(P)\sigma)$ and with that a bijective sub-*

stitution θ*, with*

$$\theta(x) = \begin{cases} \sigma(x) & \text{if } x \in \mathtt{fn}(P) \\ \rho(x) & \text{if } x \in \mathtt{fn}(P)\sigma \setminus \mathtt{fn}(P) \\ x & \text{if } x \notin \mathtt{fn}(P)\sigma \cup \mathtt{fn}(P) \end{cases}$$

such that $[P\sigma] = [P\theta]$ *and*

$$[P\theta] \xrightarrow{\beta} [Q'] \text{ implies } \exists \alpha \in \mathtt{Act}, Q \in \mathcal{P}^\pi : [P] \xrightarrow{\alpha} [Q]$$

with $\theta(\alpha) = \beta$ *and* $[Q\theta] = [Q']$. ♢

Proof: The second part is proved by induction over the inference of $[P\theta] \xrightarrow{\beta} [Q']$. All cases apart from the $E - CALL$ case as well as the existence of ρ are proved in [SW01]. Thus, let σ be a substitution, θ as described in Lemma 2.2.5 and $[P\theta] \xrightarrow{\beta} [Q']$ be inferred by the $E - CALL$ rule. Hence, $\beta = \tau$ and there is a recursive definition $A(\vec{w}) =_{\text{def}} P'$ and a parameter list $\vec{v} \subseteq \mathcal{N}$ such that $P\theta = A\langle\vec{v}\rangle$ and $Q' = P'\{\vec{v}/\vec{w}\}$. The application of a substitution to a process yields that the only way to reach $P\theta = A\langle\vec{v}\rangle$ is if there is a parameter list $\vec{v'} \subseteq \mathcal{N}$ such that $P = A\langle\vec{v'}\rangle$ and $\vec{v} = \vec{v'}\theta$. With the $E - CALL$ rule we know that $[P] \xrightarrow{\tau} \left[P'\left\{\vec{v}/\vec{w}\right\}\right]$. Since $\mathtt{fn}(P') \subseteq \vec{w}$ and so $\mathtt{fn}(P'\left\{\vec{v}/\vec{w}\right\}) \subseteq \vec{v'}$ and Convention 2.2.1 states that no bound names of the process can occur in the substitution, we know $\left[\left(P'\left\{\vec{v}/\vec{w}\right\}\right)\theta\right] = \left[P'\left\{\vec{v}\theta/\vec{w}\right\}\right] = [Q']$. So we found an action $\alpha = \tau$ with $\theta(\alpha) = \beta$ and a process $Q = P'\left\{\vec{v}/\vec{w}\right\}$ with $[Q\theta] = [Q']$ and $[P] \xrightarrow{\alpha} [Q]$. ■

We can think of the bijection ρ as the function which gives the possibility to reintegrate the names which are replaced by σ. So, for the example above with $P =_{\text{def}} y(a).\overline{x}\langle a\rangle$ and $\sigma =_{\text{def}} \{b/x\}$, we know $\mathtt{fn}(P) = \{y, x\}$ and $\mathtt{fn}(P)\sigma = \{y, b\}$. Hence, we get a bijection $\rho : \{b\} \to \{x\}$ and so $\theta = \{b \leftrightarrow x\}$. Thus, we know there is an action $\alpha =_{\text{def}} y\,b$ and a process $Q =_{\text{def}} \overline{x}\langle b\rangle$ such that $[P] \xrightarrow{\alpha} [Q]$ with $\{b \leftrightarrow x\}(\alpha) = y\,x$ and $[Q\{b \leftrightarrow x\}] = \left[\overline{b}\langle x\rangle\right]$. With this idea, we consider the special case of Lemma 2.2.5 where σ is a transposition.

Corollary 2.2.1 (Substitution on transitions (Part III)) *Given* $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ *and a transposition* $\sigma = \{a \leftrightarrow b\}$*, then*

$$[P\sigma] \xrightarrow{\beta} [Q'] \text{ implies } \exists \alpha \in \mathtt{Act}, Q \in \mathcal{P}^\pi : [P] \xrightarrow{\alpha} [Q]$$

with $\{a \leftrightarrow b\}(\alpha) = \beta$ *and* $[Q\{a \leftrightarrow b\}] = [Q']$. □

Proof: Let $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ and $\sigma = \{a \leftrightarrow b\}$ a transposition. We know a transposition is injective and so in particular injective on $\mathtt{fn}(P)$ and thus, the precondition of Lemma 2.2.5 is fulfilled.

Case $a, b \notin \mathtt{fn}(P)$**:** Let $\beta \in \mathtt{Act}, Q' \in \mathcal{P}^\pi$ with $[P\{a \leftrightarrow b\}] \xrightarrow{\beta} [Q']$. Since $a, b \notin \mathtt{fn}(P)$ we know $[P\{a \leftrightarrow b\}] = [P]$ and thus, $[P] \xrightarrow{\beta} [Q']$. Let $\alpha =_{\text{def}} \{a \leftrightarrow b\}(\beta)$ and $Q =_{\text{def}} Q'\{a \leftrightarrow b\}$, with Lemma 2.2.4 we know $[P\{a \leftrightarrow b\}] \xrightarrow{\{a \leftrightarrow b\}(\beta)} [Q'\{a \leftrightarrow b\}]$ and so, $[P] \xrightarrow{\alpha} [Q]$ with $\{a \leftrightarrow b\}(\alpha) = \beta$ and $[Q\{a \leftrightarrow b\}] = [Q']$.

Case $a \in \mathtt{fn}(P)$**,** $b \in \mathtt{fn}(P)$**:** Since in this case $\mathtt{fn}(P)\{a \leftrightarrow b\} \setminus \mathtt{fn}(P) = \emptyset$ and $\mathtt{fn}(P) \setminus \mathtt{fn}(P)\{a \leftrightarrow b\} = \emptyset$ holds, we know from Lemma 2.2.5 that there is a bijection θ with

$$\theta(x) = \begin{cases} \{a \leftrightarrow b\}(x) & \text{if } x \in \mathtt{fn}(P) \\ x & \text{else} \end{cases}$$

$[P\{a \leftrightarrow b\}] = [P\theta]$ and

$$[P\theta] \xrightarrow{\beta} [Q'] \text{ implies } \exists \alpha \in \mathtt{Act}, Q \in \mathcal{P}^\pi : [P] \xrightarrow{\alpha} [Q]$$

with $\theta(\alpha) = \beta$ and $[Q\theta] = [Q']$. Since $a, b \in \mathtt{fn}(P)$ we know $\theta = \{a \leftrightarrow b\}$ and thus, we found everything we need.

Case $a \in \mathtt{fn}(P)$**,** $b \notin \mathtt{fn}(P)$**:** Thus, we know $\mathtt{fn}(P)\{a \leftrightarrow b\} \setminus \mathtt{fn}(P) = \{b\}$ and $\mathtt{fn}(P) \setminus \mathtt{fn}(P)\{a \leftrightarrow b\} = \{a\}$ holds. Since there is just one bijection $\rho : \{b\} \to \{a\}$, we know from Lemma 2.2.5 that there is a bijection θ with

$$\theta(x) = \begin{cases} \{a \leftrightarrow b\}(x) & \text{if } x \in \mathtt{fn}(P) \\ a & \text{if } x = b \\ x & \text{else} \end{cases}$$

$[P\{a \leftrightarrow b\}] = [P\theta]$ and

$$[P\theta] \xrightarrow{\beta} [Q'] \text{ implies } \exists \alpha \in \mathtt{Act}, Q \in \mathcal{P}^\pi : [P] \xrightarrow{\alpha} [Q]$$

with $\theta(\alpha) = \beta$ and $[Q\theta] = [Q']$. We know $\theta = \{a \leftrightarrow b\}$, since $a \in \mathtt{fn}(P)$ and $b \notin \mathtt{fn}(P)$ and thus, in this case, the corollary holds.

Case $a \notin \mathtt{fn}(P)$**,** $b \in \mathtt{fn}(P)$**:** The argumentation is similar to the prior case by interchanging the roles of the names a and b.

Thus, for all cases we found the suitable action $\alpha \in \mathtt{Act}$ and process $Q \in \mathcal{P}^\pi$ with the claimed properties. ■

Furthermore, we can similarly apply Lemma 2.2.5 to a single substitution, if the inserted name do not occur free in the process under consideration.

Corollary 2.2.2 (Substitution on transitions (Part IV)) *Given* $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ *and* $\sigma = \{^a/_b\}$ *substitution with* $a \notin \mathtt{fn}(P)$ *then* $[P\sigma] = [P\{a \leftrightarrow b\}]$ *and*

$$[P\sigma] \xrightarrow{\beta} [Q'] \text{ implies } \exists \alpha \in \mathtt{Act}, Q \in \mathcal{P}^\pi : [P] \xrightarrow{\alpha} [Q]$$

with $\{a \leftrightarrow b\}(\alpha) = \beta$ *and* $[Q\{a \leftrightarrow b\}] = [Q']$. □

Proof: Since $a \notin \mathtt{fn}(P)$ and $\sigma = \{^a/_b\}$, we know that $[P\sigma] = [P\{a \leftrightarrow b\}]$. We can now similarly prove this corollary to Corollary 2.2.1, since ρ extends σ such that for the θ of Lemma 2.2.5 we know $\theta = \{a \leftrightarrow b\}$ holds. ■

So, we gain the same results for on the one hand a transposition and on the other hand a substitution which only replaces one name and the co-support is no subset of the free names of the process under consideration.

Example

As examples for the π-calculus and its semantics, we regard a system which can read and write values of and to a buffer.

Example 2.2.1 (One cell buffer [Mil99]) We can define a buffer with just one cell, for example through B_1.

$$B_1(in, out) =_{\text{def}} in(val).O_1\langle val, in, out\rangle$$
$$O_1(val, in, out) =_{\text{def}} \overline{out}\langle val\rangle.B_1\langle in, out\rangle$$

The buffer has one value *val* and two channels *in* and *out* to write respectively read the value. Since the buffer uses recursive calls, it is always ready for a new pass, after one pass of reading and writing. Its semantics is represented in Figure 2.4. ∗

Since for every input there are countably infinite names which can be received, we use dots to visualize such alternatives in Figure 2.4.

To extend the buffer so that it is possible to read and write more than one value, we introduce two versions of a buffer with two cells.

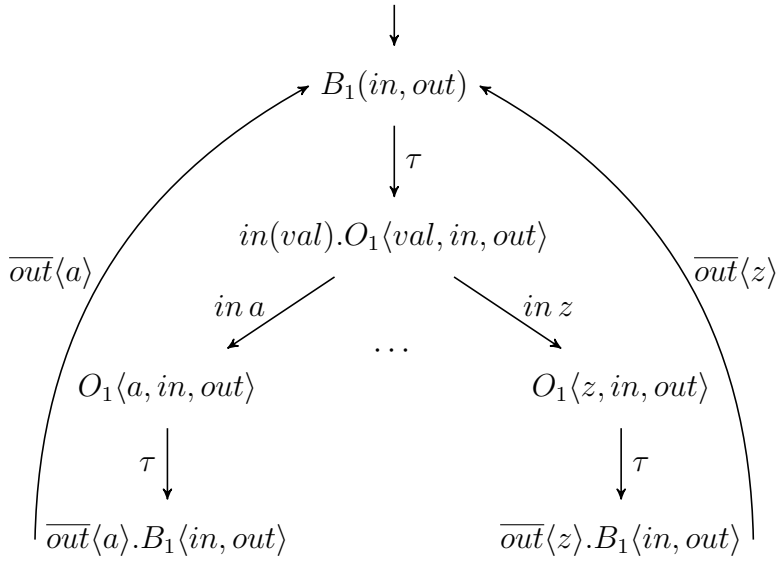

Figure 2.4: The operational semantics for buffer B_1 from Example 2.2.1.

Example 2.2.2 (Two cell buffer (Version I)) A buffer with two cells and some kind of FIFO strategy can possibly be modeled as B_2.

$$
\begin{aligned}
B_2(i,o) &=_{\text{def}} C_1\langle i,o\rangle + C_2\langle i,o\rangle \\
C_1(i,o) &=_{\text{def}} i(val_1).i(val_2).\overline{o}\langle val_1\rangle.\overline{o}\langle val_2\rangle.B_2\langle i,o\rangle \\
C_2(i,o) &=_{\text{def}} i(val_1').\overline{o}\langle val_1'\rangle.i(val_2').\overline{o}\langle val_2'\rangle.B_2\langle i,o\rangle
\end{aligned}
$$

This buffer has also an input i and an output o channel for writing respectively reading values. The semantics of B_2 is visualized in Figure 2.5. ∗

Since $i(y).\overline{o}\langle a\rangle.\overline{o}\langle y\rangle.B_2\langle i,o\rangle$ and $i(y).\overline{o}\langle z\rangle.\overline{o}\langle y\rangle.B_2\langle i,o\rangle$ are the same, apart from the name that had been received and this name has no influence on the communication of the process, we abbreviate the second part by the dots in Figure 2.5. Furthermore, we know that in the second part just the first output transition is changed from $\overline{o}\langle a\rangle$ to $\overline{o}\langle z\rangle$.

It is also possible to define a FIFO buffer with two cells as a linked list.

Example 2.2.3 (Two cell buffer (Version II)[Mil99]) A FIFO buffer with two cells can be defined with $FIFO$.

$$
\begin{aligned}
FIFO(in,out) &=_{\text{def}} \underline{\texttt{new}}\, com\, (B_3\langle in, com\rangle \mid B_3\langle com, out\rangle) \\
B_3(in,out) &=_{\text{def}} in(val).O_2\langle val, in, out\rangle
\end{aligned}
$$

$$O_2(val, in, out) =_{\text{def}} \overline{out}\langle val\rangle.B_3\langle in, out\rangle$$

This buffer can write the first value in one cell and, before receiving another one, it passes this value via the private channel *com* to the other cell. A reading is just possible if the value is located in the second cell. ∗

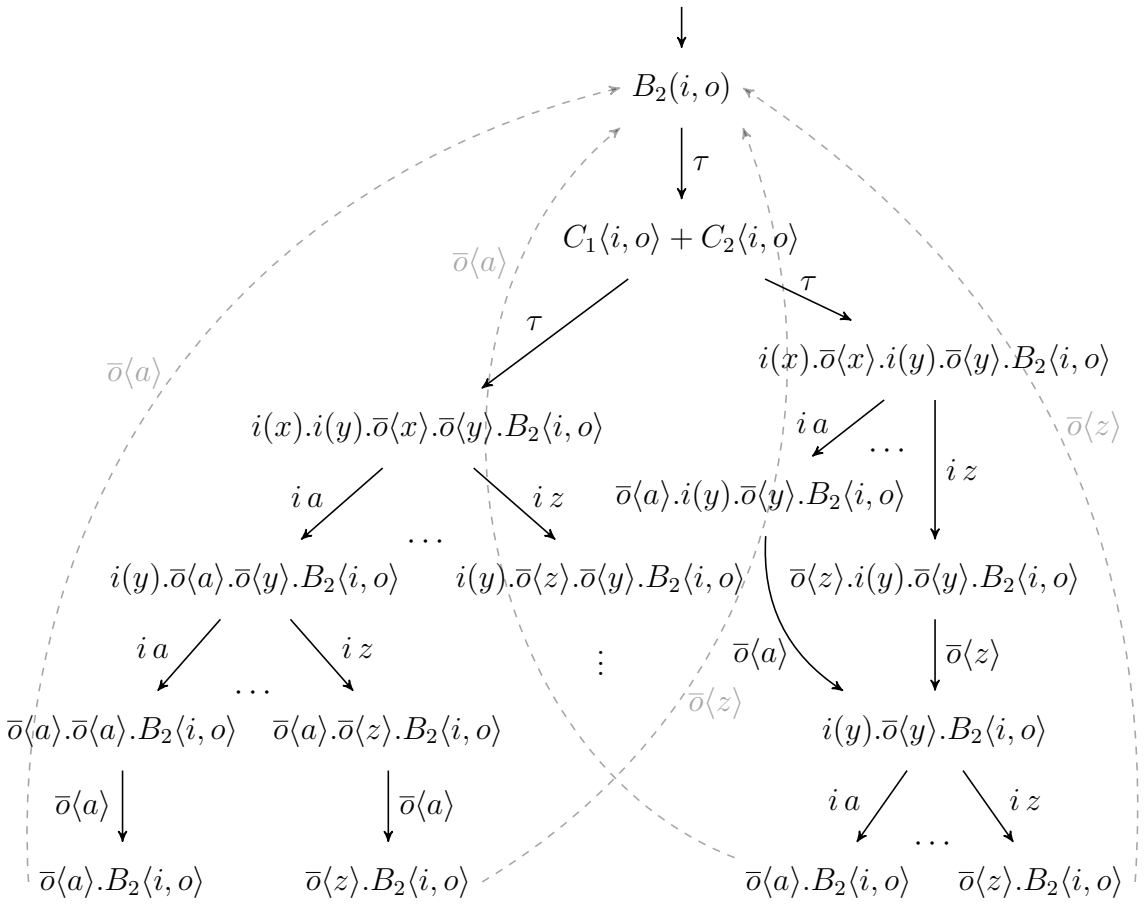

Figure 2.5: The operational semantics for buffer B_2 from Example 2.2.2.

Since the behavior of those processes – especially of the $FIFO$ process – is hard to determine, we introduce in Section 4 a denotational semantics, which simplifies the calculation of the external behavior of a process. Furthermore, we introduce a relation which enables comparing processes on the base of the denotational semantics. Thereby, we will see, that the buffer B_2 and the buffer $FIFO$ have a different visible behavior.

2.2.5 Simulation and bisimulation

For comparing processes based on their behavior there is already a notion called *strong bisimulation* defined in [MPW92]. Since they use a slightly different semantics for the transitions, we stick to the notion of bisimulations in [SW01].

Definition 2.2.11 (Strong simulation and bisimulation) A relation $\mathcal{S} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ is called a *strong simulation*, if $([P], [Q]) \in \mathcal{S}$ implies that

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

$\mathcal{S}$ is called a *strong bisimulation* if $\mathcal{S}$ and $\mathcal{S}^{-1}$ are strong simulations.

Furthermore, for a strong bisimulation $\mathcal{S}$, we call $[P]$ *strong bisimilar* to $[Q]$, written as $[P] \sim [Q]$, if $([P], [Q]) \in \mathcal{S}$. ∗

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.

Definition 2.2.12 (Weak simulation and bisimulation) A relation $\mathcal{S} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ is called a *weak simulation*, if $([P], [Q]) \in \mathcal{S}$ implies

(1) if $[P] \xrightarrow{\tau} [P']$ then $Q' \in \mathcal{P}^\pi$ exists such that $[Q] \, (\xrightarrow{\tau})^* \, [Q']$ and $([P'], [Q']) \in \mathcal{S}$,

(2) if $[P] \xrightarrow{\alpha} [P']$ then $Q' \in \mathcal{P}^\pi$ exists such that $[Q] \, (\xrightarrow{\tau})^* \fatsemi \xrightarrow{\alpha} \fatsemi (\xrightarrow{\tau})^* \, [Q']$ and $([P'], [Q']) \in \mathcal{S}$, for $\alpha \in \mathtt{Act} \setminus \{\tau\}$.

A relation $\mathcal{S} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ is called a *weak bisimulation*, if $\mathcal{S}$ and $\mathcal{S}^{-1}$ are weak simulations.

Furthermore, for a weak bisimulation $\mathcal{S}$, we call $[P]$ *weak bisimilar* to $[Q]$, written as $[P] \approx [Q]$, if $([P], [Q]) \in \mathcal{S}$. ∗

Note that with the definition of the big-step semantics in Chapter 3 there is a somehow simpler way to describe the weak bisimulation, since there is an abbreviation for an arbitrary number of τ transitions.

2.3 Discussion

Many operational semantics and variants of the syntax of the π-calculus exist. The reasons for choosing this syntactical variant of the π-calculus are already explained in Section 2.2. In this section we motivate the usage of the particular operational semantics defined in Section 2.2.4 and compare it to others.

In [MPW92] Milner, Parrow, and Walker introduce an operational semantics with *late instantiation*. That is, the variable of an input process is instantiated at the moment the communication is inferred. This contrasts to the *early instantiation*, where the variable is already instantiated when the input transition is inferred, which we consider in this thesis. The late instantiation constitutes the reason for the inapplicability of this operational semantics for our desired denotational semantics. The notion of the trace semantics defined in Chapter 4 is to collect all of a process' behavior just by observing the labels of the transitions. Consider, for example, the process $P' =_{\text{def}} x(z).\overline{z}\langle q\rangle$. The only rule which handles the external behavior of an input in the operational semantics presented in [MPW92], is the axiom

$$\underline{INPUT\text{–}ACT}: \quad \frac{}{[x(z).P] \xrightarrow{x\,w} [P\,\{^{w}/_{z}\}]} \quad w \notin \mathtt{fn}(\underline{\mathtt{new}}\, z\ P).$$

Thus, there is no transition $[P'] \xrightarrow{x\,q} [\overline{q}\langle q\rangle]$, which however is also a valid external behavior of P'. This is no problem for the internal behavior of the process, since for a communication – according to the late instantiation – the missing input variables are instantiated by the communication rule. But the single transition itself is missing, which makes it impossible to describe the whole external and internal behavior by simply observing the labels of the transition system. Note that these missing transitions are harmful for describing the behavior of a process. Thus, in the definition of bisimilarity in [MPW92] they also took this missing behavior into account while relating processes.

In [Mil99] Robin Milner defines two other operational semantics. On the one hand, the *commitments* developed for another definition of the strong bisimilarity, and on the other hand, the *input-/output experiments* to gain a better handling of the internal behavior of processes to present another definition of the weak bisimilarity. Both are not suitable for the desired denotational semantics, since they also do not represent the whole behavior of the process in the transition system itself. For instance, the commitment rules are deadlocked for one part of a sum after conducting a simple input or output transition. Hence, there is a commitment rule $a(x).x(y) \xrightarrow{a}$

$(x).x(y)$, but there is no further rule for a so-called agent $(x).x(y)$. Thus, the information that the process can interact with another process by synchronizing over channel a and, after that, synchronizing with another output process over this name, is lost by just observing the transitions.

Another problem is that for an output process $P =_{\text{def}} \overline{a}\langle x\rangle.\mathbf{0}$ the name x, which is sent over channel a, is not specified in the label of the transition $\overline{a}\langle x\rangle.\mathbf{0} \xrightarrow{\overline{a}} \langle x\rangle.\mathbf{0}$. Thus, this information gets also lost by just observing the label of the transitions. So, the transitions of P and the process $\overline{a}\langle y\rangle.\mathbf{0}$ are the same, which is not suitable for our trace semantics.

The input-/output experiments presented in [Mil99] base upon the commitment rules. In this approach, input transitions are extended such that the problems mentioned above will not occur. Even though, the output transitions are also extended, the problems concerning output processes still exists. Thus, not the whole behavior of a process is mapped to its transitions.

The motivation to take a closer look at the transition systems defined by Sangiorgi and Walker in [SW01] is given by their statement, in which they explain that they did not adapt the notion of Milner "because [they] consider the presentation [they] have given to be simpler and easier to work with"[5]. Moreover, Milner himself rates the book in the foreword "as a storehouse of ideas and techniques which is unlikely to be equalled in the next decade or two"[6].

It is a matter of interest and the field of application, whether the *late* or *early transition system* presented in [SW01] is chosen for formalizing the semantics. This is because every transition of the early can also be made, with a slight adaption, in the late transition system and vice versa [SW01]. However this adaption has to be taken into account while working with the semantics. It is for this reason that the late semantics is not suitable for a denotational semantics, which just observes the labels of the transitions in order to gain the whole behavior of a process. Even though the input rule in the late transition system differs from the rule in [MPW92]:

$$\underline{L\text{-}INP}: \ \frac{}{[x(z).P] \xrightarrow{x\,z} [P]},$$

the problem stays the same. Even if the side condition and the substitution within the INPUT-ACT can be omitted in [SW01] by the reason of a similar convention as Convention 2.2.1, there is also no transition $[x(z).\overline{z}\langle q\rangle] \xrightarrow{x\,q} [\overline{q}\langle q\rangle]$, since $x(q).\overline{q}\langle q\rangle \notin$

[5] [SW01], page 162.
[6] [SW01], page x.

$[x(z).\overline{z}\langle q\rangle]$.

Since the inapplicability of the semantics defined in [MPW92, SW01] results by the late instantiation of the variables of an input process, the more intensively investigation of other transition systems with late instantiation (for example in [CPT01, Qua96, AC12]) is not expedient.

3 Big-step semantics

For retrieving the whole behavior of a process, we investigate the transition system produced by the rules of Figure 2.3. In particular, we follow the way of the transitions and collect their labels. Thus, some kind of transition-chains – so-called *traces* – are produced. In this chapter we define a big-step semantics by stringing together the steps of the early semantics as well as the formal meaning of traces. The ideas of the definition of the big-steps and traces base upon the trace model for CSP introduced by Hoare in [Hoa80] and [Hoa85].

Additionally, in this chapter some of the definitions and properties explained in Section 2.2 are lifted to traces, since the early transition system is at the heart of the big-step semantics.

3.1 Definition

Since we would like to follow a chain of transitions and its labels through the transition system, we define a formal construct which collects actions in an ordered way. Moreover, we abstract from the internal behavior. Thus, we are only collecting visible actions.

Definition 3.1.1 (Traces) We define a set containing all sequences of external actions by

$$\texttt{Traces} =_{\text{def}} \texttt{seq}(\texttt{Act} \setminus \{\tau\})$$

and call each of its elements a *trace*. In particular, we call $\langle\rangle \in \texttt{Traces}$ the *empty trace*. $*$

Thus, with a trace, it is possible to describe the progression of external actions. In the big-step semantics this is combined with the operational semantics to easily notate by which external actions a process can evolve to another.

Definition 3.1.2 (Big-step semantics) We inductively define the *big-step semantics* $\left\{\stackrel{s}{\Rightarrow} \mid \ s \in \texttt{Traces}\right\} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ for processes $P, Q \in \mathcal{P}^\pi$ by

$$[P] \stackrel{\langle\rangle}{\Rightarrow} [Q]\,, \text{ iff } [P]\,(\xrightarrow{\tau})^*\,[Q]\,.$$

As a shorter notion it is also possible to write $[P] \Rightarrow [Q]$.

$$[P] \stackrel{\langle\alpha\rangle}{\Longrightarrow} [Q]\,, \text{ iff } [P] \stackrel{\langle\rangle}{\Rightarrow} ; \xrightarrow{\alpha} ; \stackrel{\langle\rangle}{\Rightarrow} [Q]$$

with $\alpha \in \texttt{Act} \setminus \{\tau\}$.

$$[P] \stackrel{s \frown t}{\Longrightarrow} [Q]\,, \text{ iff } [P] \stackrel{s}{\Rightarrow} ; \stackrel{t}{\Rightarrow} [Q]$$

with $s, t \in \texttt{Traces}$.

Furthermore, we call $\stackrel{\langle\rangle}{\Rightarrow}$ an *invisible big-step* and $\stackrel{t}{\Rightarrow}$ for a trace $t \in \texttt{Traces}$ a *visible big-step*. ∗

Thus, we now have the possibility to write $[a(x).\overline{x}\langle z\rangle \mid \overline{a}\langle y\rangle] \stackrel{\langle a\,b, \overline{a}\langle y\rangle, \overline{b}\langle z\rangle\rangle}{\Longrightarrow} [\mathbf{0} \mid \mathbf{0}]$, since $[a(x).\overline{x}\langle z\rangle \mid \overline{a}\langle y\rangle] \xrightarrow{a\,b} \left[\overline{b}\langle z\rangle \mid \overline{a}\langle y\rangle\right] \xrightarrow{\overline{a}\langle y\rangle} \left[\overline{b}\langle z\rangle \mid \mathbf{0}\right] \xrightarrow{\overline{b}\langle z\rangle} [\mathbf{0} \mid \mathbf{0}]$. Furthermore, we abbreviate $[a(x).\overline{x}\langle z\rangle \mid \overline{a}\langle y\rangle] \stackrel{\langle a\,b, \overline{a}\langle y\rangle\rangle}{\Longrightarrow} \left[\overline{b}\langle z\rangle \mid \mathbf{0}\right]$ and $\left[\overline{b}\langle z\rangle \mid \mathbf{0}\right] \stackrel{\langle\overline{b}\langle z\rangle\rangle}{\Longrightarrow} [\mathbf{0} \mid \mathbf{0}]$ by $[a(x).\overline{x}\langle z\rangle \mid \overline{a}\langle y\rangle] \stackrel{\langle a\,b, \overline{a}\langle y\rangle\rangle}{\Longrightarrow} \left[\overline{b}\langle z\rangle \mid \mathbf{0}\right] \stackrel{\langle\overline{b}\langle z\rangle\rangle}{\Longrightarrow} [\mathbf{0} \mid \mathbf{0}]$.

Note that $[P] \stackrel{t}{\Rightarrow} [P_1] \xrightarrow{\alpha} [Q]$ for some $\alpha \in \texttt{Act}$ and $[P] \stackrel{t}{\Rightarrow} [P_1']$ in general does not imply that $[P_1']$ must have a possibility for a transition labeled with α. Consider, for example, $P =_{\text{def}} \overline{a}\langle x\rangle + \overline{a}\langle x\rangle.\overline{b}\langle y\rangle$. Then $[P] \stackrel{\langle\overline{a}\langle x\rangle\rangle}{\Longrightarrow} \left[\overline{b}\langle y\rangle\right] \xrightarrow{\overline{b}\langle y\rangle} [\mathbf{0}]$ and $[P] \stackrel{\langle\overline{a}\langle x\rangle\rangle}{\Longrightarrow} [\mathbf{0}]$.

3.2 Names and binding

Since we already talked about objects and subjects of actions, it is useful to define functions which calculate them from the actions and especially from traces.

Definition 3.2.1 (Subjects and objects) We define $\texttt{sub} : \texttt{Act} \setminus \{\tau\} \to \mathcal{N}$ as the function which collects the *subjects* and $\texttt{obj} : \texttt{Act} \setminus \{\tau\} \to \mathcal{N}$ for the *objects* of an action such that

$$\begin{array}{lll} \texttt{sub}(\overline{a}\langle x\rangle) & =_{\text{def}} & a \\ \texttt{sub}(\overline{a}(x)) & =_{\text{def}} & a \\ \texttt{sub}(a\,x) & =_{\text{def}} & a \end{array} \qquad \begin{array}{lll} \texttt{obj}(\overline{a}\langle x\rangle) & =_{\text{def}} & x \\ \texttt{obj}(\overline{a}(x)) & =_{\text{def}} & x \\ \texttt{obj}(a\,x) & =_{\text{def}} & x \end{array}$$

holds. With those functions, we define $\mathtt{sub} : \mathtt{Traces} \to \mathbb{P}(\mathcal{N})$ with

$$\mathtt{sub}(s) =_{\text{def}} \{x \in \mathcal{N} \mid \exists \alpha \in s : x = \mathtt{sub}(\alpha)\},$$

which collects the subjects of a trace and $\mathtt{obj} : \mathtt{Traces} \to \mathbb{P}(\mathcal{N})$ with

$$\mathtt{obj}(s) =_{\text{def}} \{x \in \mathcal{N} \mid \exists \alpha \in s : x = \mathtt{obj}(\alpha)\},$$

which collects the objects. ∗

Furthermore, we define the *free* and *bound names* of a trace. This definition intuitively fits to the definition of free and bound names of processes in Section 2.2.2.

Definition 3.2.2 (Bound and free names of a trace) The *bound names* of a trace $t \in \mathtt{Traces}$ are defined by

$$\mathtt{bn}(t) =_{\text{def}} \bigcup_{\alpha \in t} \mathtt{bn}(\alpha).$$

Thus, all names are collected, which occur as an object of a bound output action within the trace. Furthermore, with

$$\mathtt{fn}(t) =_{\text{def}} \{n \in \mathtt{n}(t) \mid n \notin \mathtt{bn}(t)\}$$

we define the *free names* of a trace $t \in \mathtt{Traces}$ by the set of all the other names, which do not occur bound in t. Once more, the *names* of a transition $t \in \mathtt{Traces}$ are denoted by $\mathtt{n}(t) =_{\text{def}} \mathtt{fn}(t) \cup \mathtt{bn}(t)$. ∗

Like the connection between the operational semantics and the free names of its processes, there is also a connection between traces and the free names of its components.

Lemma 3.2.1 (Big-step semantics and free names) *Given processes $P, Q \in \mathcal{P}^{\pi}$ and a trace $t \in \mathtt{Traces}$, then*

$$[P] \stackrel{t}{\Rightarrow} [Q] \text{ implies } \mathtt{fn}(Q) \subseteq \mathtt{fn}(P) \cup \mathtt{n}(t)$$

holds. ♢

Proof: Let $P_1, P_n \in \mathcal{P}^{\pi}$ and $t \in \mathtt{Traces}$ with $[P_1] \stackrel{t}{\Rightarrow} [P_n]$ and $\#(t) = n - 1$ for some $n \in \mathbb{N}$. Hence, there are processes $P_2, \ldots, P_{n-1} \in \mathcal{P}^{\pi}$ such that $[P_1] \stackrel{\langle t_1 \rangle}{\Longrightarrow}$

$[P_2] \stackrel{\langle t_2 \rangle}{\Longrightarrow} \cdots \stackrel{\langle t_{n-2} \rangle}{\Longrightarrow} [P_{n-1}] \stackrel{\langle t_{n-1} \rangle}{\Longrightarrow} [P_n]$. For every τ step within the big-steps we know from Lemma 2.2.3 Equation (TAU) that the free names of the resulting process are a subset of the free names of the starting one. The same lemma also yields for every $i \in \{1, \ldots, n-1\}$ that $\mathtt{fn}(P_{i+1}) \subseteq \mathtt{fn}(P_i) \cup \mathtt{n}(t_i)$. Thus, $\mathtt{fn}(P_n) \subseteq \mathtt{fn}(P_{n-1}) \cup \mathtt{n}(t_{n-1}) \subseteq (\mathtt{fn}(P_{n-2}) \cup \mathtt{n}(t_{n-2})) \cup \mathtt{n}(t_{n-1}) \subseteq \cdots \subseteq \mathtt{fn}(P_1) \cup \mathtt{n}(t)$, since $\mathtt{n}(t) = \bigcup_{i \in \{1,\ldots,n-1\}} \mathtt{n}(t_i)$ holds. ■

Note that in particular the inclusion is an over-approximation. Consider, for instance, $P =_{\text{def}} a(b).\overline{c}\langle d \rangle \mid \overline{x}\langle y \rangle.z(w)$. Then, $[P] \stackrel{\langle a\, b, \overline{x}\langle y \rangle \rangle}{\Longrightarrow} [Q]$ with $Q =_{\text{def}} \overline{c}\langle d \rangle \mid z(w)$ and $\mathtt{fn}(Q) = \{c, d, z\} \subseteq \{a, c, d, x, y, z\} \cup \{a, b, x, y\} = \mathtt{fn}(P) \cup \mathtt{n}(t)$. Moreover, $[P] \stackrel{\langle a\, r, \overline{c}\langle d \rangle, \overline{x}\langle y \rangle, z\, s \rangle}{\Longrightarrow} [\mathbf{0} \mid \mathbf{0}]$ and so $\emptyset \subseteq \{a, c, d, x, y, z\} \cup \{a, r, c, d, x, y, z, s\}$.

Furthermore, it is helpful – especially for Lemma 4.2.9 – to define a function which replaces every occurrence of an output action in a trace by a bound output action for a given name.

Definition 3.2.3 (Binding) We define the *binding function* $\mathtt{bind} : \mathcal{N} \times \mathtt{Act} \to \mathtt{Act}$, with

$$\begin{array}{lll} \mathtt{bind}(a, \tau) & =_{\text{def}} & \tau \\ \mathtt{bind}(a, \overline{b}(x)) & =_{\text{def}} & \overline{b}(x) \\ \mathtt{bind}(a, b\, x) & =_{\text{def}} & b\, x \\ \mathtt{bind}(a, \overline{b}\langle x \rangle) & =_{\text{def}} & \begin{cases} \overline{b}(x) & \text{if } a = x \\ \overline{b}\langle x \rangle & \text{else} \end{cases} \end{array}$$

such that it only replaces an output action with its corresponding bound output action, if the given name is the object of the action.

Furthermore, for traces $\mathtt{bind} : \mathcal{N} \times \mathtt{Traces} \to \mathtt{Traces}$ is defined as

$$\mathtt{bind}(a, s) =_{\text{def}} \langle s'_1, s'_2, \ldots \rangle \text{ with } s'_i = \begin{cases} \mathtt{bind}(a, s_i) & \text{if } \nexists j \in \mathbb{N}\colon \quad j < i \wedge a \in \mathtt{obj}(s_j) \\ & \qquad\qquad \wedge s_j \in \mathtt{Out} \cup \mathtt{Bout} \\ s_i & \text{else} \end{cases}$$

for all $i \in \{1, 2, \ldots\}$. Moreover, we would like to bind a name in a whole set of traces such that $\mathtt{bind} : \mathcal{N} \times \mathbb{P}(\mathtt{Traces}) \to \mathbb{P}(\mathtt{Traces})$ is defined with

$$\mathtt{bind}(a, X) =_{\text{def}} \{s \in \mathtt{Traces} \mid \exists s' \in X : s = \mathtt{bind}(a, s')\}$$

as a function which binds a name to a whole trace set by applying the binding

function to every trace within the set. ∗

Note that just the first occurrence of the name as an object of an output action is replaced by the binding function. This is because by Convention 3.3.1 we will stipulate that every bound name within a trace is unique.

3.3 Substitution

Since we can replace names within processes and actions, we consequently define the application of a substitution on traces.

Definition 3.3.1 (Substitution on traces) The *application* of a substitution σ to a trace $t \in \mathtt{Traces}$ with $\mathtt{n}(\sigma) \cap \mathtt{bn}(t) = \emptyset$ is written $t\sigma \in \mathtt{Traces}$ and is defined as the application of the substitution to every action: $t\sigma =_{\text{def}} \{(i, \sigma(\alpha)) \mid (i, \alpha) \in t\}$. For a set of traces $X \subseteq \mathtt{Traces}$ we define the application of the substitution $X\sigma =_{\text{def}} \{t\sigma \mid t \in X\}$ as the application of the substitution to all elements of the set. ∗

With Lemma 2.2.4 we showed a relation between τ transitions and substitutions. In the following, we lift this lemma to the big-step semantics. Thus, we know if there is an invisible big-step between two processes, there is also an invisible big-step between the processes where a substitution has been applied to.

Lemma 3.3.1 (Substitution on big-step semantics (Part I)) *For all processes $P, Q \in \mathcal{P}^\pi$ and substitutions σ,*

$$\textit{if } [P] \overset{\langle\rangle}{\Rightarrow} [Q] \textit{ then } [P\sigma] \overset{\langle\rangle}{\Rightarrow} [Q\sigma]$$

holds. ♢

Proof: Let $P, Q \in \mathcal{P}^\pi$ and σ a substitution, then

$$\begin{array}{lll}
[P] \overset{\langle\rangle}{\Rightarrow} [Q] & \overset{\{\text{Definition 3.1.2}\}}{\text{impl.}} & [P] \left(\xrightarrow{\tau}\right)^* [Q] \\
& \text{impl.} & \exists Q_1, \ldots, Q_n \in \mathcal{P}^\pi : [P] \xrightarrow{\tau} [Q_1] \wedge [Q_1] \xrightarrow{\tau} [Q_2] \\
& & \qquad \wedge \cdots \wedge [Q_n] \xrightarrow{\tau} [Q] \\
& \overset{\{\text{Lem. 2.2.4}\}}{\text{impl.}} & \exists Q_1, \ldots, Q_n \in \mathcal{P}^\pi : [P\sigma] \xrightarrow{\sigma(\tau)} [Q_1\sigma] \wedge [Q_1\sigma] \xrightarrow{\sigma(\tau)} [Q_2\sigma] \\
& & \qquad \wedge \cdots \wedge [Q_n\sigma] \xrightarrow{\sigma(\tau)} [Q\sigma] \\
& \text{impl.} & [P\sigma] \left(\xrightarrow{\tau}\right)^* [Q\sigma] \\
& \overset{\{\text{Def. 3.1.2}\}}{\text{impl.}} & [P\sigma] \overset{\langle\rangle}{\Rightarrow} [Q\sigma]
\end{array}$$

holds. ■

Since Lemma 2.2.4 is at the heart of this proof, we see that the converse of this lemma does not hold. Consider, for example, $P =_{\text{def}} a(x) \mid \bar{b}\langle y\rangle$ and $\sigma =_{\text{def}} \{^{a}/_{b}\}$. Then $[P\sigma] \Rightarrow [Q\sigma]$, with $Q =_{\text{def}} \mathbf{0} \mid \mathbf{0}$ holds, but in $[P]$ no τ transition is possible. However, there is also a restricted version of the converse of this lemma presented in Lemma 3.3.3.

We lift the idea of Lemma 3.3.1 to traces with visible actions such that the big-step between two processes leads to a big-step with an application of a substitution to all of its components. Thereby, the lifting is limited to substitutions different to the bound names of a trace. Later, we will see with Convention 3.3.1 that this restriction is no longer harmful.

Lemma 3.3.2 (Substitution on big-step semantics (Part II)) *Given processes $P, Q \in \mathcal{P}^{\pi}$, a trace $s \in \mathtt{Traces}$ and a substitution σ with $\mathtt{n}(\sigma) \cap \mathtt{bn}(s) = \emptyset$,*

$$\textit{if } [P] \overset{s}{\Rightarrow} [Q] \textit{ then } [P\sigma] \overset{s\sigma}{\Rightarrow} [Q\sigma]$$

holds. ♢

Proof: Let $P, Q \in \mathcal{P}^{\pi}$, $s \in \mathtt{Traces}$ and σ a substitution with $\mathtt{n}(\sigma) \cap \mathtt{bn}(s) = \emptyset$. We proceed by induction over the length of s.

Base case $s = \langle\rangle$: Lemma 3.3.1 directly yields the proof of this case.

Base case $s = \langle\alpha\rangle$: Let $\alpha \in \mathtt{Act} \setminus \{\tau\}$ and $[P] \overset{s}{\Rightarrow} [Q]$ hold. So we know from the definition of the big-step semantics (Definition 3.1.2) that there have to be processes Q_i and Q_j such that $[P] \Rightarrow [Q_i] \xrightarrow{\alpha} [Q_j] \Rightarrow [Q]$. With Lemma 3.3.1 we know that $[P\sigma] \Rightarrow [Q_i\sigma]$ and $[Q_j\sigma] \Rightarrow [Q\sigma]$ hold. Furthermore, Lemma 2.2.4 yields that with $[Q_i] \xrightarrow{\alpha} [Q_j]$, also $[Q_i\sigma] \xrightarrow{\sigma(\alpha)} [Q_j\sigma]$ holds since $\mathtt{n}(\sigma) \cap \mathtt{bn}(s) = \emptyset$. So we know $[P\sigma] \xrightarrow{\langle\sigma(\alpha)\rangle} [Q\sigma]$, hence $[P\sigma] \overset{s\sigma}{\Rightarrow} [Q\sigma]$ holds.

Induction hypothesis: For a number $n \in \mathbb{N}$ and all traces $s \in \mathtt{Traces}$, with $n \leq \#(s)$ holds: For all $P, Q \in \mathcal{P}^{\pi}$ and σ substitution with $\mathtt{n}(\sigma) \cap \mathtt{bn}(s) = \emptyset$ if $[P] \overset{s}{\Rightarrow} [Q]$ then $[P\sigma] \overset{s\sigma}{\Rightarrow} [Q\sigma]$.

Inductive step $n \mapsto n+1$: Let $s' \in \mathtt{Traces}$ with $\#(s') = n+1$ and $[P] \overset{s'}{\Rightarrow} [Q]$. So we know that a trace $s \in \mathtt{Traces}$ and an action $\alpha \in \mathtt{Act} \setminus \{\tau\}$ exist such that

$s' = s \frown \langle\alpha\rangle$. Then

$$
\begin{array}{lll}
[P] \xRightarrow{s \frown \langle\alpha\rangle} [Q] & \overset{\{\text{Definition 3.1.2}\}}{\text{impl.}} & [P] \overset{s}{\Rightarrow} \mathbin{;} \xRightarrow{\langle\alpha\rangle} [Q] \\
& \text{impl.} & \exists Q' \in \mathcal{P}^\pi : [P] \overset{s}{\Rightarrow} [Q'] \wedge [Q'] \xRightarrow{\langle\alpha\rangle} [Q] \\
& \overset{\{\text{IH}\}}{\text{impl.}} & \exists Q' \in \mathcal{P}^\pi : [P\sigma] \overset{s\sigma}{\Rightarrow} [Q'\sigma] \wedge [Q'\sigma] \xRightarrow{\langle\alpha\rangle\sigma} [Q\sigma] \\
& \text{impl.} & [P\sigma] \overset{s\sigma}{\Rightarrow} \mathbin{;} \xRightarrow{\langle\alpha\rangle\sigma} [Q\sigma] \\
& \text{impl.} & [P\sigma] \xRightarrow{s\sigma \frown \langle\alpha\rangle\sigma} [Q\sigma] \\
& \text{impl.} & [P\sigma] \xRightarrow{(s \frown \langle\alpha\rangle)\sigma} [Q\sigma] \\
& \text{impl.} & [P\sigma] \xRightarrow{s'\sigma} [Q\sigma]
\end{array}
$$

holds.

Thus, Lemma 3.3.2 holds. ■

With another small example, we can see that the converse of this lemma does not hold either. Consider, for example, $P =_{\text{def}} a(x).\overline{x}\langle c\rangle \mid \overline{b}\langle y\rangle.\overline{c}\langle d\rangle$ and $\sigma =_{\text{def}} \{{}^{a}/_{b}\}$. Then $[P\sigma] = [a(x).\overline{x}\langle c\rangle \mid \overline{a}\langle y\rangle.\overline{c}\langle d\rangle] \xrightarrow{\tau} [\overline{y}\langle c\rangle \mid \overline{c}\langle d\rangle] \xrightarrow{\overline{y}\langle c\rangle} [\mathbf{0} \mid \overline{c}\langle d\rangle]$, so $[P\sigma] \xRightarrow{\langle\overline{y}\langle c\rangle\rangle} [\mathbf{0} \mid \overline{c}\langle d\rangle]$ holds. But there is no trace $s \in \texttt{Traces}$ with $s\sigma = \langle\overline{y}\langle c\rangle\rangle$ and no process $Q \in \mathcal{P}^\pi$ with $[Q\sigma] = [\mathbf{0} \mid \overline{c}\langle d\rangle]$ such that $[P] \overset{s}{\Rightarrow} [Q]$, because every trace starting in $[P]$ has right at the front an input action or $\overline{b}\langle y\rangle$. And since $y \notin \texttt{cosupp}(\sigma)$, we know that for no trace s starting in $[P]$ can $s\sigma = \langle\overline{y}\langle c\rangle\rangle$ hold.

Furthermore, consider, for example, $P =_{\text{def}} y(a)$ and $\sigma =_{\text{def}} \{{}^{b}/_{x}\}$, then $[P\sigma] = [P] \xRightarrow{\langle y\, x\rangle} [\mathbf{0}]$. Thus, the converse would yield that there must be a trace $s \in \texttt{Traces}$ and a process Q with $[P] \overset{s}{\Rightarrow} [Q]$ and $s\,\{{}^{b}/_{x}\} = \langle y\, x\rangle$. But we cannot find a trace s such that we can apply a substitution to s which replaces the name x and results in a trace where the name x occurs free.

But as in the operational semantics case there are restricted converses such that we gain a connection between big-steps with and without the application of a substitution.

Lemma 3.3.3 (Substitution on big-step semantics (Part III)) *Given $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ and*

(I) $\sigma = \{{}^{a}/_{b}\}$ a substitution with $a \notin \texttt{fn}(P)$ or

(II) $\sigma = \{a \leftrightarrow b\}$ a transposition,

then

$$[P\sigma] \Rightarrow [Q'] \text{ implies } \exists Q \in \mathcal{P}^\pi : [P] \Rightarrow [Q]$$

with $[Q\,\{a \leftrightarrow b\}] = [Q']$ *holds.* ♢

Proof: Let $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ and $\sigma = \{^a/_b\}$ substitution with $a \notin \mathtt{fn}(P)$ or $\sigma = \{a \leftrightarrow b\}$ transposition. Furthermore, there is a $Q' \in \mathcal{P}^\pi$ such that $[P\sigma] \Rightarrow [Q']$. We proceed by induction over the number $n \in \mathbb{N}$ of τ steps within $\Rightarrow$.

Base case $n = 0$**:** Thus, $[P\sigma] = [Q']$. Chose $Q =_{\text{def}} P$, so we know $[P] \Rightarrow [Q]$ and $[Q\,\{a \leftrightarrow b\}] = [P\,\{a \leftrightarrow b\}] = [P\sigma] = [Q']$ holds.

Base case $n = 1$**:** Thus, $[P\sigma] \xrightarrow{\tau} [Q']$. Corollary 2.2.1 respectively Corollary 2.2.2 yields that there is an action $\alpha \in \mathtt{Act}$ and a process $Q \in \mathcal{P}^\pi$ such that $[P] \xrightarrow{\alpha} [Q]$ and $\{a \leftrightarrow b\}(\alpha) = \tau$ and $[Q\,\{a \leftrightarrow b\}] = [Q']$. Hence, $\alpha = \tau$ and so $[P] \Rightarrow [Q]$.

Induction hypothesis: For all $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ and $\sigma = \{^a/_b\}$ a substitution with $a \notin \mathtt{fn}(P)$ or $\sigma = \{a \leftrightarrow b\}$ transposition and $[P\sigma] \Rightarrow [Q']$ with some number of τ transitions $n \in \mathbb{N}$, exists a process $Q \in \mathcal{P}^\pi$ with $[P] \Rightarrow [Q]$ and $[Q\,\{a \leftrightarrow b\}] = [Q']$.

Induction step: Let $[P\sigma] \Rightarrow [Q']$ with a number of τ steps of $n+1$. Thus, we know there is a process $Q'_1 \in \mathcal{P}^\pi$ such that $[P\sigma] \Rightarrow [Q'_1]$ in a number of τ steps of n and $[Q'_1] \xrightarrow{\tau} [Q']$. From the induction hypothesis we know that there is a process $Q_1 \in \mathcal{P}^\pi$ such that $[P] \Rightarrow [Q_1]$ with $[Q_1\,\{a \leftrightarrow b\}] = [Q'_1]$. So we know $[Q_1\,\{a \leftrightarrow b\}] \xrightarrow{\tau} [Q']$ and with Corollary 2.2.1 respectively Corollary 2.2.2 we know there is an action $\alpha \in \mathtt{Act}$ and a process $Q \in \mathcal{P}^\pi$ such that $[Q_1] \xrightarrow{\alpha} [Q]$ and $\{a \leftrightarrow b\}(\alpha) = \tau$ and $[Q\,\{a \leftrightarrow b\}] = [Q']$. Thus, $\alpha = \tau$ and due to that $[P] \Rightarrow [Q]$.

Thus, Lemma 3.3.3 holds. ■

With the help of Corollary 2.2.1 we extend the previous lemma for big-steps with visible actions.

Lemma 3.3.4 (Substitution on big-step semantics (Part IV)) *Given names* $a, b \in \mathcal{N}$*, a process* $P \in \mathcal{P}^\pi$*, a trace* $s' \in \mathtt{Traces}$*, and*

(I) $\sigma = \{^a/_b\}$ *a substitution with* $a \notin \mathtt{fn}(P)$ *or*

(II) $\sigma = \{a \leftrightarrow b\}$ *a transposition*

with $\mathtt{n}(\sigma) \cap \mathtt{bn}(s') = \emptyset$*, then*

$$[P\sigma] \stackrel{s'}{\Rightarrow} [Q'] \text{ implies } \exists Q \in \mathcal{P}^\pi, s \in \mathtt{Traces} : [P] \stackrel{s}{\Rightarrow} [Q]$$

with $s\{a \leftrightarrow b\} = s'$ *and* $[Q\{a \leftrightarrow b\}] = [Q']$. ♢

Proof: Let $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$, $s' \in \mathtt{Traces}$, and $\sigma = \{^a/_b\}$ a substitution with $a \notin \mathtt{fn}(P)$ or $\sigma = \{a \leftrightarrow b\}$ transposition with $\mathtt{n}(\sigma) \cap \mathtt{bn}(s') = \emptyset$ and $[P\sigma] \stackrel{s'}{\Rightarrow} [Q']$. We proceed by induction over the length of s'.

Base case $\#(s') = 0$**:** Thus, $s' = \langle\rangle$ holds. Hence, in this case Lemma 3.3.4 directly follows from Lemma 3.3.3.

Base case $\#(s') = 1$**:** So $s' = \langle\beta\rangle$ for an action $\beta \in \mathtt{Act} \setminus \{\tau\}$ and there are processes $Q'_1, Q'_2 \in \mathcal{P}^\pi$ such that $[P\sigma] \Rightarrow [Q'_1]$, $[Q'_1] \xrightarrow{\beta} [Q'_2]$, and $[Q'_2] \Rightarrow [Q']$. Lemma 3.3.3 yields that there is a process $Q_1 \in \mathcal{P}^\pi$ with $[P] \Rightarrow [Q_1]$ and $[Q_1\{a \leftrightarrow b\}] = [Q'_1]$. With $[Q_1\{a \leftrightarrow b\}] = [Q'_1] \xrightarrow{\beta} [Q'_2]$ and Corollary 2.2.1 respectively Corollary 2.2.2, we know that there is an action $\alpha \in \mathtt{Act}$ and a process $Q_2 \in \mathcal{P}^\pi$ such that $[Q_1] \xrightarrow{\alpha} [Q_2]$ with $\{a \leftrightarrow b\}(\alpha) = \beta$ and $[Q_2\{a \leftrightarrow b\}] = [Q'_2]$. Lemma 3.3.3 with $[Q_2\{a \leftrightarrow b\}] = [Q'_2] \Rightarrow [Q']$ yields that there is a process $Q \in \mathcal{P}^\pi$ with $[Q_2] \Rightarrow [Q]$ and $[Q\{a \leftrightarrow b\}] = [Q']$. Hence, $[P] \stackrel{\langle\alpha\rangle}{\Longrightarrow} [Q]$. Let $s =_{\text{def}} \langle\alpha\rangle$, then also $s\{a \leftrightarrow b\} = s'$ holds.

Induction hypothesis: For a number $n \in \mathbb{N}$ and all traces $t' \in \mathtt{Traces}$, with $n \leq \#(t')$ and $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$, $\sigma = \{^a/_b\}$ a substitution with $a \notin \mathtt{fn}(P)$ or $\sigma = \{a \leftrightarrow b\}$ transposition with $\mathtt{n}(\sigma) \cap \mathtt{bn}(t') = \emptyset$ and $[P\sigma] \stackrel{t'}{\Rightarrow} [Q']$, exists a process $Q \in \mathcal{P}^\pi$ and a trace $t \in \mathtt{Traces}$ with $[P] \stackrel{t}{\Rightarrow} [Q]$ and $t\{a \leftrightarrow b\} = t'$ and $[Q\{a \leftrightarrow b\}] = [Q']$.

Induction step: So $\#(s') = n + 1$ and so there is a trace $t' \in \mathtt{Traces}$, a process $Q'_1 \in \mathcal{P}^\pi$ and an action $\beta \in \mathtt{Act} \setminus \{\tau\}$ with $[P\sigma] \stackrel{t'}{\Rightarrow} [Q'_1]$ and $[Q'_1] \stackrel{\langle\beta\rangle}{\Longrightarrow} [Q']$ and $s' = t' \frown \langle\beta\rangle$. The induction hypothesis yields that there is a process $Q_1 \in \mathcal{P}^\pi$ and a trace $t \in \mathtt{Traces}$ with $[P] \stackrel{t}{\Rightarrow} [Q_1]$ and $t\{a \leftrightarrow b\} = t'$ and $[Q_1\{a \leftrightarrow b\}] = [Q'_1]$. Since $[Q_1\{a \leftrightarrow b\}] = [Q'_1] \stackrel{\langle\beta\rangle}{\Longrightarrow} [Q']$ we know from the second base case that there is a process $Q \in \mathcal{P}^\pi$ and an action $\alpha \in \mathtt{Act}$ with $[Q_1] \stackrel{\langle\alpha\rangle}{\Longrightarrow} [Q]$ and $\langle\alpha\rangle\{a \leftrightarrow b\} = \langle\beta\rangle$ and $[Q\{a \leftrightarrow b\}] = [Q']$. Thus, define $s =_{\text{def}} t \frown \langle\alpha\rangle$ then we know $s\{a \leftrightarrow b\} = s'$ and $[P] \stackrel{s}{\Rightarrow} [Q]$.

Thus, Lemma 3.3.4 is proved by induction over the length of the trace. ■

Thereby, we see that the example given after Lemma 3.3.2 for the non-existence of the converse is fixed by the previous lemma by the usage of a transposition. Again, consider $P =_{\text{def}} y(a)$ and $\sigma =_{\text{def}} \{{}^{b}/_{x}\}$, then $[P\sigma] = [P] \xrightarrow{\langle y\, x\rangle} [\mathbf{0}]$. This time, with $s =_{\text{def}} \langle y\, b\rangle$ and $Q =_{\text{def}} \mathbf{0}$, there is a trace $s \in \texttt{Traces}$ and a process Q with $[P] \stackrel{s}{\Rightarrow} [Q]$ and $s\,\{b \leftrightarrow x\} = \langle y\, x\rangle$ and $[Q\,\{b \leftrightarrow x\}] = [\mathbf{0}]$.

Since the application of a substitution is only defined on traces with bound names different to the names of the substitution, it is useful to take a closer look at the bound names of a trace. Obviously, we would like to apply a given substitution to an arbitrary trace without destroying anything of its semantics. Intuitively, since those names are bound, we should be able to change the bound names within a trace starting in a process P and the resulting trace is still a valid behavior of P.

Furthermore, we would like to abstract from traces where the bound names within are not unique for reasons of simplicity.

Definition 3.3.2 (Unique traces) We call a trace $t \in \texttt{Traces}$ *unique* if

(1) $t_i = \overline{a}(b)$ implies $\nexists c \in \mathcal{N}, j \in \mathbb{N} : i \neq j \wedge t_j = \overline{c}(b)$,

(2) $t_i = \overline{a}(b)$ implies $b \notin \texttt{n}(\langle t_1, \ldots, t_{i-1}\rangle)$

holds. *

This definition is helpful for determine the scope of a name within a trace. Consider, for example, $P =_{\text{def}} \underline{\texttt{new}}\, c\; b(x).\overline{b}\langle c\rangle.b(y)$. We know $[P] \xrightarrow{b\,a} \left[\underline{\texttt{new}}\, c\; \overline{b}\langle c\rangle.b(y)\right] \xrightarrow{\overline{b}(a)} [b(y)] \xrightarrow{b\,a} [\mathbf{0}]$, since $\underline{\texttt{new}}\, a\; \overline{b}\langle a\rangle.b(y) \in \left[\underline{\texttt{new}}\, c\; \overline{b}\langle c\rangle.b(y)\right]$. Thus, $[P] \xrightarrow{\langle b\,a, \overline{b}(a), b\,a\rangle} [\mathbf{0}]$ with different names a. In particular, for $\langle b\,a, \overline{b}(a), b\,a\rangle$ we know that on the hand the first a is a name visibly to the environment and on the other hand the second one is a different name a and till this moment invisibly for the environment.

It is for this reason that we define the *bound substitution* function, which suitably replaces the bound names within a trace.

Definition 3.3.3 (Bound substitution) As $\texttt{bn}_{\texttt{subst}} : \mathbb{P}(\mathcal{N}^2) \times \texttt{Traces} \to \texttt{Traces}$ we define the *bound substitution* such that
if $\exists i \in \mathbb{N}, x \in \mathcal{N} \nexists j \in \mathbb{N}, y \in \mathcal{N} : i < j \wedge t_i = \overline{x}(a) \wedge t_j = \overline{y}(a) \wedge \#(t) = n$ then

$$\texttt{bn}_{\texttt{subst}}(\{{}^{b}/_{a}\}, t) = \langle t_1, \ldots, t_{i-1}\rangle^\frown \langle \overline{x}(b)\rangle^\frown (\langle t_{i+1}, \ldots, t_n\rangle\, \{{}^{b}/_{a}\}),$$

if $\exists i, j \in \mathbb{N}, x, y \in \mathcal{N} : i < j \wedge t_i = \overline{x}(a) \wedge t_j = \overline{y}(a) \wedge \#(t) = n$ then

$$\texttt{bn}_{\texttt{subst}}(\{{}^{b}/_{a}\}, t) = \langle t_1, \ldots, t_{i-1}\rangle^\frown \langle \overline{x}(b)\rangle^\frown (\langle t_{i+1}, \ldots, t_{j-1}\rangle\, \{{}^{b}/_{a}\})^\frown \langle t_j, \ldots, t_n\rangle,$$

otherwise

$$\mathtt{bn}_{\mathtt{subst}}(\{{}^{b}/_{a}\}, t) = t$$

holds. $*$

We now show with the previous definition that if there is a big-step with a trace t starting in a process, then there is still a big-step starting in the process for a trace t' if we suitably changed the bound names within the trace t to obtain t'.

Lemma 3.3.5 (Big-step semantics under bound name changes) *For a name $a \in \mathcal{N}$, a trace $t \in \mathtt{Traces}$ and processes $P, Q \in \mathcal{P}^{\pi}$*

$$[P] \stackrel{t}{\Rightarrow} [Q] \text{ implies } \exists Q' \in \mathcal{P}^{\pi} : [P] \stackrel{\mathtt{bn}_{\mathtt{subst}}(\{{}^{b}/_{a}\},t)}{\Longrightarrow} [Q']$$

holds for all $b \notin \mathtt{fn}(P) \cup \mathtt{n}(t)$. $\diamondsuit$

Proof: Let $b \notin \mathtt{fn}(P) \cup \mathtt{n}(t)$, $a \in \mathcal{N}$, $P, Q \in \mathcal{P}^{\pi}$, and $t \in \mathtt{Traces}$ with $\#(t) = n$ for some $n \in \mathbb{N}$, and $[P] \stackrel{t}{\Rightarrow} [Q]$. If $a \notin \mathtt{bn}(t)$ then $t = \mathtt{bn}_{\mathtt{subst}}(\{{}^{b}/_{a}\}, t)$. Hence $[P] \stackrel{\mathtt{bn}_{\mathtt{subst}}(\{{}^{b}/_{a}\},t)}{\Longrightarrow} [Q]$. If $a \in \mathtt{bn}(t)$ there must be a number $i \in \mathbb{N}$ and a name $x \in \mathcal{N}$ such that $t = t' \frown \langle \overline{x}(a) \rangle \frown t''$ with $t' =_{\text{def}} \langle t_1, \ldots, t_{i-1} \rangle$ and $t'' =_{\text{def}} \langle t_{i+1}, \ldots, t_n \rangle$. Without loss of generality, we assume that this index is the first occurrence of a bound output with object a within the trace. So there are processes $Q_1, Q_2 \in \mathcal{P}^{\pi}$ with $[P] \stackrel{t'}{\Rightarrow} [Q_1] \stackrel{\overline{x}(a)}{\longrightarrow} [Q_2] \stackrel{t''}{\Rightarrow} [Q]$. Lemma 3.2.1 yields that $\mathtt{fn}(Q_1) \subseteq \mathtt{fn}(P) \cup \mathtt{n}(t')$ and so $b \notin \mathtt{fn}(Q_1)$. Hence, with Lemma 2.2.2 and since $\mathtt{fn}(\underline{\mathtt{new}}\, a\ Q_1) \subseteq \mathtt{fn}(Q_1)$ we know $[Q_1] \stackrel{\overline{x}(b)}{\longrightarrow} [Q_2\{{}^{b}/_{a}\}]$. If there is no number $j \in \mathbb{N}$ and name $y \in \mathcal{N}$ with $t_j = \overline{y}(a)$ and $j > i$ we know from Lemma 3.3.2 that $[Q_2\{{}^{b}/_{a}\}] \stackrel{t''\{{}^{b}/_{a}\}}{\Longrightarrow} [Q\{{}^{b}/_{a}\}]$. Furthermore, since in this case $\mathtt{bn}_{\mathtt{subst}}(\{{}^{b}/_{a}\}, t) = t' \frown \langle \overline{x}(b) \rangle \frown t''\{{}^{b}/_{a}\}$ we know $[P] \stackrel{\mathtt{bn}_{\mathtt{subst}}(\{{}^{b}/_{a}\},t)}{\Longrightarrow} [Q\{{}^{b}/_{a}\}]$. Otherwise, there is a number $j \in \mathbb{N}$ and a name $y \in \mathcal{N}$ such that $t_j = \overline{y}(a)$ and $j > i$. With $s =_{\text{def}} \langle t_{i+1}, \ldots, t_{j-1} \rangle$ and $s' =_{\text{def}} \langle t_j, \ldots, t_n \rangle$ we know there is a process $Q_2' \in \mathcal{P}^{\pi}$ such that $[P] \stackrel{t'}{\Rightarrow} [Q_1] \stackrel{\overline{x}(a)}{\longrightarrow} [Q_2] \stackrel{s}{\Rightarrow} [Q_2'] \stackrel{s'}{\Rightarrow} [Q]$. Without loss of generality, we assume that this index is the second occurrence of a bound output with object a within the trace. Thus, with Lemma 3.3.2 we know $[Q_2\{{}^{b}/_{a}\}] \stackrel{s\{{}^{b}/_{a}\}}{\Longrightarrow} [Q_2'\{{}^{b}/_{a}\}]$ holds. If $a \in \mathtt{fn}(Q_2')$ we know from Convention 2.2.1 that $a \notin \mathtt{bn}(Q_2')$. Furthermore, there is no process $R \in \mathcal{P}^{\pi}$ such that $R =_{\alpha} Q_2'$ and $a \in \mathtt{bn}(R)$, since with $a \in \mathtt{fn}(Q_2')$ we know $a \in \mathtt{fn}(R)$ for all $R =_{\alpha} Q_2'$ and so the name a would violate Convention 2.2.1 because $a \in \mathtt{fn}(R) \cup \mathtt{bn}(R)$. But if there is no process $R \in [Q_2']$ with $a \in \mathtt{bn}(R)$, there is no possibility for a transition $\stackrel{\overline{y}(a)}{\longrightarrow}$ to start in $[Q_2']$. Thus, $a \notin \mathtt{fn}(Q_2')$ and so $[Q_2'\{{}^{b}/_{a}\}] = [Q_2']$. Hence, $[P] \stackrel{t'}{\Rightarrow} [Q_1] \stackrel{\overline{x}(b)}{\longrightarrow} [Q_2\{{}^{b}/_{a}\}] \stackrel{s\{{}^{b}/_{a}\}}{\Longrightarrow} [Q_2'\{{}^{b}/_{a}\}] \stackrel{s'}{\Rightarrow} [Q]$ and thus, $[P] \stackrel{\mathtt{bn}_{\mathtt{subst}}(\{{}^{b}/_{a}\},t)}{\Longrightarrow} [Q]$. ■

Thus, we can change the bound names within a trace with any fresh name without changing its semantics. This justifies the following convention.

Convention 3.3.1 (Uniqueness of bound names in traces) *For a trace* $t \in \texttt{Traces}$ *and substitutions* $\sigma_1, \ldots, \sigma_n$ *for some* $n \in \mathbb{N}$ *under consideration, we stipulate that* t *is unique and* $\texttt{n}(\sigma_i) \cap \texttt{bn}(t) = \emptyset$ *for every* $i \in \{1, \ldots, n\}$. ∘

Note that with this convention the constraint $\texttt{n}(\sigma) \cap \texttt{bn}(t) = \emptyset$ in Lemma 3.3.2 and Lemma 3.3.4 is no longer harmful.

4 Trace semantics

In the *trace semantics* every single trace – meaning all possible sequences of external behavior – of a process is collected. Thereby, the internal steps are omitted. Thus, the trace semantics is a *denotational semantics*, which represents the communication a process offers to its environment. In Chapter 3 with the big-step semantics we regarded one possible external behavior of a process. Now with the trace semantics we introduce a denotational semantics, which is a collection of all these possible big-steps of a process. Intuitively, the big-step semantics follows *one* way through the transition system produced from the rules of Figure 2.3 and collects the visible actions. Whereas the trace semantics collects *all* of those ways through the transition system. This idea is an adaption of the trace semantics for CSP, for example, described in [Ros98].

4.1 Definition

With the big-step semantics the ability to take an external view of a process' behavior is given. Between every single action in a given sequence there can be as many internal actions as needed. The trace semantics just collects all those traces a process can possibly perform in one set.

Definition 4.1.1 (Trace semantics) For every process $P \in \mathcal{P}^\pi$ the *trace semantics* $\mathcal{T} : \mathcal{P}^\pi_\alpha \to \mathbb{P}(\texttt{Traces})$ with

$$\mathcal{T}([P]) =_{\text{def}} \left\{ t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^\pi : [P] \stackrel{t}{\Rightarrow} [Q] \right\}$$

is a set of all possible external behavior of the process P. $*$

In this way, processes can be distinguished by their external behavior. We now introduce some properties of this denotational semantics before defining the refinement, which bases upon this semantics.

4.2 Properties

In this section we investigate the trace semantics and some of its properties for gaining results for the refinement defined in Section 4.3. Furthermore, these properties help in understanding the trace semantics and its applicability. Especially, we examine its behavior while applying a substitution and its relationship to simulation and bisimulation to classify the new semantics in the established context. The main part takes the investigation, whether the trace semantics is compositional or not, resulting by the fact that already the subclass of recursion-free processes is indeed not compositional for every operator.

As a first result we achieve that every set of traces is at most countably infinite. This is because we know that every trace is produced out of a countably infinite set of symbols. In other words, traces are words over a countably infinite alphabet. More precisely, for every $P \in \mathcal{P}^\pi$ we know $\mathcal{T}([P]) \subseteq \left(\mathcal{N} \cup \overline{\mathcal{N}} \cup \texttt{SYMB}\right)^*$ where $\texttt{SYMB}$ denotes the set containing the angle brackets, the parenthesis, and the comma. Furthermore, we know that the countably infinite sets are closed under union and Kleene star. So, since $\mathcal{N}$ and $\overline{\mathcal{N}}$ are countably infinite, we know that the set $\mathcal{T}([P])$ is countably infinite. Hence, we can define a function $\mathcal{T}_\mathcal{I} : \mathcal{P}^\pi_\alpha \to \mathbb{P}(\mathbb{N} \times \texttt{Traces})$, named *indexed trace set*, such that every trace in the trace set of a equivalence class of a process is uniquely mapped to a natural number.

Furthermore, we know that every trace set $\mathcal{T}([P])$ for a process $P \in \mathcal{P}^\pi$ is closed under the prefix operator. Therefore, consider $s \in \mathcal{T}([P])$. We know for every prefix $t \in \texttt{pref}(\{s\})$ that there is a trace $u \in \texttt{Traces}$ such that $s = t \frown u$. So there is a process $Q \in \mathcal{P}^\pi$ such that $[P] \stackrel{t \frown u}{\Longrightarrow} [Q]$ and so $[P] \stackrel{t}{\Rightarrow} ; \stackrel{u}{\Rightarrow} [Q]$ holds. Hence, there is a process $Q' \in \mathcal{P}^\pi$ such that $[P] \stackrel{t}{\Rightarrow} [Q']$ and so $t \in \mathcal{T}([P])$.

Since the empty sequence is a prefix of every trace, it fits that the empty trace is included in every trace set.

Lemma 4.2.1 (Empty trace) *For all $P \in \mathcal{P}^\pi$*

$$\langle\rangle \in \mathcal{T}([P])$$

holds. ♢

Proof: Let $P \in \mathcal{P}^\pi$. Since $\left(\stackrel{\tau}{\to}\right)^*$ is reflexive, we know that $[P] \left(\stackrel{\tau}{\to}\right)^* [P]$ holds. With the definition of the big-step semantics (Definition 3.1.2) we get $[P] \stackrel{\langle\rangle}{\Rightarrow} [P]$. With that and from the definition of the trace semantics (Definition 4.1.1) we know

that $\langle\rangle \in \mathcal{T}([P])$ holds, since $P \in \mathcal{P}^\pi$. ■

Now, we examine the behavior of the trace semantics in the context of substitutions.

4.2.1 Substitution

Due to Lemma 3.3.2, traces between two processes are preserved under substitution. We can lift this result to the trace semantics.

Lemma 4.2.2 (Substitution on trace semantics (Part I)) *For all $P \in \mathcal{P}^\pi$ and substitutions σ*

$$\mathcal{T}([P])\sigma \subseteq \mathcal{T}([P\sigma])$$

holds. ♢

Proof: Let $P \in \mathcal{P}^\pi$, σ a substitution, and $s \in (\mathcal{T}([P])\sigma)$. Then we know from the definition of the substitution on traces and the trace semantics that there exists a process $Q \in \mathcal{P}^\pi$ and a trace $s' \in \texttt{Traces}$ such that $[P] \stackrel{s'}{\Rightarrow} [Q]$ and $s = s'\sigma$ holds. Lemma 3.3.2 yields that $[P\sigma] \stackrel{s'\sigma}{\Longrightarrow} [Q\sigma]$ holds. So we know $[P\sigma] \stackrel{s}{\Rightarrow} [Q\sigma]$ and since $Q\sigma \in \mathcal{P}^\pi$, we know from the definition of the trace semantics that $s \in \mathcal{T}([P\sigma])$. ■

From the example to Lemma 3.3.2 we already know that the converse set inclusion does not hold. Because there, we found a trace $\langle\overline{y}\langle c\rangle\rangle \in \mathcal{T}([P\,\{^a/_b\}])$ for $P =_{\text{def}} a(x).\overline{x}\langle c\rangle \mid \overline{b}\langle y\rangle.\overline{c}\langle d\rangle$ and argued that no trace s starting in $[P]$ can satisfy $s\sigma = \langle\overline{y}\langle c\rangle\rangle$. So $\langle\overline{y}\langle c\rangle\rangle \notin \mathcal{T}([P])\sigma$.

But once more, we can prove a restricted version of the converse of the previous Lemma and this directly leads to a restricted version of a set equality.

Lemma 4.2.3 (Substitution on trace semantics (Part II)) *Given $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ then*

$$\mathcal{T}([P\,\{a \leftrightarrow b\}]) = \mathcal{T}([P])\,\{a \leftrightarrow b\}$$

holds. ♢

Proof: Let $a, b \in \mathcal{N}$, $P \in \mathcal{P}^\pi$ and $\sigma = \{a \leftrightarrow b\}$ transposition. Then, Lemma 4.2.2 directly yields $\mathcal{T}([P\,\{a \leftrightarrow b\}]) \supseteq \mathcal{T}([P])\,\{a \leftrightarrow b\}$ since a transposition is a special case of a substitution. For the other inclusion we take a trace $s' \in \mathcal{T}([P\sigma])$. Thus, there is a process $Q' \in \mathcal{P}^\pi$ such that $[P\sigma] \stackrel{s'}{\Rightarrow} [Q']$. Lemma 3.3.4 yields that there is a process $Q \in \mathcal{P}^\pi$ and a trace $s \in \texttt{Traces}$ with $[P] \stackrel{s}{\Rightarrow} [Q]$ and $s\,\{a \leftrightarrow b\} = s'$. Hence, $s \in \mathcal{T}([P])$ and so $s' \in \mathcal{T}([P])\,\{a \leftrightarrow b\}$. ■

Thus, for a transposition the trace sets are equal no matter whether the transposition is applied to the process or to the trace set. Furthermore, for a substitution which only replaces one name which is a fresh one for the process, the trace set of the process with applied substitution is the same as the trace set of the process where a transposition of the names of the substitution is applied to every trace. This is because of $[P\sigma] = [P\,\{a \leftrightarrow b\}]$ for a substitution $\sigma =_{\text{def}} \{^{a}/_{b}\}$ and $a \notin \mathtt{fn}(P)$.

Furthermore, we can see that the set inclusion is preserved under the application of a substitution.

Lemma 4.2.4 (Substitution and trace inclusion (Part I)) *For all processes $P, Q \in \mathcal{P}^{\pi}$ and substitutions σ,*

$$\textit{if } \mathcal{T}([P]) \subseteq \mathcal{T}([Q]) \textit{ then } \mathcal{T}([P])\sigma \subseteq \mathcal{T}([Q])\sigma$$

holds. ♢

Proof: Let $P, Q \in \mathcal{P}^{\pi}$, σ a substitution, and $\mathcal{T}([P]) \subseteq \mathcal{T}([Q])$. Given $s \in \mathcal{T}([P])\sigma$, we know there has to be a trace $s' \in \mathcal{T}([P])$, with $s = s'\sigma$. So $s' \in \mathcal{T}([Q])$ holds and also $s = s'\sigma \in \mathcal{T}([Q])\sigma$ holds. ■

The converse of the previous lemma cannot hold. Consider, for example, $P =_{\text{def}} \overline{a}\langle x\rangle$, $Q =_{\text{def}} \overline{b}\langle x\rangle$ and $\sigma =_{\text{def}} \{^{a}/_{b}\}$. Then $\mathcal{T}([P]) = \{\langle\rangle, \langle\overline{a}\langle x\rangle\rangle\}$, and $\mathcal{T}([Q]) = \left\{\langle\rangle, \langle\overline{b}\langle x\rangle\rangle\right\}$. Hence, $\mathcal{T}([P])\sigma = \{\langle\rangle, \langle\overline{a}\langle x\rangle\rangle\}$ and $\mathcal{T}([Q])\sigma = \{\langle\rangle, \langle\overline{a}\langle x\rangle\rangle\}$. Thus, $\mathcal{T}([P])\sigma \subseteq \mathcal{T}([Q])\sigma$ but $\mathcal{T}([P]) \not\subseteq \mathcal{T}([Q])$.

In spite of that, Lemma 4.2.3 and Lemma 4.2.4 yield directly a helpful connection between trace set inclusion and a transposition.

Corollary 4.2.1 (Substitution and trace inclusion (Part II)) *Given $P, Q \in \mathcal{P}^{\pi}$ and $a, b \in \mathcal{N}$ then,*

$$\textit{if } \mathcal{T}([P]) \subseteq \mathcal{T}([Q]) \textit{ then } \mathcal{T}([P\,\{a \leftrightarrow b\}]) \subseteq \mathcal{T}([Q\,\{a \leftrightarrow b\}])$$

holds □

Proof: Let $P, Q \in \mathcal{P}^{\pi}$ and $\sigma = \{a \leftrightarrow b\}$ be a transposition with $\mathcal{T}([P]) \subseteq \mathcal{T}([Q])$. Furthermore, let $t \in \mathcal{T}([P\sigma])$. On the one hand Lemma 4.2.3 yields that $t \in \mathcal{T}([P])\,\{a \leftrightarrow b\}$ and on the other hand Lemma 4.2.4 yields that $\mathcal{T}([P])\,\{a \leftrightarrow b\} \subseteq \mathcal{T}([Q])\,\{a \leftrightarrow b\}$. Thus, $t \in \mathcal{T}([Q])\,\{a \leftrightarrow b\}$ and again with Lemma 4.2.3 we know $t \in \mathcal{T}([Q\sigma])$ holds. ■

Thus, for transpositions, the trace inclusion is preserved. Furthermore, since $[P\sigma] = [P\{a \leftrightarrow b\}]$ for a substitution $\sigma =_{\text{def}} \{^{a}/_{b}\}$ and $a \notin \texttt{fn}(P)$, we know Corollary 4.2.1 is also true for a substitution which only replaces one name with a new fresh one.

4.2.2 Compositionality

In this section we investigate the compositionality of the trace semantics. That is, for a binary operator $\circ$ on processes, we have a binary operation $\circ_{\mathcal{T}}$ on the trace sets such that $\mathcal{T}([P \circ Q]) = \mathcal{T}([P]) \circ_{\mathcal{T}} \mathcal{T}([Q])$ holds. By this investigation, we get that a formal compositionality of every operator most likely does not exist. For example, consider an input process $a(x).P$, then the new communications, which can be established by the incoming name, can possibly not be obtained by the traces of P. Therefore, we develop a algorithmic notion for calculating a subset of the traces compositional. That is, we can algorithmically calculate the traces of a process compositional, if we assume that only a finite number of names can be send from the outside to the process.

Furthermore, we restrict the compositionality of the parallel operator to the class of restriction-free and recursion-free processes for reasons of simplicity. This is of minor influence, since we know with Lemma 2.2.1 that every process can be transformed in extended standard form such that all restriction operators are right at the front of the process. Additionally, by a suitable unfolding technique of the process calls, we assume that we can possibly get rid of the restriction to recursion-free process as long as we can find some kind of fix-point algorithm for the unfolding. However, this investigation was out of the scope of this thesis. For practically usage of this semantics, the following results appear to be helpful.

Inaction, τ and output process, choice

We start the investigation of the compositionality of the trace semantics with the unproblematic cases for the inaction, the τ and output process, and the choice.

Lemma 4.2.5 (Compositionality: τ processes/output processes/choices) *Given processes $P, Q \in \mathcal{P}^{\pi}$ and names $a, x \in \mathcal{N}$, then*

$$\mathcal{T}([\mathbf{0}]) = \{\langle\rangle\} \qquad \text{(INACTION)}$$

$$\mathcal{T}([\tau.P]) = \mathcal{T}([P]) \qquad \text{(TAU)}$$

$$\mathcal{T}([\overline{a}\langle x\rangle.P]) = \{\langle\rangle\} \cup \{\langle \overline{a}\langle x\rangle\rangle\}^\frown \mathcal{T}([P]) \qquad \text{(OUTPUT)}$$
$$\mathcal{T}([P+Q]) = \mathcal{T}([P]) \cup \mathcal{T}([Q]) \qquad \text{(CHOICE)}$$

holds. ♢

Proof: To prove Lemma 4.2.5 we handle every case individually.

Case $\mathcal{T}([\mathbf{0}])$: From the definition of the trace semantics (Definition 4.1.1) we know that

$$\mathcal{T}([\mathbf{0}]) = \left\{ t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^\pi : [\mathbf{0}] \stackrel{t}{\Rightarrow} [Q] \right\}$$

holds. Furthermore, with Lemma 4.2.1 we know $\langle\rangle \in \mathcal{T}([\mathbf{0}])$ so that

$$\mathcal{T}([\mathbf{0}]) = \{\langle\rangle\} \cup \underbrace{\left\{ t \in \texttt{Traces} \setminus \{\langle\rangle\} \mid \exists Q \in \mathcal{P}^\pi : [\mathbf{0}] \stackrel{t}{\Rightarrow} [Q] \right\}}_{Y =_{\text{def}}}. \tag{4.1}$$

Let us assume that $Y \neq \emptyset$ and therefore let $t \in Y$. Then, from Equation 4.1, we know that a process $Q \in \mathcal{P}^\pi$ must exist such that $[\mathbf{0}] \stackrel{t}{\Rightarrow} [Q]$. Since $t \neq \langle\rangle$, there has to be an action $\alpha \in t$ with $\alpha \neq \tau$ and a trace $s \in \texttt{Traces}$ due to the definition of the big-step semantics (Definition 3.1.2) such that $[\mathbf{0}] \Rightarrow \stackrel{\alpha}{\rightarrow} \Rightarrow [Q]$ holds and $t = \langle\alpha\rangle^\frown s$. But there is no rule of the operational semantics (Definition 2.2.10), such that for a process $Q' \in \mathcal{P}^\pi$, there is a transition like $[\mathbf{0}] \stackrel{\tau}{\rightarrow} [Q']$ or $[\mathbf{0}] \stackrel{\alpha}{\rightarrow} [Q']$. This is a contradiction, thus, $Y = \emptyset$. Hence with Equation 4.1 we know that all in all

$$\mathcal{T}([\mathbf{0}]) = \{\langle\rangle\}$$

holds.

Case $\mathcal{T}([\tau.P])$: Let $P \in \mathcal{P}^\pi$. As in the prior case we know from Definition 4.1.1 that

$$\mathcal{T}([\tau.P]) = \left\{ t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^\pi : [\tau.P] \stackrel{t}{\Rightarrow} [Q] \right\}$$

From the definition of the operational semantics (Definition 2.2.10) we know that there is only one rule, which handles a τ prefix. Hence with the $E-TAU$

rule we know that

$$\mathcal{T}([\tau.P]) = \left\{t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^\pi : [\tau.P] \xrightarrow{\tau} [P] \wedge [P] \overset{s}{\Rightarrow} [Q] \wedge t = \langle\rangle \frown s\right\}$$

holds, due to the fact that τ actions are omitted in traces. Since $[\tau.P] \xrightarrow{\tau} [P]$ is true for all $P \in \mathcal{P}^\pi$, it can be omitted so that

$$\mathcal{T}([\tau.P]) = \left\{t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^\pi : [P] \overset{s}{\Rightarrow} [Q] \wedge t = \langle\rangle \frown s\right\}$$

holds. This leads with the definition of the concatenation of traces and the definition of the trace semantics to $\mathcal{T}([\tau.P]) = \mathcal{T}([P])$.

Case $\mathcal{T}([\overline{a}\langle x\rangle.P])$: Let $P \in \mathcal{P}^\pi$ be a process and $a, x \in \mathcal{N}$ be names. From Definition 4.1.1 and Lemma 4.2.1 we, analogously to the stop process case, know that

$$\mathcal{T}([\overline{a}\langle x\rangle.P]) = \{\langle\rangle\} \cup \underbrace{\left\{t \in \texttt{Traces} \setminus \{\langle\rangle\} \mid \exists Q \in \mathcal{P}^\pi : [\overline{a}\langle x\rangle.P] \overset{t}{\Rightarrow} [Q]\right\}}_{Y =_{\text{def}}} \quad (4.2)$$

holds and once again there is only one rule – the $E - OUT$ rule – in Definition 2.2.10, which handles an output prefix. So we get

$$Y = \Big\{t \in \texttt{Traces} \setminus \{\langle\rangle\} \mid \exists Q \in \mathcal{P}^\pi : [\overline{a}\langle x\rangle.P] \xrightarrow{\overline{a}\langle x\rangle} [P] \wedge [P] \overset{s}{\Rightarrow} [Q] \wedge t = \langle \overline{a}\langle x\rangle\rangle \frown s\Big\}.$$

Since we know from the operational semantics that $[\overline{a}\langle x\rangle.P] \xrightarrow{\overline{a}\langle x\rangle} [P]$ is true for every process P and furthermore, we know that the empty trace has already been excluded, we have

$$Y = \left\{t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^\pi : [P] \overset{s}{\Rightarrow} [Q] \wedge t = \langle \overline{a}\langle x\rangle\rangle \frown s\right\}.$$

By the definition of the concatenation of trace sets (Definition 2.1.2) we get

$$Y = \{\langle \overline{a}\langle x\rangle\rangle\} \frown \left\{t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^\pi : [P] \overset{t}{\Rightarrow} [Q]\right\}.$$

Now from the definition of the trace semantics (Definition 4.1.1)

$$Y = \{\langle \overline{a}\langle x\rangle\rangle\} \frown \mathcal{T}([P])$$

holds. Hence, all together we know that

$$\mathcal{T}([\overline{a}\langle x\rangle.P]) \overset{\text{Eq. 4.2}}{=} \{\langle\rangle\} \cup Y = \{\langle\rangle\} \cup \{\langle \overline{a}\langle x\rangle\rangle\} \frown \mathcal{T}([P])$$

holds.

Case $\mathcal{T}([P+Q])$**:** Let $P, Q \in \mathcal{P}^{\pi}$. From the definition of the trace semantics (Definition 4.1.1) we know, like in all the other cases, that

$$\mathcal{T}([P+Q]) = \left\{t \in \texttt{Traces} \mid \exists Q' \in \mathcal{P}^{\pi} : [P+Q] \overset{t}{\Rightarrow} [Q']\right\}$$

holds. Furthermore, the definition of the operational semantics shows us that there are only two rules which handle the choice operator. Thus, with the $E - SUM_L$ rule we know, that for all $Q' \in \mathcal{P}^{\pi}$ if $[P] \overset{t}{\Rightarrow} [Q']$ holds, then also $[P+Q] \overset{t}{\Rightarrow} [Q']$ holds. Hence we have the following subset relation:

$$\begin{aligned} &\left\{t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^{\pi} : [P] \overset{t}{\Rightarrow} [Q]\right\} \\ \subseteq &\left\{t \in \texttt{Traces} \mid \exists Q' \in \mathcal{P}^{\pi} : [P+Q] \overset{t}{\Rightarrow} [Q']\right\}, \end{aligned}$$

which means no more than $\mathcal{T}([P]) \subseteq \mathcal{T}([P+Q])$. Analogously, from the $E - SUM_R$ rule we get $\mathcal{T}([Q]) \subseteq \mathcal{T}([P+Q])$, thus, we know that $\mathcal{T}([P]) \cup \mathcal{T}([Q]) \subseteq \mathcal{T}([P+Q])$ holds.

Since there is no further rule which handles the choice operator, the behavior of both cases must have been at least all of the behavior of a choice so that $\mathcal{T}([P+Q]) \subseteq \mathcal{T}([P]) \cup \mathcal{T}([Q])$. Which all together shows that

$$\mathcal{T}([P+Q]) = \mathcal{T}([P]) \cup \mathcal{T}([Q])$$

holds.

Thus, Lemma 4.2.5 is proved. ■

Input process

Before giving a set notation for the traces of an input process, which simplifies its trace calculation, we present an idea why the trace semantics is most likely not compositional for input processes. Consider, for example, $P =_{\text{def}} \overline{x}\langle z\rangle \mid y(b)$. Hence, $\mathcal{T}([P]) = \texttt{pref}(\{\langle \overline{x}\langle z\rangle, y\, b\rangle \mid b \in \mathcal{N}\} \cup \{\langle y\, b, \overline{x}\langle z\rangle\rangle \mid b \in \mathcal{N}\})$. Furthermore,

consider $Q =_{\text{def}} a(x).P = a(x).(\overline{x}\langle z\rangle \mid y(b))$. We recognize that if Q receives the name y, new communication can be established. This additional behavior has to be calculated from the traces of P and added in a compositional way. As a first intuition, we can think of gaining the behavior by applying all possible substitutions for x to the traces of P. Afterwards, we would collect the traces where the input action directly succeeds the corresponding output action and vice versa. In these cases, we can think of a possible communication and so delete those actions, since the communication would result in a τ step. Now, if we consider $P' =_{\text{def}} \overline{x}\langle z\rangle.y(b) + y(b).\overline{x}\langle z\rangle$ we know $\mathcal{T}([P]) = \mathcal{T}([P'])$. However, if we examine $a(x).P'$ we are not allowed to add those traces. So it seems to be a problem to detect the possible additional behavior just by knowing the traces of P, since we do not have enough information within $\mathcal{T}([P])$.

Consequently, we just show an equality of the traces of an input process $a(x).P$ to a set with the empty trace and the traces from $P\{^y/_x\}$ prepended by an $a\,y$ action for all $y \in \mathcal{N}$. Since we calculate directly the traces of $P\{^y/_x\}$, the possible new communications are within the trace set of $P\{^y/_x\}$.

Lemma 4.2.6 (Calculation of input processes) *Given a process $P \in \mathcal{P}^\pi$ and names $a, x \in \mathcal{N}$, then*

$$\mathcal{T}([a(x).P]) = \{\langle\rangle\} \cup \{\langle a\,y\rangle^\frown s \mid s \in \mathcal{T}([P\{^y/_x\}]), y \in \mathcal{N}\}$$

holds. ♢

Proof: Let $P \in \mathcal{P}^\pi$ and $a, x \in \mathcal{N}$. Analogously to the case of the output action of Lemma 4.2.5, we know from Definition 4.1.1 and Lemma 4.2.1 that

$$\mathcal{T}([a(x).P]) = \{\langle\rangle\} \cup \underbrace{\left\{t \in \texttt{Traces} \setminus \{\langle\rangle\} \mid \exists Q \in \mathcal{P}^\pi : [a(x).P] \stackrel{t}{\Rightarrow} [Q]\right\}}_{Y =_{\text{def}}} \tag{4.3}$$

holds. Furthermore, there is also just one rule in Definition 2.2.10 which handles an input action. Thus, with the $E - IN$ rule we know

$$Y = \Big\{t \in \texttt{Traces} \setminus \{\langle\rangle\} \mid \exists Q \in \mathcal{P}^\pi, y \in \mathcal{N} : [a(x).P] \xrightarrow{a\,y} [P\{^y/_x\}] \wedge [P\{^y/_x\}] \stackrel{s}{\Rightarrow} [Q] \wedge t = \langle a\,y\rangle^\frown s\Big\}$$

holds. Since every trace has the prefix $a\,y$ for some $y \in \mathcal{N}$, there is – as in the output case of Lemma 4.2.5 – no possibility for the empty trace. Moreover, the

input transition is no further condition due to the fact that the substitution is applicable for every name $y \in \mathcal{N}$, so we can safely omit it. Thus, we know that

$$Y = \left\{ t \in \texttt{Traces} \mid \exists Q \in \mathcal{P}^{\pi}, y \in \mathcal{N} : [P\,\{^{y}/_{x}\}] \stackrel{s}{\Rightarrow} [Q] \wedge t = \langle a\,y \rangle^\frown s \right\}$$

holds. With the definition of the trace semantics (Definition 4.1.1) we can also write

$$Y = \{ t \in \texttt{Traces} \mid \exists s \in \mathcal{T}([P\,\{^{x}/_{y}\}]), y \in \mathcal{N} : t = \langle a\,y \rangle^\frown s \}\,.$$

Thus, all in all we know

$$\mathcal{T}([a(x).P]) \stackrel{\text{Eq. 4.3}}{=} \{\langle\rangle\} \cup Y = \{\langle\rangle\} \cup \{\langle a\,y \rangle^\frown s \mid s \in \mathcal{T}([P\,\{^{y}/_{x}\}]), y \in \mathcal{N}\}$$

holds. ■

Note that we know from Section 4.2.1 that $\mathcal{T}([P\,\{^{y}/_{x}\}]) = \mathcal{T}([P])\,\{^{y}/_{x}\}$ does not hold for every $y \in \mathcal{N}$ and thus, $\mathcal{T}([P\,\{^{y}/_{x}\}])$ cannot be replaced by $\mathcal{T}([P])\,\{^{y}/_{x}\}$ in Lemma 4.2.6.

Parallel composition

To show the compositionality of the parallel operator for processes without restriction and recursion, we take advantage of the at most countable infinity number of traces within a trace set. Thus, we can uniquely index every trace of a process P and a process Q to reach their indexed trace sets. Because of those indices we can remember which traces of P and Q are used to gain the behavior of the composition in an inductive calculation. This intuition is visualized in Figure 4.1.

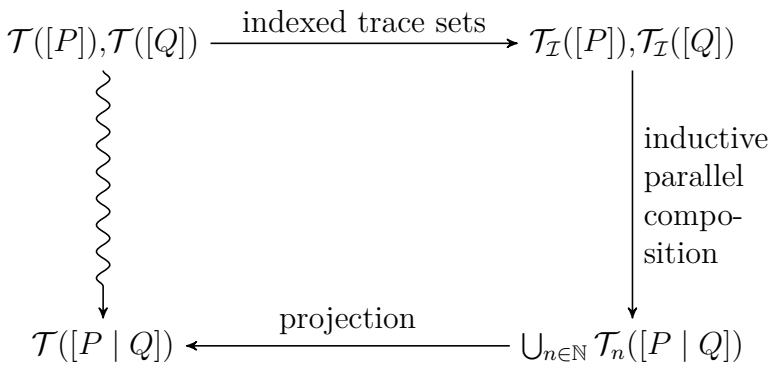

Figure 4.1: Visualization of the compositionality of the parallel operator (Part I).

The behavior of a parallel composition of two processes is their interleaving behavior and the behavior originated from the possible communications between those two processes. For the calculation of the behavior of a parallel composition it is very helpful that we only investigate processes without any restriction operators. Thus, no bound action can exist in any trace. Hence, we neither have to attend to the side condition of the $E - PAR_L$ respectively $E - PAR_R$ rule for the interleaving behavior nor mind the $E - CLOSE_L$ respectively $E - CLOSE_R$ rule of Figure 2.3 for the communication case.

Note that for the construction of the additional behavior originated of the communications, we do not have the same problem as described for the case of an input process. This is because in this case, we do have the information to which process the actions belong by calculating the communication behavior.

Following the intuition, we first inductively define the *inductive parallel composition* of a parallel composition by using the indexed trace sets of the components. That is, we memorize from which trace of the one and the other process in the parallel composition the trace of the parallel composed process is constructed and how many of the actions from the traces are used for this construction.

Definition 4.2.1 (Inductive parallel composition) Let $P, Q \in \mathcal{P}^{\pi}_{\texttt{resf}} \cap \mathcal{P}^{\pi}_{\texttt{recf}}$. Then the function family $\mathcal{T}_n : \mathcal{P}^{\pi}_{\alpha} \to \mathbb{P}\left((\mathbb{N}^2 \times \mathbb{N}^2) \times \texttt{Traces}\right)$ is defined with

$$
\begin{aligned}
\mathcal{T}_0([P \mid Q]) =_{\text{def}} & \left\{(((i_P, 0), (i_Q, 0)), \langle\rangle) \mid i_P, i_Q \in \mathbb{N}\right\} \\
\mathcal{T}_{n+1}([P \mid Q]) =_{\text{def}} & \Big\{(((i_P, j_P), (i_Q, j_Q)), s') \in (\mathbb{N}^2 \times \mathbb{N}^2) \times \texttt{Traces} \mid \\
& \quad \exists \left(\left((i_P, j'_P), \left(i_Q, j'_Q\right)\right), t'\right) \in \mathcal{T}_n([P \mid Q]), \\
& \qquad (i_P, s) \in \mathcal{T}_{\mathcal{I}}([P]), (i_Q, t) \in \mathcal{T}_{\mathcal{I}}([Q]) : \\
& \quad \text{PL} =_{\text{def}} \left(j_P = j'_P + 1 \wedge j_Q = j'_Q \wedge s' = t' \frown \langle s_{j_P} \rangle\right) \\
& \wedge \text{PR} =_{\text{def}} \left(j_P = j'_P \wedge j_Q = j'_Q + 1 \wedge s' = t' \frown \langle t_{j_Q} \rangle\right) \\
& \wedge \text{COM} =_{\text{def}} \left(j_P = j'_P + 1 \wedge j_Q = j'_Q + 1 \wedge s_{j_P} = \overline{t_{j_Q}} \wedge s' = t'\right) \\
& \wedge (\text{PR} \vee \text{PL} \vee \text{COM})\Big\}
\end{aligned}
$$

by induction over $n \in \mathbb{N}$. $*$

Thus, for a trace $t \in \texttt{Traces}$ of a parallel composition $P \mid Q \in \mathcal{P}^{\pi}_{\texttt{resf}} \cap \mathcal{P}^{\pi}_{\texttt{recf}}$, we save the index i_P as well as the index i_Q to memorize from which trace of P respectively Q the trace t is constructed. Additionally, we save with j_P and j_Q the

index within the trace of P respectively Q, till this index the trace is used for the construction.

Note that the index of the sets does not map the length of the containing traces, since with the conjunctive clause COM every τ step raises the index, but does not extend the trace. We just can say that for a trace $t \in \mathcal{T}_n([P \mid Q])$ the inequality $\#(t) \leq n$ holds. Moreover, the index n counts the inference steps of the operational semantics which had been done to reach the actual tuple of indices and trace.

We now connect this definition to the big-step semantics to have a tool for proving the compositionality of the parallel composition of restriction and recursion free processes.

Lemma 4.2.7 (Inductive parallel composition with big-step semantics) *Let $P, Q \in \mathcal{P}^{\pi}_{\mathtt{resf}} \cap \mathcal{P}^{\pi}_{\mathtt{recf}}$. Then for all $n \in \mathbb{N} \setminus \{0\}$:*

if a tuple $(((i_P, j_P), (i_Q, j_Q)), s') \in \mathcal{T}_n([P \mid Q])$ exists, then there are tuples

$$\begin{aligned}\left(\left((i_P, j'_P), \left(i_Q, j'_Q\right)\right), t'\right) &\in \mathcal{T}_{n-1}([P \mid Q]),\\ (i_P, s) &\in \mathcal{T}_{\mathcal{I}}([P]),\\ (i_Q, t) &\in \mathcal{T}_{\mathcal{I}}([Q])\end{aligned}$$

such that

$$\begin{aligned}&\Big(j_P = j'_P + 1 \wedge j_Q = j'_Q \wedge s' = t' \frown \langle s_{j_P}\rangle \wedge \exists P_1, P_2, Q_1 \in \mathcal{P}^{\pi} :\\ &\quad [P] \xRightarrow{\langle s_1,\ldots,s_{j_P}\rangle} [P_2] \wedge [Q] \xRightarrow{\langle t_1,\ldots,t_{j_Q}\rangle} [Q_1] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [P_1 \mid Q_1] \xrightarrow{s_{j_P}} [P_2 \mid Q_1]\Big)\\ \vee&\Big(j_P = j'_P \wedge j_Q = j'_Q + 1 \wedge s' = t' \frown \langle t_{j_Q}\rangle \wedge \exists P_1, Q_1, Q_2 \in \mathcal{P}^{\pi} :\\ &\quad [P] \xRightarrow{\langle s_1,\ldots,s_{j_P}\rangle} [P_1] \wedge [Q] \xRightarrow{\langle t_1,\ldots,t_{j_Q}\rangle} [Q_2] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [P_1 \mid Q_1] \xrightarrow{t_{j_Q}} [P_1 \mid Q_2]\Big)\\ \vee&\Big(j_P = j'_P + 1 \wedge j_Q = j'_Q + 1 \wedge s_{j_P} = \overline{t_{j_Q}} \wedge s' = t' \wedge \exists P_1, P_2, Q_1, Q_2 \in \mathcal{P}^{\pi} :\\ &\quad [P] \xRightarrow{\langle s_1,\ldots,s_{j_P}\rangle} [P_2] \wedge [Q] \xRightarrow{\langle t_1,\ldots,t_{j_Q}\rangle} [Q_2] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [P_1 \mid Q_1] \xrightarrow{\tau} [P_2 \mid Q_2]\Big)\end{aligned}$$

holds. ♢

Proof: Let $P, Q \in \mathcal{P}^{\pi}_{\mathtt{resf}} \cap \mathcal{P}^{\pi}_{\mathtt{recf}}$. So we know that neither a restriction operator nor a recursive call occurs in the given processes. Then we prove Lemma 4.2.7 by induction over the index of the inductive parallel composition sets.

Base case $n = 1$**:** Let $(((i_P, j_P), (i_Q, j_Q)), s') \in \mathcal{T}_1([P \mid Q])$. From the definition of $\mathcal{T}_1$, we know there are tuple $\left(\left((i_P, j'_P), \left(i_Q, j'_Q\right)\right), t'\right) \in \mathcal{T}_0([P \mid Q])$, $(i_P, s) \in \mathcal{T}_\mathcal{I}([P])$ and $(i_Q, t) \in \mathcal{T}_\mathcal{I}([Q])$ with

$$\begin{aligned}&\left(j_P = j'_P + 1 \wedge j_Q = j'_Q \wedge s' = t' \frown \langle s_{j_P} \rangle\right)\\ \vee &\left(j_P = j'_P \wedge j_Q = j'_Q + 1 \wedge s' = t' \frown \langle t_{j_Q} \rangle\right)\\ \vee &\left(j_P = j'_P + 1 \wedge j_Q = j'_Q + 1 \wedge s_{j_P} = \overline{t_{j_Q}} \wedge s' = t'\right).\end{aligned}$$

From the definition of $\mathcal{T}_0$, we know that $j'_P = j'_Q = 0$ and $t' = \langle\rangle$.

Case $\left(j_P = j'_P + 1 = 1 \wedge j_Q = j'_Q = 0 \wedge s' = t' \frown \langle s_{j_P} \rangle = \langle s_1 \rangle\right)$**:** Since $(i_P, s) \in \mathcal{T}_\mathcal{I}([P])$ and the trace sets are prefix closed, we know there are processes $P_1, P_2 \in \mathcal{P}^\pi$ such that $[P] \Rightarrow [P_1] \xrightarrow{s_1} [P_2]$ holds. Furthermore, with the $E - PAR_L$ rule, we know that every τ transition is also possible in a parallel composition. Moreover, since there is no restriction operator within the processes, and so $\mathtt{bn}(s_1) = \emptyset$, we know that also the s_1 transition is possible in the parallel context. Hence, $[P \mid Q] \xRightarrow{t'=\langle\rangle} [P_1 \mid Q] \xrightarrow{s_{j_P}=s_1} [P_2 \mid Q]$ holds and furthermore, we know that $[P] \xRightarrow{\langle s_1,\ldots,s_{j_P}\rangle=\langle s_1\rangle} [P_2]$ and since $\langle t_1, \ldots, t_0 \rangle = \langle\rangle$, $[Q] \xRightarrow{\langle t_1,\ldots,t_{j_Q}\rangle=\langle\rangle} [Q]$ holds.

Case $\left(j_P = j'_P = 0 \wedge j_Q = j'_Q + 1 = 1 \wedge s' = t' \frown \langle t_{j_Q} \rangle = \langle t_1 \rangle\right)$**:** Analogously to the prior case, by application of the $E - PAR_R$ rule.

Case $\left(j_P = j'_P + 1 = 1 \wedge j_Q = j'_Q + 1 = 1 \wedge s_1 = \overline{t_1} \wedge s' = t' = \langle\rangle\right)$**:** Like in the other cases we know from $(i_P, s) \in \mathcal{T}_\mathcal{I}([P])$ and $(i_Q, t) \in \mathcal{T}_\mathcal{I}([Q])$ and that trace sets are prefix closed that there exists processes $P_1, P_2, Q_1, Q_2 \in \mathcal{P}^\pi$, with $[P] \Rightarrow [P_1] \xrightarrow{s_1} [P_2]$ and $[Q] \Rightarrow [Q_1] \xrightarrow{t_1} [Q_2]$. Hence, with the $E - PAR_L$ and $E - PAR_R$ rule we know $[P \mid Q] \Rightarrow [P_1 \mid Q_1]$ holds and since $s_1 = \overline{t_1}$, we know, with the $E - COM_L$ (respectively $E - COM_R$) rule, that $[P_1 \mid Q_1] \xrightarrow{\tau} [P_2 \mid Q_2]$ holds. So all in all we know $[P \mid Q] \xRightarrow{t'=\langle\rangle} [P_1 \mid Q_1] \xrightarrow{\tau} [P_2 \mid Q_2]$ and in addition to that, $[P] \xRightarrow{\langle s_1,\ldots,s_{j_P}\rangle=\langle s_1\rangle} [P_2]$ and $[Q] \xRightarrow{\langle t_1,\ldots,t_{j_Q}\rangle=\langle t_1\rangle} [Q_2]$ holds.

Induction hypothesis: For a given $n \in \mathbb{N} \setminus \{0\}$ Lemma 4.2.7 holds.

Induction step $n \mapsto n + 1$**:** Let $(((i_P, j_P), (i_Q, j_Q)), s') \in \mathcal{T}_{n+1}([P \mid Q])$. From the definition of inductive parallel composition set $\mathcal{T}_{n+1}$ we know there exists tuple $(i_P, s) \in \mathcal{T}_\mathcal{I}([P])$, $(i_Q, t) \in \mathcal{T}_\mathcal{I}([Q])$, and $\left(\left((i_P, j'_P), \left(i_Q, j'_Q\right)\right), t'\right) \in \mathcal{T}_n([P \mid Q])$

such that

$$\begin{aligned}&\Big(j_P = j'_P + 1 \wedge j_Q = j'_Q \wedge s' = t' \frown \langle s_{j_P} \rangle\Big)\\ \vee &\Big(j_P = j'_P \wedge j_Q = j'_Q + 1 \wedge s' = t' \frown \langle t_{j_Q} \rangle\Big)\\ \vee &\Big(j_P = j'_P + 1 \wedge j_Q = j'_Q + 1 \wedge s_{j_P} = \overline{t_{j_Q}} \wedge s' = t'\Big).\end{aligned} \tag{4.4}$$

holds. Moreover we know from the induction hypothesis that there are tuple $\Big(\big((i_P, j''_P), (i_Q, j''_Q)\big), t''\Big) \in \mathcal{T}_{n-1}([P \mid Q])$, $(i_P, u) \in \mathcal{T}_{\mathcal{I}}([P])$ and $(i_Q, v) \in \mathcal{T}_{\mathcal{I}}([Q])$ such that

$$\begin{aligned}&\Big(j'_P = j''_P + 1 \wedge j'_Q = j''_Q \wedge t' = t'' \frown \langle u_{j'_P} \rangle \wedge \exists P_1, P_2, Q_1 \in \mathcal{P}^\pi :\\ &[P] \xrightarrow{\langle u_1, \ldots, u_{j'_P} \rangle} [P_2] \wedge [Q] \xrightarrow{\langle v_1, \ldots, v_{j'_Q} \rangle} [Q_1] \wedge [P \mid Q] \overset{t''}{\Rightarrow} [P_1 \mid Q_1] \xrightarrow{u_{j'_P}} [P_2 \mid Q_1]\Big)\\ \vee&\Big(j'_P = j''_P \wedge j'_Q = j''_Q + 1 \wedge t' = t'' \frown \langle v_{j'_Q} \rangle \wedge \exists P_1, Q_1, Q_2 \in \mathcal{P}^\pi :\\ &[P] \xrightarrow{\langle u_1, \ldots, u_{j'_P} \rangle} [P_1] \wedge [Q] \xrightarrow{\langle v_1, \ldots, v_{j'_Q} \rangle} [Q_2] \wedge [P \mid Q] \overset{t''}{\Rightarrow} [P_1 \mid Q_1] \xrightarrow{v_{j'_Q}} [P_1 \mid Q_2]\Big)\\ \vee&\Big(j'_P = j''_P + 1 \wedge j'_Q = j''_Q + 1 \wedge u_{j'_P} = \overline{v_{j'_Q}} \wedge t' = t'' \wedge \exists P_1, P_2, Q_1, Q_2 \in \mathcal{P}^\pi :\\ &[P] \xrightarrow{\langle u_1, \ldots, u_{j'_P} \rangle} [P_2] \wedge [Q] \xrightarrow{\langle v_1, \ldots, v_{j'_Q} \rangle} [Q_2] \wedge [P \mid Q] \overset{t''}{\Rightarrow} [P_1 \mid Q_1] \xrightarrow{\tau} [P_2 \mid Q_2]\Big).\end{aligned}$$

holds. Hence, we know in every case there are processes $P_2, Q_2 \in \mathcal{P}^\pi$ such that $[P] \xrightarrow{\langle u_1, \ldots, u_{j'_P} \rangle} [P_2]$, $[Q] \xrightarrow{\langle v_1, \ldots, v_{j'_Q} \rangle} [Q_2]$ and $[P \mid Q] \overset{t'}{\Rightarrow} [P_2 \mid Q_2]$ holds. Furthermore, since $(i_P, s) \in \mathcal{T}_{\mathcal{I}}([P])$ and $(i_P, u) \in \mathcal{T}_{\mathcal{I}}([P])$ and the indexing of the traces is unique, we know that $s = u$ and also $t = v$ holds. Since Formula 4.4 holds, we again consider three cases:

Case $\Big(j_P = j'_P + 1 \wedge j_Q = j'_Q \wedge s' = t' \frown \langle s_{j_P} \rangle\Big)$**:** Since $(i_P, s) \in \mathcal{T}_{\mathcal{I}}([P])$ and the trace sets are prefix closed and additionally $j_p = j'_P + 1$ we know there exists a process $P_3 \in \mathcal{P}^\pi$, such that $[P] \xrightarrow{\langle s_1, \ldots, s_{j'_p} \rangle} [P_2] \xrightarrow{s_{j_P}} [P_3]$ holds. Again, with the $E - PAR_L$ rule and the same arguments as within the base case, we know that the s_{j_P} transition is possible in the parallel context. Hence, $[P \mid Q] \overset{t'}{\Rightarrow} [P_2 \mid Q_2] \xrightarrow{s_{j_P}} [P_3 \mid Q_2]$ holds and furthermore, we know that $[P] \xrightarrow{\langle s_1, \ldots, s_{j_P} \rangle} [P_3]$ and $[Q] \xrightarrow{\langle t_1, \ldots, t_{j_Q} \rangle} [Q_2]$ holds, since $j_Q = j'_Q$.

Case $\Big(j_P = j'_P \wedge j_Q = j'_Q + 1 \wedge s' = t' \frown \langle t_{j_Q} \rangle\Big)$**:** This is again proved similarly to the prior case, by application of the $E - PAR_R$ rule.

Case $\Big(j_P = j'_P + 1 \wedge j_Q = j'_Q + 1 \wedge s_{j_P} = \overline{t_{j_Q}} \wedge s' = t'\Big)$**:** Like within the other

cases, we know from $(i_P, s) \in \mathcal{T}_{\mathcal{I}}([P])$ and $(i_Q, t) \in \mathcal{T}_{\mathcal{I}}([Q])$ and that trace sets are prefix closed that there exists processes $P_3, Q_3 \in \mathcal{P}^{\pi}$, with $[P] \xRightarrow{\langle s_1,\ldots,s_{j'_P}\rangle} [P_2] \xrightarrow{s_{j_P}} [P_3]$ and $[Q] \xRightarrow{\langle t_1,\ldots,t_{j'_Q}\rangle} [Q_2] \xrightarrow{t_{j_Q}} [Q_3]$. Since $s_{j_P} = \overline{t_{j_Q}}$ holds, we know with the $E - COM_L$ (respectively $E - COM_R$) rule that $[P_2 \mid Q_2] \xrightarrow{\tau} [P_3 \mid Q_3]$. So again $[P \mid Q] \overset{t'}{\Rightarrow} [P_2 \mid Q_2] \xrightarrow{\tau} [P_3 \mid Q_3]$ holds.

So we proved by induction that Lemma 4.2.7 holds for every $n \in \mathbb{N} \setminus \{0\}$. ■

Furthermore, all of this behavior is invertible. We know about the traces of a parallel composition that they have to be inferred from the rules of Figure 2.3. Therefore, we can follow every inference step and raise the related counter to receive the inductive parallel composition. Now we know

$$\mathcal{T}_{\mathcal{I}}([P]) = \left\{ (i_P, s) \in \mathbb{N} \times \texttt{Traces} \mid \exists j \in \mathbb{N} : ((i_P, j), (i_Q, 0), s) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q]) \right\}$$

and similarly for $\mathcal{T}_{\mathcal{I}}([Q])$. Thus, we can gain the traces of P and Q by a projection. This idea is visualized in Figure 4.2.

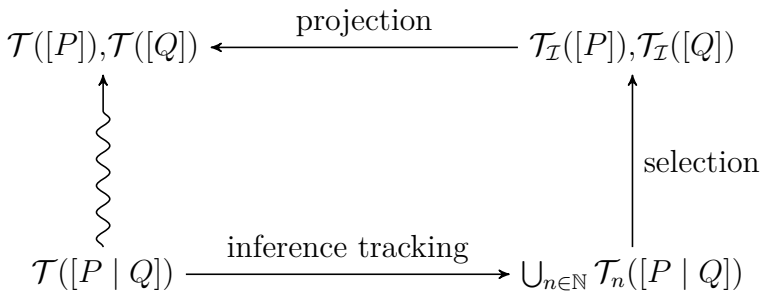

Figure 4.2: Visualization of the compositionality of the parallel operator (Part II).

With Lemma 4.2.7 we can show that there is a trace equality such that the traces of the parallel composition of processes without restriction and recursion is equal to the traces within the union of the inductive parallel composition sets.

Lemma 4.2.8 (Compositionality of parallel composition) *Given two processes $P, Q \in \mathcal{P}^{\pi}_{\texttt{resf}} \cap \mathcal{P}^{\pi}_{\texttt{recf}}$. Then for the traces of a parallel composition*

$$\mathcal{T}([P \mid Q]) = \left\{ s \in \texttt{Traces} \mid \exists (\cdot, s) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q]) \right\}$$

holds. ♢

Proof: Let $P, Q \in \mathcal{P}^{\pi}_{\texttt{resf}} \cap \mathcal{P}^{\pi}_{\texttt{recf}}$. For the $\supseteq$-inclusion we know from Lemma 4.2.7 that for all $t \in \left\{s \in \texttt{Traces} \mid \exists\,(\cdot, s) \in \bigcup_{n\in\mathbb{N}\setminus\{0\}} \mathcal{T}_n([P \mid Q])\right\}$ there are $P_2, Q_2 \in \mathcal{P}^{\pi}$ such that $[P \mid Q] \stackrel{t}{\Rightarrow} [P_2 \mid Q_2]$ and so $t \in \mathcal{T}([P \mid Q])$ holds. Since with $t \in \{s \in \texttt{Traces} \mid \exists\,(\cdot, s) \in \mathcal{T}_0([P \mid Q])\}$ follows $t = \langle\rangle$ and from Lemma 4.2.1 we know that the empty trace is included in every trace set, we know that the fully inclusion holds.

For the $\subseteq$-inclusion, we induce over the length of a trace $s \in \mathcal{T}([P \mid Q])$:

Base case $s = \langle\rangle$**:** Since $\mathcal{T}_0([P \mid Q]) = \{(((i_P, 0), (i_Q, 0)), \langle\rangle) \mid i_P, i_Q \in \mathbb{N}\}$, we know $s \in \{t \in \texttt{Traces} \mid \exists\,(\cdot, t) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q])\}$.

Base case $s = \langle\alpha\rangle$ **with** $\alpha \in \texttt{Act} \setminus \{\tau\}$**:** From the definition of the big-step semantics (Definition 3.1.2) we know that there are processes $P_1, P_2, Q_1, Q_2 \in \mathcal{P}^{\pi}$ such that $[P \mid Q] \Rightarrow [P_1 \mid Q_1] \stackrel{\alpha}{\to} [P_2 \mid Q_2]$. Since we have no restriction operator, we know the τ transitions within $[P \mid Q] \Rightarrow [P_1 \mid Q_1]$ can either be produced from the $E - COM_L$ (respectively $E - COM_R$) rule or from a τ prefix with the $E - PAR_L$ (respectively $E - PAR_R$) rule. So there is a number $m \in \mathbb{N}$ and actions $\alpha_1, \ldots, \alpha_m \in \texttt{Act} \setminus \{\tau\}$ and $\beta_1, \ldots, \beta_m \in \texttt{Act} \setminus \{\tau\}$ such that $[P] \stackrel{\langle\alpha_1,\ldots,\alpha_m\rangle}{\Longrightarrow} [P_1]$ and $[Q] \stackrel{\langle\beta_1,\ldots,\beta_m\rangle}{\Longrightarrow} [Q_1]$ holds. Furthermore, there are just the $E - PAR_L$ and $E - PAR_R$ rule, which can produce the visible action α. So we know $[P_1] \stackrel{\alpha}{\to} [P_2]$ or $[Q_1] \stackrel{\alpha}{\to} [Q_2]$ holds. Since both cases can be handled analogously, we just describe the first case. In this case we know $[P] \stackrel{\langle\alpha_1,\ldots,\alpha_m,\alpha\rangle}{\Longrightarrow} [P_2]$ and $Q_1 = Q_2$ holds. Hence, there are numbers $i_P, i_Q \in \mathbb{N}$ with $(i_P, \langle\alpha_1, \ldots, \alpha_m, \alpha\rangle) \in \mathcal{T}_{\mathcal{I}}([P])$ and $(i_Q, \langle\beta_1, \ldots, \beta_m\rangle) \in \mathcal{T}_{\mathcal{I}}([Q])$. We also know that $(((i_P, 0), (i_Q, 0)), \langle\rangle) \in \mathcal{T}_0([P \mid Q])$ holds so that we can step by step raise the counter for every α_i and β_i with $i \in \{1, \ldots, m\}$ by taking in every step the COM conjunctive clause of Definition 4.2.1. Hence, $(((i_P, m), (i_Q, m)), \langle\rangle) \in \mathcal{T}_m([P \mid Q])$ holds. So we see that there is a tuple $(((i_P, m+1), (i_Q, m)), \langle\alpha\rangle) \in \mathcal{T}_{m+1}([P \mid Q])$, because the PL conjunctive clause of the definition of $\mathcal{T}_{m+1}$ is fulfilled. So we know $s \in \{t \in \texttt{Traces} \mid \exists\,(\cdot, t) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q])\}$ holds.

Induction hypothesis: For a given $n \in \mathbb{N}$, we know that for all $s \in \mathcal{T}([P \mid Q])$ with $\#(s) \leq n$ holds that $s \in \{t \in \texttt{Traces} \mid \exists\,(\cdot, t) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q])\}$.

Induction step $n \mapsto n + 1$**:** So $\#(s) = n + 1$. Again we know from the definition of the big-step semantics that there are processes $P_1, P_2, Q_1, Q_2 \in \mathcal{P}^{\pi}$ such

that $[P \mid Q] \overset{s'}{\Rightarrow} [P_1 \mid Q_1] \xrightarrow{s_{n+1}} [P_2 \mid Q_2]$ with $s' = \langle s_1, \ldots, s_n \rangle$ and $s_{n+1} \neq \tau$. Furthermore, since the cases of the empty trace and a trace with just one visible action are already done in the base cases, we consider $\#(s) \geq 2$. Hence, we know $s' \neq \langle\rangle$ holds. So there have to be processes $P_1', P_2', Q_1', Q_2' \in \mathcal{P}^\pi$ with $[P \mid Q] \xRightarrow{\langle s_1,\ldots,s_{n-1}\rangle} [P_1' \mid Q_1'] \xrightarrow{s_n} [P_2' \mid Q_2'] \Rightarrow [P_1 \mid Q_1] \xrightarrow{s_{n+1}} [P_2 \mid Q_2]$. We now would like to find the suitable indices for $(\cdot, s') \in \mathcal{T}_x([P \mid Q])$ so that with this indices really the case $[P \mid Q] \xRightarrow{\langle s_1,\ldots,s_{n-1}\rangle} [P_1' \mid Q_1'] \xrightarrow{s_n} [P_2' \mid Q_2']$ is described. So the visible action is the last step of the deduction and P_2' and Q_2' are really reached.

Since $[P \mid Q] \overset{s'}{\Rightarrow} [P_1 \mid Q_1]$ holds, we know with the induction hypothesis that $s' \in \{t \in \texttt{Traces} \mid \exists\, (\cdot, t) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q])\}$. Since $s' \neq \langle\rangle$, we know there are $k_P, l_P, k_Q, l_Q, n' \in \mathbb{N} \setminus \{0\}$ such that $(((k_P, l_P), (k_Q, l_Q)), s') \in \mathcal{T}_{n'}([P \mid Q])$. So, from Lemma 4.2.7, we know there are tuple $\Big(\big((k_P, l_P'), \big(k_Q, l_Q'\big)\big), t'\Big) \in \mathcal{T}_{n'-1}([P \mid Q])$, $(k_P, r) \in \mathcal{T}_{\mathcal{I}}([P])$ and $(k_Q, t) \in \mathcal{T}_{\mathcal{I}}([Q])$ such that

$$
\begin{aligned}
&\Big(l_P = l_P' + 1 \wedge l_Q = b_Q' \wedge s' = t' \frown \langle r_{l_P} \rangle \wedge \exists R_1, R_2, S_1 \in \mathcal{P}^\pi : \\
&\quad [P] \xRightarrow{\langle r_1,\ldots,r_{l_P}\rangle} [R_2] \wedge [Q] \xRightarrow{\langle t_1,\ldots,t_{l_Q}\rangle} [S_1] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [R_1 \mid S_1] \xrightarrow{r_{l_P}} [R_2 \mid S_1]\Big) \\
\vee&\Big(l_P = l_P' \wedge l_Q = l_Q' + 1 \wedge s' = t' \frown \langle t_{l_Q} \rangle \wedge \exists R_1, S_1, S_2 \in \mathcal{P}^\pi : \\
&\quad [P] \xRightarrow{\langle r_1,\ldots,r_{l_P}\rangle} [R_1] \wedge [Q] \xRightarrow{\langle t_1,\ldots,t_{l_Q}\rangle} [S_2] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [R_1 \mid S_1] \xrightarrow{t_{l_Q}} [R_1 \mid S_2]\Big) \\
\vee&\Big(l_P = l_P' + 1 \wedge l_Q = l_Q' + 1 \wedge r_{l_P} = \overline{t_{l_Q}} \wedge s' = t' \wedge \exists R_1, R_2, S_1, S_2 \in \mathcal{P}^\pi : \\
&\quad [P] \xRightarrow{\langle r_1,\ldots,r_{l_P}\rangle} [R_2] \wedge [Q] \xRightarrow{\langle t_1,\ldots,t_{l_Q}\rangle} [S_2] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [R_1 \mid S_1] \xrightarrow{\tau} [R_2 \mid S_2]\Big)
\end{aligned}
$$

holds. Hence, we know there are processes $R_2, S_2 \in \mathcal{P}^\pi$ with $[P \mid Q] \overset{s'}{\Rightarrow} [R_2 \mid S_2]$, but we do not know if these are the right indices such that $R_2 = P_2'$, $S_2 = Q_2'$ holds and s_n is really the last deduction step. This is because arbitrary many τ steps could be performed after the visible action s_n with the COM conjunctive clause of Definition 4.2.1 and nevertheless $s' \in \{t \in \texttt{Traces} \mid \exists\, (\cdot, t) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q])\}$ would hold. But we can revert every of the excessive τ steps, so we know there exists $m, j_P', j_P'', j_Q', j_Q'' \in \mathbb{N}$ with $m \leq n'$,$j_P'' \leq l_P'$ and $j_Q'' \leq l_Q'$ such that $\Big(\big((k_P, j_P'), \big(k_Q, j_Q'\big)\big), s'\Big) \in \mathcal{T}_m([P \mid Q])$, $\Big(\big((k_P, j_P''), \big(k_Q, j_Q''\big)\big), t'\Big) \in \mathcal{T}_{m-1}([P \mid Q])$, $(k_P, r) \in \mathcal{T}_{\mathcal{I}}([P])$ and

$(k_Q, t) \in \mathcal{T}_{\mathcal{I}}([Q])$ exists such that

$$\Big(j'_P = j''_P + 1 \wedge j'_Q = j''_Q \wedge r_{j'_P} = s_n \wedge s' = t' \frown \langle s_n \rangle$$
$$\wedge [P] \xrightarrow{\langle r_1,\ldots,r_{j'_P}\rangle} [P'_2] \wedge [Q] \xrightarrow{\langle t_1,\ldots,t_{j'_Q}\rangle} [Q'_1] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [P'_1 \mid Q'_1] \xrightarrow{s_n} [P'_2 \mid Q'_1]\Big)$$
$$\vee\Big(j'_P = j''_P \wedge j'_Q = j''_Q + 1 \wedge t_{j'_Q} = s_n \wedge s' = t' \frown \langle s_n \rangle$$
$$\wedge [P] \xrightarrow{\langle r_1,\ldots,r_{j'_P}\rangle} [P'_1] \wedge [Q] \xrightarrow{\langle t_1,\ldots,t_{j'_Q}\rangle} [Q'_2] \wedge [P \mid Q] \overset{t'}{\Rightarrow} [P'_1 \mid Q'_1] \xrightarrow{s_n} [P'_1 \mid Q'_2]\Big)$$

holds. The conjunctive clause which handles the communication case is not satisfiable since the last step has to be a visible action and is due to that omitted.

Since $t' = \langle s_1, \ldots, s_{n-1} \rangle$, we found the suitable indices such that

$$[P \mid Q] \xrightarrow{\langle s_1,\ldots,s_{n-1}\rangle} [P'_1 \mid Q'_1] \xrightarrow{s_n} [P'_2 \mid Q'_2]$$

holds. We now have to raise the indices for the τ steps within $[P'_2 \mid Q'_2] \Rightarrow [P_1 \mid Q_1]$ and the visible action $[P_1 \mid Q_1] \xrightarrow{s_{n+1}} [P_2 \mid Q_2]$. Again we know that the visible action s_{n+1} must be produced from the $E - PAR_L$ (respectively $E - PAR_R$) rule and since both cases are similar, we concentrate on the $E - PAR_L$ case.

To find the suitable indices such that $[P \mid Q] \overset{s'}{\Rightarrow} [P'_2 \mid Q'_2] \Rightarrow [P_1 \mid Q_1] \xrightarrow{s_{n+1}} [P_2 \mid Q_2]$ holds, we can – as in the second base case – raise the indices for every τ step within $[P'_2 \mid Q'_2] \Rightarrow [P_1 \mid Q_1]$ with the COM conjunctive clause. But in this case, we do not know, how r and t look like. If they are long enough so that every action which is needed in the premise of the $E - COM_L$ (respectively $E - COM_R$) rule to deduct the τ steps within $[P'_2 \mid Q'_2] \Rightarrow [P_1 \mid Q_1]$ and as well the action s_{n+1} is contained, then we can just raise the counter. Otherwise, if r and t do not contain every needed action, we know $[P \mid Q] \xrightarrow{\langle s_1,\ldots,s_{n-1}\rangle} [P'_1 \mid Q'_1] \xrightarrow{s_n} [P'_2 \mid Q'_2] \Rightarrow [P_1 \mid Q_1] \xrightarrow{s_{n+1}} [P_2 \mid Q_2]$ holds and that those steps can just be performed by applying the $E - PAR_L$ (respectively $E - PAR_R$) or $E - COM_L$ (respectively $E - COM_R$) rules. Hence, we know, there have to be traces $u, v \in \texttt{Traces}$ with $[P] \overset{r}{\Rightarrow} [P'_2] \xrightarrow{u \frown \langle s_{n+1}\rangle} [P_2]$ and $[Q] \overset{t}{\Rightarrow} [Q'_2] \overset{v}{\Rightarrow} [Q_2]$, since we consider the $E - PAR_L$ case. Hence, we know that nothing changes, if we use the tuple $(i_P, u') \in \mathcal{T}_{\mathcal{I}}([P])$ with $u' =_{\text{def}} r \frown u \frown s_{n+1}$ instead of $(k_P, r) \in \mathcal{T}_{\mathcal{I}}([P])$ and $(i_Q, v') \in \mathcal{T}_{\mathcal{I}}([Q])$ with $v' =_{\text{def}} t \frown v$ instead of $(k_Q, t) \in$

$\mathcal{T}_{\mathcal{I}}([Q])$. With this we can, as in the base case, increase j'_P and j'_Q by one for every τ deduction within $[P'_2 \mid Q'_2] \Rightarrow [P_1 \mid Q_1]$. So we found the suitable numbers $m', i_P, j_P, i_Q, j_Q \in \mathbb{N}$ such that $(((i_P, j_P), (i_Q, j_Q)), s') \in \mathcal{T}_{m'}([P \mid Q])$, $(i_P, u') \in \mathcal{T}_{\mathcal{I}}([P])$ and $(i_Q, v') \in \mathcal{T}_{\mathcal{I}}([Q])$, with $m' \geq m$, $j_P \geq j'_P$ and $j_Q \geq j'_Q$ holds for $[P \mid Q] \xRightarrow{\langle s_1,\ldots,s_{n-1}\rangle} [P'_1 \mid Q'_1] \xrightarrow{s_n} [P'_2 \mid Q'_2] \Rightarrow [P_1 \mid Q_1]$. Hence, we know that $(((i_P, j_P + 1), (i_Q, j_Q)), s) \in \mathcal{T}_{m'+1}([P \mid Q])$ holds, since the PL conjunctive clause holds, because of considering the $E - PAR_L$ case. So $s \in \{t \in \texttt{Traces} \mid \exists\,(\cdot, t) \in \bigcup_{n\in\mathbb{N}} \mathcal{T}_n([P \mid Q])\}$ holds.

So we proved that the parallel composition is compositional for processes without restriction and recursion. ■

The next section dealts with the last unconsidered operator of recursion-free π-calculus processes.

Restriction

To show an idea for the compositionality of the trace semantics for the restriction operator, we firstly define a function called *restriction set* for two names and a set of traces. The idea of this approach is to take the traces of P for a process $\underline{\texttt{new}}\, a\ P$ and replace the name a by every fresh name $a' \notin \texttt{fn}(P)$. This does not cause any trouble because of Lemma 4.2.3 respectively Lemma 3.3.3 and Lemma 3.3.4. We have to consider every $a' \notin \texttt{fn}(P)$ since we know that $\underline{\texttt{new}}\, a'\, (P\,\{^{a'}/_{a}\}) \in [\underline{\texttt{new}}\, a\ P]$ holds for all of such a'. Then, we can change the first occurrence of an output action $\overline{x}\langle a'\rangle$ to $\overline{x}(a')$ if existent. This yields from the $E - OPEN$ rule of Figure 2.3 and from Convention 3.3.1. After that work is done, we can filter out the desired traces. That is, we know that after an application of the $E - OPEN$ rule the restriction operator is omitted, thus every possible behavior is allowed. Otherwise, the name a' is not allowed to occur as subject of an action. Hence, we only collect those traces, where something is send over the channel a' if a' has firstly been an object of a bound output action. The proof of the equality of the union of this restriction sets to the trace set of the restricted process, is not completely finish. So we only give ideas and limitations for this case.

Definition 4.2.2 (Restriction set) For $a, a' \in \mathcal{N}$ and $P \in \mathcal{P}^{\pi}$ we define $\texttt{res} : \mathcal{N} \times \mathcal{N} \times \mathbb{P}(\texttt{Traces}) \to \mathbb{P}(\texttt{Traces})$ with

$$\texttt{res}(a, a', \mathcal{T}([P])) =_{\text{def}} \Big\{ s \in \texttt{bind}(a', (\mathcal{T}([P])\,\{a' \leftrightarrow a\})) \mid$$

$$\forall i \in \mathbb{N} : a' \in \mathtt{sub}(s_i) \Rightarrow \exists j \in \mathbb{N} : a' \in \mathtt{bn}(s_j) \wedge j < i\Big\}$$

as the *restriction set* of a, a', and $\mathcal{T}([P])$. $*$

To investigate the equality of a union of restriction sets to the traces of a corresponding restricted process, we list a lemma which reduces one inclusion of this problem to the big-step semantics.

Lemma 4.2.9 (From restriction set to restriction) *Given $P \in \mathcal{P}^{\pi}_{\mathtt{recf}}$ and $a \in \mathcal{N}$. For a name $a' \in \mathcal{N} \setminus \mathtt{fn}(\underline{\mathtt{new}}\, a\ P)$ we know*

$$t \in \mathtt{res}(a, a', \mathcal{T}([P])), [P] \stackrel{s}{\Rightarrow} [Q] \ \textit{ with } t = \mathtt{bind}(a', s\,\{a' \leftrightarrow a\}) \textit{ and } a' \notin \mathtt{bn}(t) \Rightarrow \nexists i \in \mathbb{N}\colon a' \in \mathtt{obj}(t_i) \wedge t_i \in \mathtt{In}$$

implies

(I) $[\underline{\mathtt{new}}\, a'\,(P\,\{a' \leftrightarrow a\})] \stackrel{t}{\Rightarrow} [\underline{\mathtt{new}}\, a'\,(Q\,\{a' \leftrightarrow a\})]$ *with* $a' \notin \mathtt{bn}(t)$ *or*

(II) $[\underline{\mathtt{new}}\, a'\,(P\,\{a' \leftrightarrow a\})] \stackrel{t}{\Rightarrow} [Q\,\{a' \leftrightarrow a\}]$ *with* $a' \in \mathtt{bn}(t)$

holds. $\diamondsuit$

Proof: Let $P \in \mathcal{P}^{\pi}_{\mathtt{recf}}$, $a \in \mathcal{N}$, $a' \in \mathcal{N} \setminus \mathtt{fn}(\underline{\mathtt{new}}\, a\ P)$, $t \in \mathtt{res}(a, a', \mathcal{T}([P]))$ and $[P] \stackrel{s}{\Rightarrow} [Q]$ with $t = \mathtt{bind}(a', s\,\{a' \leftrightarrow a\})$ and $a' \notin \mathtt{bn}(t) \Rightarrow \nexists i \in \mathcal{N}\colon a' \in \mathtt{obj}(t_i) \wedge t_i \in \mathtt{In}$. We proceed by induction over the length $n \in \mathbb{N}$ of trace s.

Base case $n = 0$**:** Hence, $s = \langle\rangle$ and so $t = \mathtt{bind}(a', \langle\rangle\,\{a' \leftrightarrow a\}) = \mathtt{bind}(a', \langle\rangle) = \langle\rangle$. Since $[P] \stackrel{\langle\rangle}{\Rightarrow} [Q]$ holds, we know with Lemma 3.3.1 that $[P\,\{a' \leftrightarrow a\}] \stackrel{\langle\rangle}{\Rightarrow} [Q\,\{a' \leftrightarrow a\}]$. With multiple application of the $E - RES$ rule of Figure 2.3, we know $[\underline{\mathtt{new}}\, a'\,(P\,\{a' \leftrightarrow a\})] \stackrel{\langle\rangle}{\Rightarrow} [\underline{\mathtt{new}}\, a'\,(Q\,\{a' \leftrightarrow a\})]$ with $a' \notin \mathtt{n}(t)$.

Induction hypothesis: For an arbitrary number $n \in \mathbb{N}$, we know for all processes $P \in \mathcal{P}^{\pi}$ and names $a \in \mathcal{N}$ that for a name $a' \in \mathcal{N} \setminus \mathtt{fn}(\underline{\mathtt{new}}\, a\ P)$ that if a trace $t \in \mathtt{res}(a, a', \mathcal{T}([P]))$ with $t = \mathtt{bind}(a', s\,\{a' \leftrightarrow a\})$ such that $a' \notin \mathtt{bn}(t) \Rightarrow \nexists i \in \mathcal{N}\colon a' \in \mathtt{obj}(t_i) \wedge t_i \in \mathtt{In}$ holds and $[P] \stackrel{s}{\Rightarrow} [Q]$ and $\#(s) \leq n$ exists, then $[\underline{\mathtt{new}}\, a'\,(P\,\{a' \leftrightarrow a\})] \stackrel{t}{\Rightarrow} [\underline{\mathtt{new}}\, a'\,(Q\,\{a' \leftrightarrow a\})]$ with $a' \notin \mathtt{bn}(t)$ or $[\underline{\mathtt{new}}\, a'\,(P\,\{a' \leftrightarrow a\})] \stackrel{t}{\Rightarrow} [Q\,\{a' \leftrightarrow a\}]$ with $a' \in \mathtt{bn}(t)$ holds.

Induction step $n \mapsto n+1$**:** Thus, there is a trace $s' \in \mathtt{Traces}$ and a visible action $\alpha' \in \mathtt{Act} \setminus \{\tau\}$ with $s = s' \frown \langle \alpha' \rangle$. Thus, $t = \mathtt{bind}(a', (s' \frown \langle \alpha' \rangle)\{a' \leftrightarrow a\}) = \mathtt{bind}(a', s'\{a' \leftrightarrow a\} \frown \langle \alpha' \rangle \{a' \leftrightarrow a\})$. Since $[P] \stackrel{s}{\Rightarrow} [Q]$ we know there exists processes $Q_1, Q_2 \in \mathcal{P}^\pi$ with $[P] \stackrel{s'}{\Rightarrow} [Q_1] \stackrel{\alpha'}{\rightarrow} [Q_2] \Rightarrow [Q]$. With Convention 3.3.1 we know $a, a' \notin \mathtt{bn}(s') \cup \mathtt{bn}(\alpha')$. Let $t' =_{\text{def}} \mathtt{bind}(a', s'\{a' \leftrightarrow a\})$ and $\alpha =_{\text{def}} \{a' \leftrightarrow a\}(\alpha')$. Since $t \in \mathtt{res}(a, a', \mathcal{T}([P]))$ and t' is a prefix of t we know $t' \in \mathtt{res}(a, a', \mathcal{T}([P]))$ and so the induction hypothesis yields that (I) $[\underline{\mathtt{new}}\, a' (P\{a' \leftrightarrow a\})] \stackrel{t'}{\Rightarrow} [\underline{\mathtt{new}}\, a' (Q_1\{a' \leftrightarrow a\})]$ with $a' \notin \mathtt{bn}(t')$ or (II) $[\underline{\mathtt{new}}\, a' (P\{a' \leftrightarrow a\})] \stackrel{t'}{\Rightarrow} [Q_1\{a' \leftrightarrow a\}]$ with $a' \in \mathtt{bn}(t')$ holds.

Case $a' \in \mathtt{bn}(t')$**:** Thus, $[\underline{\mathtt{new}}\, a' (P\{a' \leftrightarrow a\})] \stackrel{t'}{\Rightarrow} [Q_1\{a' \leftrightarrow a\}]$. Since the `bind` function just binds the first occurrence of the given name in a trace and leaves the rest of the trace unaltered, $t = t' \frown \langle \alpha \rangle$ with $\langle \alpha \rangle =_{\text{def}} \langle \alpha' \rangle \{a' \leftrightarrow a\}$ and $a' \notin \mathtt{bn}(\alpha)$ holds. Since $[Q_1] \stackrel{\langle \alpha' \rangle}{\Longrightarrow} [Q]$ holds, we know with Lemma 3.3.2 that $[Q_1\{a' \leftrightarrow a\}] \stackrel{\langle \alpha' \rangle \{a' \leftrightarrow a\}}{\Longrightarrow} [Q\{a' \leftrightarrow a\}]$ holds and so $[\underline{\mathtt{new}}\, a' (P\{a' \leftrightarrow a\})] \stackrel{t}{\Rightarrow} [Q\{a' \leftrightarrow a\}]$. Since, we have seen $a' \in \mathtt{bn}(t')$ and $a' \notin \mathtt{bn}(\alpha)$, we know $a' \in \mathtt{bn}(t)$.

Case $a' \notin \mathtt{bn}(t')$**:** Hence, $[\underline{\mathtt{new}}\, a' (P\{a' \leftrightarrow a\})] \stackrel{t'}{\Rightarrow} [\underline{\mathtt{new}}\, a' (Q_1\{a' \leftrightarrow a\})]$. Furthermore, $t' = s'\{a' \leftrightarrow a\}$, since $a' \notin \mathtt{bn}(t')$. Since $[Q_1] \stackrel{\alpha'}{\rightarrow} [Q_2]$ we know with Lemma 2.2.4 that $[Q_1\{a' \leftrightarrow a\}] \stackrel{\alpha}{\rightarrow} [Q_2\{a' \leftrightarrow a\}]$ holds and analogously Lemma 3.3.1 yields $[Q_2\{a' \leftrightarrow a\}] \Rightarrow [Q\{a' \leftrightarrow a\}]$.

If $a' \notin \mathtt{n}(\alpha)$ the $E-RES$ rule of Figure 2.3 yields $[\underline{\mathtt{new}}\, a' (Q_1\{a' \leftrightarrow a\})] \stackrel{\alpha}{\rightarrow} [\underline{\mathtt{new}}\, a' (Q_2\{a' \leftrightarrow a\})]$. Similarly, we know with multiple application of the $E-RES$ rule that $[\underline{\mathtt{new}}\, a' (Q_2\{a' \leftrightarrow a\})] \Rightarrow [\underline{\mathtt{new}}\, a' (Q\{a' \leftrightarrow a\})]$ and since $a' \notin \mathtt{n}(\alpha)$ we know $\mathtt{bind}(a', \alpha) = \alpha$ and so $[\underline{\mathtt{new}}\, a' (P\{a' \leftrightarrow a\})] \stackrel{t}{\Rightarrow} [\underline{\mathtt{new}}\, a' (Q\{a' \leftrightarrow a\})]$ with $a' \notin \mathtt{bn}(t)$.

If $a' \in \mathtt{n}(\alpha)$, we know that $a' \notin \mathtt{sub}(\alpha)$, because otherwise a' must have been bound in t', since $a' \notin \mathtt{n}(t')$, $t = t' \frown \mathtt{bind}(a', \langle \alpha \rangle)$ and $t \in \mathtt{res}(a, a', \mathcal{T}([P]))$ and so $\forall i \in \mathbb{N} : a' \in \mathtt{sub}(t_i) \Rightarrow \exists j \in \mathbb{N} : a' \in \mathtt{bn}(t_j) \wedge j < i$ holds. Thus, $a' \in \mathtt{obj}(\alpha)$ and as we already know $a' \notin \mathtt{bn}(\alpha)$. Hence, there is a name $x \in \mathcal{N} \setminus \{a'\}$ with $\alpha = \overline{x}\langle a' \rangle$ or $\alpha = x\, a'$. Since we know there is no input action α' within t with $a' \in \mathtt{obj}(\alpha')$ from the side condition of Lemma 4.2.9, $\alpha = x\, a'$ is not possible. Thus, $\alpha = \overline{x}\langle a' \rangle$. Then we know with the $E - OPEN$ rule from Figure 2.3 that $[\underline{\mathtt{new}}\, a' (Q_1\{a' \leftrightarrow a\})] \stackrel{\overline{x}(a')}{\longrightarrow} [Q_2\{a' \leftrightarrow a\}]$, since

$[Q_1\{a' \leftrightarrow a\}] \xrightarrow{\overline{x}\langle a'\rangle} [Q_2\{a' \leftrightarrow a\}]$ holds. Since we already showed that $[\underline{\texttt{new}}\, a'\,(P\{a' \leftrightarrow a\})] \overset{t'}{\Rightarrow} [\underline{\texttt{new}}\, a'\,(Q_1\{a' \leftrightarrow a\})]$ holds and furthermore that $[Q_2\{a' \leftrightarrow a\}] \Rightarrow [Q\{a' \leftrightarrow a\}]$ holds and we know in this case $\texttt{bind}(a', \alpha) = \overline{x}(a')$ holds, we know $[\underline{\texttt{new}}\, a'\,(P\{a' \leftrightarrow a\})] \overset{t}{\Rightarrow} [Q\{a' \leftrightarrow a\}]$ with $a' \in \texttt{bn}(t)$.

Thus, all cases of Lemma 4.2.9 are proved. ■

We conjecture that the restriction in Lemma 4.2.9 to the traces t, which do not have an input action α with $a' \in \texttt{obj}(\alpha)$ if $a' \notin \texttt{bn}(t)$, is no restriction for the set inclusion of the union of the restricted sets and the set of traces of the corresponding restricted process. Furthermore, we assume that also the other inclusion holds, since there is only a similar case in which we have not been able to prove this inclusion yet.

Conjecture 4.2.1 (Compositionality of restriction) *Given $P \in \mathcal{P}^{\pi}_{\texttt{recf}}$ and $a \in \mathcal{N}$, then*

$$\mathcal{T}([\underline{\texttt{new}}\, a\ P]) = \bigcup_{a' \in \mathcal{N} \setminus \texttt{fn}(\underline{\texttt{new}}\, a\ P)} \texttt{res}(a, a', \mathcal{T}([P]))$$

holds. ○

To give an idea of a possible proof of the $\supseteq$-direction, we take $P \in \mathcal{P}^{\pi}_{\texttt{recf}}$, $a \in \mathcal{N}$ and $t \in \bigcup_{a' \in \mathcal{N} \setminus \texttt{fn}(\underline{\texttt{new}}\, a\ P)} \texttt{res}(a, a', \mathcal{T}([P]))$. Thus, there is a name $a' \in \mathcal{N} \setminus \texttt{fn}(\underline{\texttt{new}}\, a\ P)$ such that $t \in \texttt{res}(a, a', \mathcal{T}([P]))$ and so there is a trace $s \in \mathcal{T}([P])$ such that $t = \texttt{bind}(a', s\{a' \leftrightarrow a\})$. So, if there is no input action $\alpha \in t$ with $a' \in \texttt{obj}(\alpha)$ in the case that $a' \notin \texttt{bn}(t)$, we get with Lemma 4.2.9 that $t \in \mathcal{T}([\underline{\texttt{new}}\, a'\,(P\{a' \leftrightarrow a\})])$. Since $a' \notin \texttt{fn}(\underline{\texttt{new}}\, a\ P)$ and so $a' \notin \texttt{fn}(P)$, we know $\underline{\texttt{new}}\, a'\,(P\{a' \leftrightarrow a\}) = \underline{\texttt{new}}\, a'\,(P\{^{a'}/_{a}\})$ and by applying α-conversion we know $[\underline{\texttt{new}}\, a'\,(P\{^{a'}/_{a}\})] = [\underline{\texttt{new}}\, a\ P]$ and so $t \in \mathcal{T}([\underline{\texttt{new}}\, a\ P])$. If there is an input action $\alpha \in t$ with $a' \in \texttt{obj}(\alpha)$ and $a' \notin \texttt{bn}(t)$, we assume that we can chose another name $a'' \notin \texttt{fn}(\underline{\texttt{new}}\, a\ P)$ such that for t all premises of Lemma 4.2.9 are fulfilled.

To evaluate the assumption of the possibility to chose another name with which all the premises are fulfilled, we consider, for example, $P =_{\text{def}} \underline{\texttt{new}}\, a'\ x(y).\overline{b}\langle a'\rangle$. Thus, $\left[\underline{\texttt{new}}\, a'\ x(y).\overline{b}\langle a'\rangle\right] \xrightarrow{x\, a'} \left[\underline{\texttt{new}}\, a'\ \overline{b}\langle a'\rangle\right]$, since for example $\underline{\texttt{new}}\, a''\ x(y).\overline{b}\langle a''\rangle \in [P]$ and $\underline{\texttt{new}}\, a''\ \overline{b}\langle a''\rangle \in \left[\underline{\texttt{new}}\, a'\ \overline{b}\langle a'\rangle\right]$. So it seems meaningful that the restriction mention above only take effect in the cases, where we did not chose the corresponding restricted name to the trace. So we can chose another one without harming anything of the properties.

The same problem arises for the other set inclusion. We can prove a similar lemma to the converse of Lemma 4.2.9 as far as we stuck in the case that the restricted name occurs as object of an input action. Furthermore, we tried to develop a term of trace association to restricted processes. That is, we can associate the name a of a restriction $\underline{\texttt{new}}\, a\ P$ to a trace t if $\underline{\texttt{new}}\, a\ P$ is the process of the equivalence class which is used to create the trace t. This is especially interesting if we consider, for example, $P =_{\text{def}} \underline{\texttt{new}}\, a \left(x(y).\underline{\texttt{new}}\, a' \left(\overline{b}\langle a'\rangle.\overline{b}\langle a\rangle\right)\right)$. If we interchange the names a and a' we know $[P] \xRightarrow{\langle x\, y, \overline{b}(a), \overline{b}(a')\rangle} [\mathbf{0}]$. But this approach has also not solved the problem yet.

Applications

As an application of the compositionality of some operators, we can show that the prefix and the sum operator are distributive.

Lemma 4.2.10 (Distributivity of prefix and sum operator) *For an arbitrary prefix π and sums $M_1, M_2 \in \mathcal{P}_M^\pi$ the trace equality*

$$\mathcal{T}([\pi.M_1 + \pi.M_2]) = \mathcal{T}([\pi.\,(M_1 + M_2)])$$

holds. ♢

Proof: Let π be a prefix and $M_1, M_2 \in \mathcal{P}_M^\pi$ sums. If we consider $\pi = \tau$, then we know

$$\begin{aligned}
\mathcal{T}([\tau.M_1 + \tau.M_2]) &\overset{\{\text{Lem. } 4.2.5\}}{=} \mathcal{T}([\tau.M_1]) \cup \mathcal{T}([\tau.M_2]) \\
&\overset{\{\text{Lem. } 4.2.5\}}{=} \mathcal{T}([M_1]) \cup \mathcal{T}([M_2]) \\
&\overset{\{\text{Lem. } 4.2.5\}}{=} \mathcal{T}([M_1 + M_2]) \\
&\overset{\{\text{Lem. } 4.2.5\}}{=} \mathcal{T}([\tau.\,(M_1 + M_2)])
\end{aligned}$$

holds. Otherwise, if $\pi = \overline{a}\langle x\rangle$, the equations

$$\begin{aligned}
\mathcal{T}([\overline{a}\langle x\rangle.M_1 + \overline{a}\langle x\rangle.M_2]) &\overset{\{\text{Lem. } 4.2.5\}}{=} \mathcal{T}([\overline{a}\langle x\rangle.M_1]) \cup \mathcal{T}([\overline{a}\langle x\rangle.M_2]) \\
&\overset{\{\text{Lem. } 4.2.5\}}{=} \{\langle\rangle\} \cup \{\overline{a}\langle x\rangle\} \frown \mathcal{T}([M_1]) \cup \{\overline{a}\langle x\rangle\} \frown \mathcal{T}([M_2]) \\
&\overset{\{\text{Lem. } 2.1.1\}}{=} \{\langle\rangle\} \cup \{\overline{a}\langle x\rangle\} \frown (\mathcal{T}([M_1]) \cup \mathcal{T}([M_2])) \\
&\overset{\{\text{Lem. } 4.2.5\}}{=} \{\langle\rangle\} \cup \{\overline{a}\langle x\rangle\} \frown \mathcal{T}([M_1 + M_2])
\end{aligned}$$

$$\overset{\{\text{Lem. } 4.2.5\}}{=} \mathcal{T}([\overline{a}\langle x\rangle.\,(M_1 + M_2)])$$

hold. Finally, if $\pi = a(x)$, then

$$\begin{aligned}
\mathcal{T}([a(x).M_1 + a(x).M_2]) &\overset{\{\text{Lem. } 4.2.5\}}{=} \mathcal{T}([a(x).M_1]) \cup \mathcal{T}([a(x).M_2]) \\
&\overset{\{\text{Lem. } 4.2.6\}}{=} \{\langle\rangle\} \cup \{\langle a\, y\rangle^\frown s \mid s \in \mathcal{T}([M_1\,\{^y/_x\}]), y \in \mathcal{N}\} \\
&\qquad \cup \{\langle a\, y\rangle^\frown s \mid s \in \mathcal{T}([M_2\,\{^y/_x\}]), y \in \mathcal{N}\} \\
&\overset{\{\text{set union}\}}{=} \{\langle\rangle\} \cup \Big\{\langle a\, y\rangle^\frown s \mid s \in \Big(\mathcal{T}([M_1\,\{^y/_x\}]) \\
&\qquad \cup \mathcal{T}([M_2\,\{^y/_x\}])\Big), y \in \mathcal{N}\Big\} \\
&\overset{\{\text{Lem. } 4.2.5\}}{=} \{\langle\rangle\} \cup \Big\{\langle a\, y\rangle^\frown s \mid s \in \mathcal{T}([M_1\,\{^y/_x\} + M_2\,\{^y/_x\}]), \\
&\qquad y \in \mathcal{N}\Big\} \\
&\overset{\{\text{Def. } 2.2.5\}}{=} \{\langle\rangle\} \cup \Big\{\langle a\, y\rangle^\frown s \mid s \in \mathcal{T}([(M_1 + M_2)\,\{^y/_x\}]), \\
&\qquad y \in \mathcal{N}\Big\} \\
&\overset{\{\text{Lem. } 4.2.6\}}{=} \mathcal{T}([a(x).\,(M_1 + M_2)])
\end{aligned}$$

holds. Thus, for every possible prefix Lemma 4.2.10 holds. ■

Another application is the preservation of the trace inclusion for the output prefix and the choice composition.

Lemma 4.2.11 (Trace inclusion for choice composition and output prefix) *Given processes $P, Q \in \mathcal{P}^\pi$, $M_1, M_2, M_3, M_4 \in \mathcal{P}^\pi_M$, and names $a, x \in \mathcal{N}$, then*

(1) $\mathcal{T}([P]) \subseteq \mathcal{T}([Q])$ is equivalent to $\mathcal{T}([\overline{a}\langle x\rangle.P]) \subseteq \mathcal{T}([\overline{a}\langle x\rangle.Q])$,

(2) $\mathcal{T}([M_1]) \subseteq \mathcal{T}([M_2])$ and $\mathcal{T}([M_3]) \subseteq \mathcal{T}([M_4])$ implies

$$\mathcal{T}([M_1 + M_3]) \subseteq \mathcal{T}([M_2 + M_4])$$

holds. ♢

Proof: Let $P, Q \in \mathcal{P}^\pi$ and $a, x \in \mathcal{N}$ with $\mathcal{T}([P]) \subseteq \mathcal{T}([Q])$ and let $s \in \mathcal{T}([\overline{a}\langle x\rangle.P]) = \{\langle\rangle\} \cup \{\langle\overline{a}\langle x\rangle\rangle\}^\frown \mathcal{T}([P])$ with Lemma 4.2.5. If $s = \langle\rangle$, then we know with Lemma 4.2.1 that $s \in \mathcal{T}([\overline{a}\langle x\rangle.Q])$. Otherwise, if $s \in \{\langle\overline{a}\langle x\rangle\rangle\}^\frown \mathcal{T}([P])$ we know there is a trace $s' \in \mathcal{T}([P])$ such that $s = \langle\overline{a}\langle x\rangle\rangle^\frown s'$. Hence, $s' \in \mathcal{T}([Q])$ and so $s \in$

$\{\langle\overline{a}\langle x\rangle\rangle\}^\frown \mathcal{T}([Q]) = \mathcal{T}([\overline{a}\langle x\rangle.Q])$. The other direction can be proved by using the same lemmas.

Let $M_1, M_2, M_3, M_4 \in \mathcal{P}^\pi_M$ with $\mathcal{T}([M_1]) \subseteq \mathcal{T}([M_2])$ and $\mathcal{T}([M_3]) \subseteq \mathcal{T}([M_4])$. Hence, Lemma 4.2.5 yields that $\mathcal{T}([M_1 + M_3]) = \mathcal{T}([M_1]) \cup \mathcal{T}([M_3]) \subseteq \mathcal{T}([M_2]) \cup \mathcal{T}([M_4]) = \mathcal{T}([M_2 + M_4])$ holds. ■

Thus, a few applications of the compositionality of some operators are given. We now compare our semantics to simulations and bisimulations.

4.2.3 Simulation and bisimulation

For a placement of our developed trace semantics in the existent context, we investigate the connection of the trace semantics and the notion of simulation and bisimulation. Thus, we gain that a simulation (weak or strong) implies trace inclusion and a bisimulation (weak or strong) implies trace equality. Whereas, on the one hand, the inverse statements for bisimulation and strong simulation does not hold, we conjecture that on the other hand weak simulation is even equivalent to trace inclusion.

First we show that for a pair of a weak simulation and a trace starting in the left process of the pair, there is also a process reached by the right process of the pair such that the reached processes are elements of the simulation.

Lemma 4.2.12 (Weak simulation and big-steps semantics) *Given processes $P, P', Q \in \mathcal{P}^\pi$ and $t \in \texttt{Traces}$, then there exists a weak simulation $\mathcal{S} \subseteq \mathcal{P}^\pi_\alpha \times \mathcal{P}^\pi_\alpha$ such that*

$$([P], [Q]) \in \mathcal{S} \wedge [P] \overset{t}{\Rightarrow} [P'] \text{ implies } \exists Q' \in \mathcal{P}^\pi : [Q] \overset{t}{\Rightarrow} [Q'] \wedge ([P'], [Q']) \in \mathcal{S}$$

holds. ♢

Proof: Let $P, P', Q \in \mathcal{P}^\pi$, $t \in \texttt{Traces}$ and $\mathcal{S}$ a weak simulation with $([P], [Q]) \in \mathcal{S}$ and $[P] \overset{t}{\Rightarrow} [P']$. We proceed by induction over the length $n \in \mathbb{N}$ of trace t.

Base case $n = 0$**:** Hence, $t = \langle\rangle$ and so there are processes $P_1, \ldots, P_m \in \mathcal{P}^\pi$ for $m \in \mathbb{N}$ such that $[P] \xrightarrow{\tau} [P_1] \wedge \cdots \wedge [P_m] \xrightarrow{\tau} [P']$. With the definition of the weak simulation (Definition 2.2.12) we know there is a process $Q_1 \in \mathcal{P}^\pi$ with $[Q] \Rightarrow [Q_1]$ and $([P_1], [Q_1]) \in \mathcal{S}$. This argument applied to every further τ step yields that there is a process $Q' \in \mathcal{P}^\pi$ with $[Q_m] \Rightarrow [Q']$ and $([P'], [Q']) \in \mathcal{S}$.

Thus, with the transitivity of the big-step semantics we know $[Q] \Rightarrow [Q']$ and so found the claimed Q'.

Induction hypothesis: For a given $n \in \mathbb{N}$ and for all processes $P, P', Q \in \mathcal{P}^\pi$, $t \in \mathtt{Traces}$ with $\#(t) \leq n$ and weak simulations $\mathcal{S}$ with $([P], [Q]) \in \mathcal{S}$ and $[P] \overset{t}{\Rightarrow} [P']$, there is a process $Q' \in \mathcal{P}^\pi$ with $[Q] \overset{t}{\Rightarrow} [Q']$ and $([P'], [Q']) \in \mathcal{S}$.

Induction step $n \mapsto n+1$**:** Thus, $t = t' \frown \langle \alpha \rangle$ with $t' \in \mathtt{Traces}$ and $\alpha \in \mathtt{Act} \setminus \{\tau\}$. Hence, there are processes $P_1, P_2, P_3 \in \mathcal{P}^\pi$ with $[P] \overset{t'}{\Rightarrow} [P_1] \Rightarrow [P_2] \overset{\alpha}{\rightarrow} [P_3] \Rightarrow [P']$. The induction hypothesis yields that there is a process $Q_1 \in \mathcal{P}^\pi$ with $[Q] \overset{t'}{\Rightarrow} [Q_1]$ and $([P_1], [Q_1]) \in \mathcal{S}$. Furthermore, since $([P_1], [Q_1]) \in \mathcal{S}$, we know additionally from the induction hypothesis that there is a process $Q_2 \in \mathcal{P}^\pi$ with $[Q_1] \Rightarrow [Q_2]$ and $([P_2], [Q_2]) \in \mathcal{S}$. So, Definition 2.2.12 yields that there is a process $Q_3 \in \mathcal{P}^\pi$ with $[Q_2] \overset{\langle \alpha \rangle}{\Longrightarrow} [Q_3]$ and $([P_3], [Q_3]) \in \mathcal{S}$. Hence, with another application of the induction hypothesis we know that there is a process $Q' \in \mathcal{P}^\pi$ with $[Q_3] \Rightarrow [Q']$ and $([P'], [Q']) \in \mathcal{S}$ and thus, $[Q] \overset{t}{\Rightarrow} [Q']$

Thus, Lemma 4.2.12 is proved by induction over the length of the trace. ■

With the help of the previous lemma we can show that for pairs of a weak simulation the trace inclusion holds.

Lemma 4.2.13 (Weak simulation and trace inclusion) *Let $P, Q \in \mathcal{P}^\pi$ processes, then there exists a weak simulation $\mathcal{S} \subseteq \mathcal{P}^\pi_\alpha \times \mathcal{P}^\pi_\alpha$ such that*

$$([P], [Q]) \in \mathcal{S} \textit{ implies } \mathcal{T}([P]) \subseteq \mathcal{T}([Q])$$

holds. ♢

Proof: Given a weak simulation $\mathcal{S} \subseteq \mathcal{P}^\pi_\alpha \times \mathcal{P}^\pi_\alpha$ and let $P, Q \in \mathcal{P}^\pi$ be processes with $([P], [Q]) \in \mathcal{S}$ and $t \in \mathcal{T}([P])$. Thus, there is a process $P' \in \mathcal{P}^\pi$ with $[P] \overset{t}{\Rightarrow} [P']$ and Lemma 4.2.12 yields that there is a process $Q' \in \mathcal{P}^\pi$ with $[Q] \overset{t}{\Rightarrow} [Q']$. Hence, $t \in \mathcal{T}([Q])$. ■

We conjecture that also the converse of Lemma 4.2.13 holds, but a proof is not yet fully developed. For an idea of a proof we define a set of process tuples, which are all good candidates to stay in a simulation if a trace inclusion is given.

Definition 4.2.3 (Weak processes set) For $P, Q \in \mathcal{P}^\pi$ and $i \in \mathbb{N}$ we inductively define $\mathcal{S}_i^{P,Q} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ with

$$\begin{aligned}
\mathcal{S}_0^{P,Q} &=_{\text{def}} \{([P], [Q])\} \\
\mathcal{S}_{n+1}^{P,Q} &=_{\text{def}} \Big\{([P''], [Q'']) \mid \exists([P'], [Q']) \in \mathcal{S}_n^{P,Q} : [P'] \xrightarrow{\tau} [P''] \wedge [Q'] \Rightarrow [Q''] \\
&\qquad \vee \exists \alpha \in \texttt{Act} \setminus \{\tau\} : [P'] \xrightarrow{\alpha} [P''] \wedge [Q'] \overset{\langle\alpha\rangle}{\Longrightarrow} [Q'']\Big\},
\end{aligned}$$

and call it the *weak processes set*. ∗

For this set we can show that for every tuple within, there is a trace from the processes the set is conducted from to the elements of the tuple.

Lemma 4.2.14 (Weak simulation set and traces) *Let $P, Q \in \mathcal{P}^\pi$ and for all $n \in \mathbb{N}$ let $\mathcal{S}_n^{P,Q}$ defined as in Definition 4.2.3. Then*

$$\forall n \in \mathbb{N} : ([P'], [Q']) \in \mathcal{S}_n^{P,Q} \text{ implies } \exists t \in \texttt{Traces} : [P] \overset{t}{\Rightarrow} [P'] \wedge [Q] \overset{t}{\Rightarrow} [Q']$$

holds. ♢

Proof: Let $P, Q \in \mathcal{P}^\pi$ and for all $n \in \mathbb{N}$ let $\mathcal{S}_n^{P,Q}$ be defined as in Definition 4.2.3. Then we proceed by induction over the index n of the weak simulation sets.

Base case $n = 0$**:** Since the only element in $\mathcal{S}_0^{P,Q}$ is $([P], [Q])$, we chose $t =_{\text{def}} \langle\rangle$. Hence $[P] \overset{t}{\Rightarrow} [P] \wedge [Q] \overset{t}{\Rightarrow} [Q]$.

Induction hypothesis: For a given $n \in \mathbb{N}$ and for all $([P'], [Q']) \in \mathcal{S}_n^{P,Q}$ there is a trace $t \in \texttt{Traces}$ with $[P] \overset{t}{\Rightarrow} [P']$ and $[Q] \overset{t}{\Rightarrow} [Q']$.

Induction step $n \mapsto n+1$**:** Thus let $([P''], [Q'']) \in \mathcal{S}_{n+1}^{P,Q}$. From the definition of $\mathcal{S}_{n+1}^{P,Q}$ we know there is a tuple $([P'], [Q']) \in \mathcal{S}_n^{P,Q}$ with $[P'] \xrightarrow{\tau} [P''] \wedge [Q'] \Rightarrow [Q'']$ or there is an action $\alpha \in \texttt{Act} \setminus \{\tau\}$ with $[P'] \xrightarrow{\alpha} [P''] \wedge [Q'] \overset{\langle\alpha\rangle}{\Longrightarrow} [Q'']$. The induction hypothesis yields that there is a trace $t \in \texttt{Traces}$ with $[P] \overset{t}{\Rightarrow} [P']$ and $[Q] \overset{t}{\Rightarrow} [Q']$. Hence, either $[P] \overset{t}{\Rightarrow} [P'']$ and $[Q] \overset{t}{\Rightarrow} [Q'']$ or $[P] \overset{t^\frown\langle\alpha\rangle}{\Longrightarrow} [P'']$ and $[Q] \overset{t^\frown\langle\alpha\rangle}{\Longrightarrow} [Q'']$ holds.

Thus, Lemma 4.2.14 is proved by induction over the index of the weak simulation set. ■

We now guess that one can minimize the union of the weak processes set such that the result is a weak simulation if the trace inclusion holds.

In general $\mathcal{T}([P]) \subseteq \mathcal{T}([Q])$ does not imply that $\mathcal{S} =_{\text{def}} \bigcup_{i \in \mathbb{N}} \mathcal{S}_i^{P,Q}$ is a weak simulation. Consider, for example, $P =_{\text{def}} \overline{a}\langle x\rangle.\overline{b}\langle x\rangle$ and $Q =_{\text{def}} \overline{a}\langle x\rangle + \overline{a}\langle x\rangle.\overline{b}\langle x\rangle$. Thus, $\mathcal{T}([P]) = \mathcal{T}([Q])$, $\mathcal{S}_0^{P,Q} = \{([P], [Q])\}$ and $\mathcal{S}_1^{P,Q} = \left\{\left(\left[\overline{b}\langle x\rangle\right], [\mathbf{0}]\right), \left(\left[\overline{b}\langle x\rangle\right], \left[\overline{b}\langle x\rangle\right]\right)\right\}$. Hence, $\left(\left[\overline{b}\langle x\rangle\right], [\mathbf{0}]\right) \in \mathcal{S}$ but $\left[\overline{b}\langle x\rangle\right]$ has a visible transition and $[\mathbf{0}]$ has no transitions. Such tuple has to be filtered to achieve a weak simulation, since

$$\mathcal{S} \setminus \left\{\left(\left[\overline{b}\langle x\rangle\right], [\mathbf{0}]\right)\right\} = \left\{([P], [Q]), \left(\left[\overline{b}\langle x\rangle\right], \left[\overline{b}\langle x\rangle\right]\right), ([\mathbf{0}], [\mathbf{0}])\right\}$$

is a weak simulation.

Since a strong simulation is also a weak one, Lemma 4.2.13 directly yields that the existence of a strong simulation implies the trace inclusion.

Corollary 4.2.2 (Strong simulation and trace inclusion) *Let $P, Q \in \mathcal{P}^\pi$ processes, then there exists a strong simulation $\mathcal{S} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ such that*

$$([P], [Q]) \in \mathcal{S} \text{ implies } \mathcal{T}([P]) \subseteq \mathcal{T}([Q])$$

holds. □

Proof: Let $\mathcal{S} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ be a strong simulation and $P, Q \in \mathcal{P}^\pi$ processes with $([P], [Q]) \in \mathcal{S}$. Since $\mathcal{S}$ is a strong simulation and so a weak one [SW01], we know with Lemma 4.2.13 that $\mathcal{T}([P]) \subseteq \mathcal{T}([Q])$. ■

A very simple example already shows that the converse of Corollary 4.2.2 cannot hold. Consider, for example, $P =_{\text{def}} \tau.\overline{a}\langle x\rangle$ and $Q =_{\text{def}} \overline{a}\langle x\rangle$. Then $\mathcal{T}([P]) = \mathcal{T}([Q])$ but since there is no τ transition starting in $[Q]$, but $[P] \xrightarrow{\tau} [\overline{a}\langle x\rangle]$, we know that no strong simulation $\mathcal{S}$ can exist such that $([P], [Q]) \in \mathcal{S}$.

The definition of a bisimulation directly yields with Lemma 4.2.13 and Corollary 4.2.2 that the existence of a weak respectively strong bisimulation implies trace equality.

Corollary 4.2.3 (Bisimulation and trace inclusion) *Let $P, Q \in \mathcal{P}^\pi$ processes, then there exists a weak or strong bisimulation $\mathcal{S} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ such that*

$$([P], [Q]) \in \mathcal{S} \text{ implies } \mathcal{T}([P]) = \mathcal{T}([Q])$$

holds. □

Proof: Let $\mathcal{S} \subseteq \mathcal{P}_\alpha^\pi \times \mathcal{P}_\alpha^\pi$ be a weak or a strong bisimulation. Hence, we know $\mathcal{S}$ and $\mathcal{S}^{-1}$ are weak respectively strong simulations. Let $([P], [Q]) \in \mathcal{S}$ and so $([Q], [P]) \in \mathcal{S}^{-1}$. Thus, with Lemma 4.2.13 respectively Corollary 4.2.2 we know $\mathcal{T}([P]) \subseteq \mathcal{T}([Q])$ and $\mathcal{T}([P]) \supseteq \mathcal{T}([Q])$ holds and so $\mathcal{T}([P]) = \mathcal{T}([Q])$. ■

The same example which is used to show that the converse of Corollary 4.2.2 cannot hold also yields that the converse of the strong case for Corollary 4.2.3 cannot hold. The converse of the weak case of Corollary 4.2.3 can also not hold. Consider, for example, $P =_{\text{def}} \overline{a}\langle x\rangle \mid a(y)$ and $Q =_{\text{def}} \overline{a}\langle x\rangle.a(z) + a(z).\overline{a}\langle x\rangle$. Then we know $\mathcal{T}([P]) = \texttt{pref}(\{\langle a\, y, \overline{a}\langle x\rangle\rangle \mid y \in \mathcal{N}\} \cup \{\langle \overline{a}\langle x\rangle, a\, y\rangle \mid y \in \mathcal{N}\}) = \mathcal{T}([Q])$. If such a weak bisimulation $\mathcal{S}$ with $([P], [Q]) \in \mathcal{S}$ exists, then we know $([\mathbf{0} \mid \mathbf{0}], [Q]) \in \mathcal{S}$, since $[P] \xrightarrow{\tau} [\mathbf{0} \mid \mathbf{0}]$ and $[Q] \Rightarrow [Q]$. Thus, $([Q], [\mathbf{0} \mid \mathbf{0}]) \in \mathcal{S}^{-1}$. But since $[Q]$ has a transition, for instance, labeled with $\overline{a}\langle x\rangle$ and in $[\mathbf{0} \mid \mathbf{0}]$ is no transition, especially no visible transition, possible, we know there is no possible way for the existence of a bisimulation $\mathcal{S}$ with $([P], [Q]) \in \mathcal{S}$.

4.3 Refinement

The definition of traces leads directly to the *trace refinement*. Hence, the situation that a process has the same external behavior as another one and, in addition to that, can possibly have some more behavior, is intuitively achieved by set inclusion. This definition directly corresponds to the definition of trace refinement in CSP, for example in [Ros98].

Definition 4.3.1 (Trace refinement) Let $P, Q \in \mathcal{P}^\pi$, then Q is a *trace refinement* of P ($P \sqsubseteq_\mathcal{T} Q$) if the inverse set inclusion holds:

$$P \sqsubseteq_\mathcal{T} Q \text{ is equivalent to } \mathcal{T}([Q]) \subseteq \mathcal{T}([P]).$$

We also say for $P \sqsubseteq_\mathcal{T} Q$ that Q *refines* P.

Two processes are called *trace equivalent*, if and only if they have got the same external behavior: $\mathcal{T}([P]) = \mathcal{T}([Q])$. ∗

Thus, Q refines P if and only if Q has less behavior than P. In contrast to the simulations, where the processes are collected to compare their behavior, with the trace sets we do not observe the processes, but only save the labels of the possible transitions.

Since the refinement is just a set inclusion, we know that all the results of Section 4.2 for set inclusion also hold for the refinement.

Lemma 4.3.1 (Refinement properties) *Given two processes $P, Q \in \mathcal{P}^\pi$ and sums $M_1, M_2, M_3, M_4 \in \mathcal{P}^\pi_M$. Then*

(1) For all substitutions $\sigma = \{^a/_b\}$ with $a \notin \mathtt{fn}(P) \cup \mathtt{fn}(Q)$ or $\sigma = \{a \leftrightarrow b\}$ transposition,

$$Q \sqsubseteq_\mathcal{T} P \text{ implies } Q\sigma \sqsubseteq_\mathcal{T} P\sigma$$

holds,

(2) $Q \sqsubseteq_\mathcal{T} P$ is equivalent to $\overline{a}\langle x\rangle.Q \sqsubseteq_\mathcal{T} \overline{a}\langle x\rangle.P$, for all $a, x \in \mathcal{N}$,

(3) $M_2 \sqsubseteq_\mathcal{T} M_1$ and $M_4 \sqsubseteq_\mathcal{T} M_3$ implies $M_2 + M_4 \sqsubseteq_\mathcal{T} M_1 + M_3$,

(4) For $\mathcal{S} \subseteq \mathcal{P}^\pi_\alpha \times \mathcal{P}^\pi_\alpha$ weak or strong simulation

$$([P], [Q]) \in \mathcal{S} \text{ implies } Q \sqsubseteq_\mathcal{T} P$$

holds

holds. ♢

Proof: Corollary 4.2.1, Lemma 4.2.11, Lemma 4.2.13, Corollary 4.2.2 directly yield the corresponding properties. ■

In the next section we present some examples of applications of the trace refinement.

4.4 Example

In this section we briefly state the usage of the trace refinement by reconsidering the buffer of Section 2.2.4. Furthermore, we introduce a rudimentary idea of a first implementation of the calculation of the trace sets. For this approach we almost directly convert the compositional definitions of Section 4.2.2 to pseudocode.

```
INPUT: P ∈ P^π_recf , 𝔑 ⊂ 𝒩 finite
OUTPUT: 𝒯_𝔑([P])

P' := esf(P) // to extended standard form
switch P' do
  case P' = 0: return {⟨⟩}
  case P' = τ.P'': return calc(P'', 𝔑)
  case P' = ā⟨x⟩.P'': T := calc(P'', 𝔑)
                      return {⟨⟩} ∪ {ā⟨x⟩}⌢T
  case P' = a(x).P'': return calc_inp(P', 𝔑) // see Figure 4.4
  case P' = M1 + M2: T1 := calc(M1, 𝔑)
                     T2 := calc(M2, 𝔑)
                     return M1 ∪ M2
  case P' = P1 | P2: return calc_par(P', 𝔑) // see Figure 4.5
  case P' = new a P'': return calc_res(P', 𝔑) // see Figure 4.6
od
```

Figure 4.3: Compositional computation of traces `calc`.

```
INPUT: P ∈ P^π_recf with P = a(x).P' , 𝔑 ⊂ 𝒩 finite
OUTPUT: 𝒯_𝔑([P])

Tr := calc(P', 𝔑) // see Figure 4.3

T := ∅
foreach n ∈ 𝔑 ∪ fn(P) do
  if n ∉ fn(P') then
    T' := Tr{n ↔ x} // cf. Corollary 4.2.1
  else if n = x then
    T' := Tr
  else
    P'' := P'{n/x}
    T' := calc(P'', 𝔑) // see Figure 4.3
  fi
  T := T ∪ {⟨⟩} ∪ {⟨a n⟩}⌢T'
od
return T
```

Figure 4.4: Compositional computation of traces $\texttt{calc}_{\texttt{inp}}$.

Thus, in Figure 4.3 we list the main calculation algorithm, which decides by the process' structure, which sub-algorithm should be used. All algorithms take a recursion-free process and an finite subset of names as input parameters. The limitation to recursion-free processes depends on the circumstances that we neither have any fix-point algorithm up to now nor showed that one exists. The second input parameter is invented for the calculation of the traces of an input process and denotes the names which can be send from the environment to the system. First of all the process is transformed in extended standard form for a possible application of the algorithm for the parallel composition. The algorithm for an inaction, a choice, and an output process are already given in this listing. They are all the direct translation of the definitions of Section 4.2.2.

In Figure 4.4 the algorithm for calculating the traces of an input process is given. In this algorithm we can understand the need of the restriction to a finite set of names. Since all names which can possibly be send from the environment to the process are collected in $\mathfrak{N}$, we know that only those names and the names which occur free in the process under consideration can possibly establish new communication. Thus, we only have to calculate traces for the processes, where the subject of the input prefix is substituted by such names. To optimize the number of calculations, we take advantage of Corollary 4.2.1.

The algorithm for calculating the traces of the parallel composition is listed in Figure 4.5. Mainly, the algorithm uses Definition 4.2.1 to calculated the sets of the inductive parallel composition of the traces of P_1 and P_2. This is also a straightforward translation of this definition.

Figure 4.6 show the computation of the traces of a restriction. This computation directly yields from Conjecture 4.2.1.

Utilizing the approach, we can investigate the buffer defined in Section 2.2.4. Since we do not have any algorithm for computing processes with recursive calls, we have to unfold the examples. For a clearer arrangement we do not write the processes as a whole, but naming parts of the process separately. Those naming do not have to be understood as a recursive call without parameter. First of all we reconsider the one cell buffer of [Mil99] unfolded once:

$$ONE\_CELL =_{\text{def}} i(v_1).\overline{o}\langle v_1\rangle.i(v_2).\overline{o}\langle v_2\rangle.$$

INPUT: $P \in \mathcal{P}^{\pi}_{\texttt{recf}}$ with $P = P_1 \mid P_2$, $\mathfrak{N} \subset \mathcal{N}$ finite
OUTPUT: $\mathcal{T}_{\mathfrak{N}}([P])$

$T_1' := \texttt{calc}(P_1, \mathfrak{N})$ // see Figure 4.3
$T_2' := \texttt{calc}(P_2, \mathfrak{N})$ // see Figure 4.3
$T_1' := \texttt{idx}(T_1')$ // adding indices I_P to T_1'
$T_2' := \texttt{idx}(T_2')$ // adding indices I_Q to T_2'

$T_0 := \{(((i_P, 0), (i_Q, 0)), \langle\rangle) \mid i_P \in I_P, i_Q \in I_Q\}$
$i := 1$
$finish := false$
while $\neg finish$ do
  $finish := true$
  foreach $((i_P, j_P'), (i_Q, j_Q'), t') \in T_{i-1}$ do
    Let $(i_P, s) \in T_1'$ and $(i_Q, t) \in T_2'$.
    if $j_P' + 1 \leq \#(s)$ then
      add $((i_P, j_P' + 1), (i_Q, j_Q'), t' \frown \langle s_{j_P'+1} \rangle)$ to T_i
      $finish := false$
    fi
    if $j_Q' + 1 \leq \#(t)$ then
      add $((i_P, j_P'), (i_Q, j_Q' + 1), t' \frown \langle t_{j_Q'+1} \rangle)$ to T_i
      $finish := false$
    fi
    if $j_P' + 1 \leq \#(s) \wedge j_Q' + 1 \leq \#(t) \wedge s_{j_P'+1} = \overline{t_{j_Q'+1}}$ then
      add $((i_P, j_P' + 1), (i_Q, j_Q' + 1), t')$ to T_i
      $finish := false$
    fi
  od
  $i := i + 1$
od
return $\{s \in \texttt{Traces} \mid \exists (\cdot, s) \in \bigcup_{n \in \mathbb{N}} \mathcal{T}_n([P \mid Q])\}$

Figure 4.5: Compositional computation of traces $\texttt{calc}_{\texttt{par}}$.

With attention to Figure 2.4 or by using the algorithm, we directly see that

$$\mathcal{T}([ONE\_CELL]) = \texttt{pref}(\{\langle i\, v_1, \overline{o}\langle v_1\rangle, i\, v_2, \overline{o}\langle v_2\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}).$$

Furthermore, we can omit the recursive calls of the second buffer and reach

$$TWO\_CELL =_{\text{def}} i(v_1).\overline{o}\langle v_1\rangle.i(v_2).\overline{o}\langle v_2\rangle + i(v_1').i(v_2').\overline{o}\langle v_1'\rangle.\overline{o}\langle v_2'\rangle.$$

INPUT: $P \in \mathcal{P}^{\pi}_{\texttt{recf}}$ with $P = \underline{\texttt{new}}\, a\, P'$, $\mathfrak{N} \subset \mathcal{N}$ finite
OUTPUT: $\mathcal{T}_{\mathfrak{N}}([P])$

$T := \texttt{calc}(P', \mathfrak{N})$ // see Figure 4.3
$T' := \emptyset$
foreach $a' \in \mathcal{N} \setminus \texttt{fn}(\underline{\texttt{new}}\, a\, P')$ do
  $T'' := T\,\{a' \leftrightarrow a\}$
  $T'' := \texttt{bind}(a', T'')$
  $T'' := \{s \in T'' \mid \forall i \in \mathbb{N} : a' \in \texttt{sub}(s_i) \Rightarrow \exists j \in \mathbb{N} : a' \in \texttt{bn}(s_j) \wedge j < i\}$
  $T' := T' \cup T''$
od
return T'

Figure 4.6: Compositional computation of traces $\texttt{calc}_{\texttt{res}}$.

Thus, the set of traces is again easily to compute:

$$\begin{aligned}\mathcal{T}([TWO\_CELL]) = \;& \texttt{pref}(\{\langle i\, v_1, \overline{o}\langle v_1\rangle, i\, v_2, \overline{o}\langle v_2\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}) \\ \cup\; & \texttt{pref}(\{\langle i\, v_1, i\, v_2, \overline{o}\langle v_1\rangle, \overline{o}\langle v_2\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}).\end{aligned}$$

Thus, since $\mathcal{T}([ONE\_CELL]) \subseteq \mathcal{T}([TWO\_CELL])$ holds, we know that the ONE_CELL buffer refines the TWO_CELL buffer. That is $TWO\_CELL \sqsubseteq_{\mathcal{T}} ONE\_CELL$. As expected this does not hold the other way round.

We can now think of our invented TWO_CELL buffer as an implementation of Milner's $FIFO$ buffer defined in Example 2.2.3. Thus, we would like that our implementation satisfy the specification of a FIFO buffer by Milner. Therefore, we unfold the buffer once:

$$FIFO =_{\text{def}} \underline{\texttt{new}}\, c\, (i(v_1).\overline{c}\langle v_1\rangle.i(v_2).\overline{c}\langle v_2\rangle \mid c(v_1').\overline{o}\langle v_1'\rangle.c(v_2').\overline{o}\langle v_2'\rangle)\,.$$

So we see, that the channel c is private and hence, only the traces where a communicated over c has been established are within the set of traces. All traces may not so obviously be read from the definition of $FIFO$. So we implemented the algorithm presented at the beginning of this section. Thus, we get:

$$\begin{aligned}\mathcal{T}([FIFO]) = \;& \texttt{pref}(\{\langle i\, v_1, \overline{o}\langle v_1\rangle, i\, v_2, \overline{o}\langle v_2\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}) \\ \cup\; & \texttt{pref}(\{\langle i\, v_1, i\, v_2, \overline{o}\langle v_1\rangle, \overline{o}\langle v_2\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}).\end{aligned}$$

Hence, we know that the TWO_CELL and the $FIFO$ buffer are trace equivalent

and therefore for this unfolding the TWO_CELL buffer satisfy its specification. But if we unfold the two cell buffer of Example 2.2.2 once and the FIFO buffer triply so that they are both able to perform four read and write passes, we see that we did not developed a FIFO buffer with the same behavior than Milner's one. So, we unfold our two cell buffer once and so define TC with:

$$\begin{array}{rcl} TC & =_{\text{def}} & i(v_1).i(v_2).\overline{o}\langle v_1\rangle.\overline{o}\langle v_2\rangle.\,(B_1 + B_2) \\ & + & i(v_1').\overline{o}\langle v_1'\rangle.i(v_2').\overline{o}\langle v_2'\rangle.\,(B_3 + B_4) \\ B_1 & =_{\text{def}} & i(v_3).i(v_4).\overline{o}\langle v_3\rangle.\overline{o}\langle v_4\rangle \\ B_2 & =_{\text{def}} & i(v_3').\overline{o}\langle v_3'\rangle.i(v_4').\overline{o}\langle v_4'\rangle \\ B_3 & =_{\text{def}} & i(v_3'').i(v_4'').\overline{o}\langle v_3''\rangle.\overline{o}\langle v_4''\rangle \\ B_4 & =_{\text{def}} & i(v_3''').\overline{o}\langle v_3'''\rangle.i(v_4''').\overline{o}\langle v_4'''\rangle. \end{array}$$

Furthermore, with F_2 we denote the triply unfolded FIFO buffer with:

$$\begin{array}{rcl} F_2 & =_{\text{def}} & \underline{\texttt{new}}\, c\,(i(v_1).\overline{c}\langle v_1\rangle.i(v_2).\overline{c}\langle v_2\rangle.B_1 \mid c(v_1').\overline{o}\langle v_1'\rangle.c(v_2').\overline{o}\langle v_2'\rangle.B_2) \\ B_1 & =_{\text{def}} & i(v_3).\overline{c}\langle v_3\rangle.i(v_4).\overline{c}\langle v_4\rangle \\ B_2 & =_{\text{def}} & c(v_3').\overline{o}\langle v_3'\rangle.c(v_4').\overline{o}\langle v_4'\rangle. \end{array}$$

The traces for the two cell buffer TC are again readable of the definition, since the traces of the choice operator are just the union of the separate parts. Thus, we know

$$\begin{array}{rcl} \mathcal{T}([TC]) & = & \texttt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, \overline{o}\langle v_2\rangle, i\,v_3, i\,v_4, \overline{o}\langle v_3\rangle, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\ & \cup & \texttt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, \overline{o}\langle v_2\rangle, i\,v_3, \overline{o}\langle v_3\rangle, i\,v_4, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\ & \cup & \texttt{pref}(\{\langle i\,v_1, \overline{o}\langle v_1\rangle, i\,v_2, \overline{o}\langle v_2\rangle, i\,v_3, i\,v_4, \overline{o}\langle v_3\rangle, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\ & \cup & \texttt{pref}(\{\langle i\,v_1, \overline{o}\langle v_1\rangle, i\,v_2, \overline{o}\langle v_2\rangle, i\,v_3, \overline{o}\langle v_3\rangle, i\,v_4, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \end{array}$$

holds. For the FIFO buffer F_2 we needed to adapt our rudimentary implementation a bit for this special case. Otherwise the computer needed more than 35GB RAM (including swap), although we swap all actual not needed sets of the inductive parallel composition to the hard drive. Thus, since we know that the input variables never are used as channels we always take the first case in Figure 4.4. That is, we are treating the input variables in any case as not free within the process. Furthermore, we just compute the traces for four names possible as input variable. At least, for the parallel composition we added another condition for adding traces to the sets of the inductive parallel composition. Since we know that after computing the traces of the parallel composition, we will in this case filter out all traces containing the

name c with the restriction algorithm. Thus, we already omitting them during the computation of the parallel composition. This approach already calculates 463761 traces for the inductive parallel composition with index 0 and ends up by 24505 traces. This all leads to the following result of the set of traces for the F_2 FIFO buffer:

$$\begin{aligned}
\mathcal{T}([F_2]) &= \mathtt{pref}(\{\langle i\,v_1, \overline{o}\langle v_1\rangle, i\,v_2, \overline{o}\langle v_2\rangle, i\,v_3, \overline{o}\langle v_3\rangle, i\,v_4, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\
&\cup \mathtt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, i\,v_3, \overline{o}\langle v_2\rangle, i\,v_4, \overline{o}\langle v_3\rangle, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\
&\cup \mathtt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, i\,v_3, \overline{o}\langle v_2\rangle, \overline{o}\langle v_3\rangle, i\,v_4, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\
&\cup \mathtt{pref}(\{\langle i\,v_1, \overline{o}\langle v_1\rangle, i\,v_2, \overline{o}\langle v_2\rangle, i\,v_3, i\,v_4, \overline{o}\langle v_3\rangle, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\
&\cup \mathtt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, \overline{o}\langle v_2\rangle, i\,v_3, i\,v_4, \overline{o}\langle v_3\rangle, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\
&\cup \mathtt{pref}(\{\langle i\,v_1, \overline{o}\langle v_1\rangle, i\,v_2, i\,v_3, \overline{o}\langle v_2\rangle, i\,v_4, \overline{o}\langle v_3\rangle, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\
&\cup \mathtt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, \overline{o}\langle v_2\rangle, i\,v_3, \overline{o}\langle v_3\rangle, i\,v_4, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}) \\
&\cup \mathtt{pref}(\{\langle i\,v_1, \overline{o}\langle v_1\rangle, i\,v_2, i\,v_3, \overline{o}\langle v_2\rangle, \overline{o}\langle v_3\rangle, i\,v_4, \overline{o}\langle v_4\rangle\rangle \mid v_1,\ldots,v_4 \in \mathcal{N}\}).
\end{aligned}$$

Thus, we get that F_2 does satisfy the specification of a FIFO buffer, but since for example $\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, i\,v_3, \overline{o}\langle v_2\rangle, \overline{o}\langle v_3\rangle, i\,v_4, \overline{o}\langle v_4\rangle\rangle \notin \mathcal{T}([TC])$, we know those buffer are not trace equivalent.

For a buffer not satisfying the specification of Milner's FIFO buffer, we consider

$$\begin{aligned}
BUF &=_{\text{def}} i(v_1).\overline{o}\langle v_1\rangle.i(v_2).\overline{o}\langle v_2\rangle \\
&+ i(v_1').i(v_2').\overline{o}\langle v_1'\rangle.\overline{o}\langle v_2'\rangle \\
&+ i(v_1'').i(v_2'').\overline{o}\langle v_2''\rangle.\overline{o}\langle v_1''\rangle.
\end{aligned}$$

Thus, we know

$$\begin{aligned}
\mathcal{T}([BUF]) = &\ \mathtt{pref}(\{\langle i\,v_1, \overline{o}\langle v_1\rangle, i\,v_2, \overline{o}\langle v_2\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}) \\
\cup &\ \mathtt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_1\rangle, \overline{o}\langle v_2\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}) \\
\cup &\ \mathtt{pref}(\{\langle i\,v_1, i\,v_2, \overline{o}\langle v_2\rangle, \overline{o}\langle v_1\rangle\rangle \mid v_1, v_2 \in \mathcal{N}\}).
\end{aligned}$$

holds. Hence, BUF is not satisfying the specification of $FIFO$, since for example $\langle i\,v_1, i\,v_2, \overline{o}\langle v_2\rangle, \overline{o}\langle v_1\rangle\rangle \notin \mathcal{T}([FIFO])$.

5 Conclusion and future work

In this thesis we developed a trace refinement for the π-calculus and investigated its properties and limitations.

The aim of the definition is that the trace semantics preserves some of the advantages of the set-theoretical denotation of the CSP approach in [Ros98]. Unfortunately, we found out that our trace semantics are most likely not compositional in general. On the one hand, we showed that the τ and output prefixes and the choice are compositional in the way the corresponding in CSP are. On the other hand, we explained that, according to the specialty of the π-calculus to transmit channels through channels, the possibly newly generated communications most likely limit the compositionality of the input prefix. In spite of that, we sketched a straightforward algorithmic idea to calculate prefix processes for a finite set of names coming from the environment. This approach takes advantage of the set equality for the application of a transposition within the trace semantics or outside of it. Furthermore, we proved the compositionality of the parallel composition limited to recursion- and restriction-free processes. For an algorithmic approach we invented a new standard form for processes and showed that every process can be transformed to a structurally equal one in this form. Therefore, the limitation to restriction-free processes is not harmful for the algorithmic calculation. Furthermore, we stated an idea for the compositionality of the restriction operator, but could not completely prove it within the scope of this thesis. Finally, we detected that the existence of a weak or strong simulation implies trace refinement and showed that the converse for strong simulations does not hold. Furthermore, we conjecture that the concept of trace refinement should be equivalent to weak simulation.

The two conjectures mentioned above and the restrictions of the compositionality are a good start for future work. Furthermore, it would be promising to distinguish processes by their different behaviour in the degree of nondeterminism and divergence like the failures and failures divergence refinement described in [Ros98] for CSP. Since there are many other definitions of simulation and bisimulation, for example the barbed and full one [SW01], it would be interesting to investigate their

connection to the trace semantics. For example, consider the full bisimulation, which contains processes which are bisimilar regardless of the application of substitutions. This could be related to the lemma, which states that the application of a transposition to the traces of a process results the same as first applying the transposition to the process and then calculating the traces. Considering this lemma it would be of further interest to investigate whether this trace equivalence for the application of a transposition can be extended to substitutions injective on the free names of the process, since the heart of the trace equivalence proof is a lemma in [SW01] with those premises. In the case the compositionality of the parallel composition and the restriction is given, it could be promising to investigate the preservation of the trace inclusion under these operators, like it is done for the output prefix and the choice. Considerably more work will need to be done to determine if we can develop another operational semantics to reduce the number of transitions and thus the number of traces. Since we know that we can replace the bound names of a trace without changing its affiliation to a trace set, it could be possible to change something within the rules which use the restriction operator. Therefore, [FMQ96] could be an interesting approach, since they invented explicit substitutions and state that they can reduce the transition system to a standard structured operational semantics framework. Apart from that, the definition of the compositionality of the parallel composition seems to be a bit complex. Furthermore, the direct implementation led to a huge amount of data. Maybe there is a more convincing way of a non-inductive definition. Generally, a complete or at least less rudimentary implementation than the existing one would be useful to have, especially for the case of an input process and the parallel composition. This work could be inspired by the FDR model checking tool, which has algorithms for checking refinements of CSP processes [Ltd97]. Furthermore, it would be recommended to use a framework like OpenCL[1], since the setting of the algorithm is highly concurrent. Finally, it would be interesting to investigate the usability of a combination of the π-calculus and the formal specification language Z [ASM80, Spi89] to model data in an easier way. Since there is an approach using Object-Z [CDD+89, Smi00] and CSP to model data in CSP (CSP-OZ [Fis97]) and this notion bases on the failure-divergence semantics and takes advantage of the set-theoretical approach and refinement, it would be interesting to determine whether some kind of π-OZ is suitable.

Another view on handling bound names is presented in [CC03], where they define

[1] http://www.khronos.org/opencl/ visited 29th March 2014.

a so-called safe substitution, which changes the bound names of a process to fresh ones while applying the substitution. This approach does not suitably fit our setting, since it complicates the application of substitutions on traces and their concatenations. Otherwise, they take advantage of the transposition of names to preserve some features while applying a substitution. This also turned out to be a good idea for our approach. Furthermore, they define the so-called property sets for having more information about names during the handling of their formula. This approach could be interesting for extending the semantics with some sets, which saves suitable names for a better handling of our denotational semantics. A connection between CCS and CSP is given in [HH10, HH06], where they state that CSP is a retract of CCS. They can derive the denotational semantics of CSP by operationally defined links of the operational semantics of CCS. Since we found this paper late in the elaboration of this thesis, there could be interesting ideas for defining a denotational semantics in a different manner. This paper also led to the consideration of observation equivalence (e.g. weak bisimulation) as testing equivalence [Abr87]. Such equivalences are regarded in [NH84] for CCS and for mobile processes in [BN95]. In [BN95] they gain a fully abstract denotational model of the π-calculus related to a weak testing-based behavioral relation. This approach differs to ours, since they use a so-called testing equivalence view. This requires observers and especially they extended the original π-calculus by a mismatch operator to gain the needed observational power. Since the mismatch operator violates Lemma 2.2.4, which is in some sense a foundation of our theory, we did not want to integrate the mismatch prefix. Apart from that it is interesting that they also only consider a finite variant of the π-calculus. They may also provide some good ideas for future improvement. This statement also fits to [Hen02] where two set-theoretical semantics for the π-calculus are constructed, by extending the syntax of the π-calculus.

Bibliography

[Abr87] S. Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53(2):225–241, 1987.

[AC12] A. Alexandru and G. Ciobanu. Nominal semantics of mobility. *Romanian journal of information science and technology*, 15(3):171–214, 2012.

[ASM80] J. Abrial, S. A. Schuman, and B. Meyer. A specification language. *On the Construction of Programs, R. M. McKeag A. M. Macnaghten*, 1980.

[BN95] M. Boreale and R. De Nicola. Testing equivalence for mobile processes. *Information and Computation*, 120(2):279–303, 1995.

[CC03] L. Caires and L. Cardelli. A spatial logic for concurrency (part i). *Information and Computation*, 186(2):194–235, 2003.

[CDD+89] D. A. Carrington, D. J. Duke, R. Duke, P. King, G. A. Rose, and G. Smith. Object-z: An object-oriented extension to z. In *Proceedings of the IFIP TC/WG6. 1 Second International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols*, pages 281–296. North-Holland Publishing Co., 1989.

[CPT01] C. Canal, E. Pimentel, and J. M. Troya. Compatibility and inheritance in software architectures. *Science of Computer Programming*, 41(2):105–138, 2001.

[EN86] U. Engberg and M. Nielsen. A calculus of communicating systems with label passing. *DAIMI Report Series*, 15(208), 1986.

[Fis97] C. Fischer. Csp-oz: A combination of object-z and csp. *Formal methods for open object-based distributed systems*, 2:423–438, 1997.

[FMQ96] G.-L. Ferrari, U. Montanari, and P. Quaglia. A π-calculus with explicit substitutions. *Theoretical Computer Science*, 168(1):53–103, 1996.

[Hen02] M. Hennessy. A fully abstract denotational semantics for the π-calculus. *Theoretical Computer Science*, 278(1):53–89, 2002.

[HH06] J. He and C. A. R. Hoare. Csp is a retract of ccs. In *Unifying Theories of Programming*, pages 38–62. Springer, 2006.

[HH10] J. He and C. A. R. Hoare. Csp is a retract of ccs. *Theoretical Computer Science*, 411(11):1311–1337, 2010.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the Association for Computing Machinery*, 21(8):666–677, 1978.

[Hoa80] C. A. R. Hoare. A model for communicating sequential process, 1980.

[Hoa85] C. A. R. Hoare. *Communicating sequential processes*, volume 178. Prentice-hall Englewood Cliffs, 1985.

[Hoa06] C. A. R. Hoare. Why ever csp? *Electronic Notes in Theoretical Computer Science*, 162:209–215, 2006.

[Ltd97] Formal Systems (Europe) Ltd. Failures-divergence refinement: Fdr2 manual, 1997.

[Mey09] R. Meyer. *Structural Stationarity in the π-Calculus*. PhD thesis, 2009.

[Mil80] R. Milner. A calculus of communicating systems. *Lecture Notes in Computer Science*, 92, 1980.

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

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts i and ii. *Information and Computation*, 100(1):1–77, 1992.

[NH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34(1):83–133, 1984.

[PS96] M. Pistore and D. Sangiorgi. A partition refinement algorithm for the π-calculus. In *Computer Aided Verification*, pages 38–49. Springer, 1996.

[Qua96] P. Quaglia. *The π-calculus with explicit substitutions*. PhD thesis, 1996.

[Ros98] A. W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998.

[Smi00] G. Smith. The object-z specification language. *Advances in Formal Methods. Kluwer Academic Publishers, Dordrecht*, 2000.

[Spi89] M. Spivey. *The Z Notation: a reference manual*. Prentice Hall, 1989.

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

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 29. März 2014

_____

(Manuel Gieseking)