On Parameterization of Higher-Order Processes

Qiang Yin[a,1], Xian Xu[b,1,*], Huan Long[a,1]

[a] *BASICS, Shanghai Jiaotong University, China*
[b] *East China University of Science and Technology, China*

Abstract

Parameterization provides an effective way to improve on the expressiveness of process-passing. In this paper, we study two kinds of parameterization: name parameterization and process parameterization. Firstly, we show that process parameterization retains the characterization of context bisimulation in terms of the far simpler normal bisimulation, in which universal quantifiers are eliminated. Secondly, we prove that name parameterization is at least as expressive as process parameterization by giving an encoding from the higher-order calculus with process parameterization into the higher-order calculus with name parameterization. These results clarify further the theoretical framework of higher-order processes, and shed light on the relationship between the two kinds of parameterization.

Keywords: Context Bisimulation, Normal Bisimulation, Expressiveness, Parameterization, Higher-order, Process calculi
2010 MSC: 68Q85

1. Introduction

Concurrency theory aims at developing a theory for describing and analyzing the behaviour of computing systems that are composed of sub-systems which may communicate with each other. One would expect such a theory to be useful in the sense of being able to give explanations and solutions to problems in various concurrent applications. In order to achieve this goal, different concurrent models have been proposed for different purposes. Among them the higher-order process model is one of the most widely studied [1, 2, 3]. Compared with the traditional first-order (name-passing) processes [4], higher-order processes can transmit themselves (i.e., integral programs) in communication, and are convenient for modeling, e.g., functional programming that exploits higher-order functions [5, 6]

Parameterization (i.e., abstraction, akin to that in lambda-calculus) [1] is a well-known mechanism frequently used in programming languages, distributed computing, and service-oriented computing (e.g., templates and abstract classes). Intuitively speaking, it involves some kind of mechanism that maps a variable (higher-order or first-order) to an object (process or name). Such mechanism cannot be achieved in the basic process-passing paradigm [7]. In this paper, we focus on purely higher-order processes with parameterization. We use Π to denote the basic (name-passing free) higher-order pi-calculus [1, 6], which has the elementary operators including input prefix ($a(X).P$ in which X is a bound variable), output prefix ($\overline{a}A.P$), parallel composition ($P \mid Q$) and restriction ($(c)P$ in which c is a bound name).

Notation Π^D_n (respectively Π^d_n) stands for the calculus extending Π with process parameterization (respectively name parameterization) of arity n ($n \in \mathbb{N}, n{\neq}0$ and $\mathbb{N}$ is the set of natural numbers). The calculus Π^D (respectively Π^d) is the union of the Π^D_n (respectively Π^d_n). For example, the process P_1 below is a Π process (it means outputting on name a the process just received and continuing as null); P_2 is a Π^D_1 process; P_3 is a Π^d_1 process, where the parameterization operation is denoted by (leftmost) $\langle\cdot\rangle$. In the case of P_2, the X is a parameterized variable that can be instantiated with an application (rightmost $\langle\cdot\rangle$). For instance, $P_2\langle A\rangle$ results in the process $\overline{a}A.0$. We will formally define the calculi in Section 2.

$$P_1 \stackrel{\text{def}}{=} a(X).\overline{a}X.0 \qquad P_2 \stackrel{\text{def}}{=} \langle X\rangle\overline{a}X.0 \qquad P_3 \stackrel{\text{def}}{=} \langle y\rangle y(X).\overline{y}X.0$$

*Corresponding author: xuxian@ecust.edu.cn
[1] The authors are supported by NSFC (61261130589, 61202023, 61173048, 61472239, 61572318), and ANR PACE (12IS02001).

Related work and motivation

The mechanism of process-passing provides an alternative way (in addition to name-passing) to achieve mobility. The distinction between name-passing and process-passing lies in several aspects, among which the behavioral theory is of pivotal importance. As the basis of a behavioral theory of processes, bisimulation theory has been studied since the very early work on concurrent models. A well-studied bisimulation for higher-order processes (and probably the standard one) is the context bisimulation [1]. It was proposed to improve on the previous forms of bisimulation, such as (applicative) higher-order bisimulation [8, 9], by considering the sent process and residual process at the same time. Later more effort was devoted to the study of context bisimulation [3, 10, 11, 12, 13, 14]. Another significant topic relevant to behavioral theory is expressiveness. Recent years have seen increased importance of both positive and negative results in the studies of various models [6]. On the whole, the endeavor deals with the relationship among: (1) classical computational models (e.g., Turing machines or equivalent models [13, 15, 16]); (2) models of reactive system (e.g., first-order process calculi [1, 2, 9, 8, 17] and Petri Nets [18]); (3) variants of higher-order processes [7, 13, 19]. Regarding expressiveness, a direct advantage of modeling (or programming) with higher-order processes is that one may communicate whole terms at once. With this ability it is simple to perform usual tasks such as duplicating a process (e.g., using $a(X).(X \mid X)$ that inputs a process over port a and clones another copy of it), whereas first-order processes do not (immediately) have this kind of mechanism. In this paper we will address both bisimulation theory and expressiveness theory of higher-order processes equipped with parameterization. Below we give some overview of these aspects by focusing on the related issues we shall deal with in this paper. The reader is referred to, e.g., [6, 20] for surveys on related areas.

- *Bisimulation theory.* Context bisimulation, in its original form, is not so convenient in that, for example, it involves universal quantifiers when comparing actions. That is a particularly heavy burden because there are infinitely many contexts. This situation is improved by the so-called normal bisimulation, which coincides with context bisimulation but requires only the checking with some special context [1, 3, 12]. A common methodology for establishing such a characterization in [1, 3, 12] consists of two main ingredients: (1) Showing a factorization theorem using triggers. The factorization theorem provides a mechanism for relocating a (sub)process to a new place, and setting up a pointer to the new place for potential use. Such a pointer is represented by a trigger. (2) Showing the coincidence of context and normal bisimulation with the help of an intermediate bisimulation equivalence called triggered bisimulation, which is defined on a subclass of higher-order processes communicating only triggers. This design of normal bisimulation, particularly the special context used in it, is guided by the factorization theorem as described above. More details about the normal bisimulation can be found in Section 3.

  Motivated by the characterization of context bisimulation in terms of normal bisimulation over basic higher-order processes, the first purpose of this paper is to build a similarly simplified behavioral theory for parameterized higher-order processes. Specifically, we study the extension of normal bisimulation, including its relevant techniques (e.g., trigger, factorization), to higher-order processes with parameterization. Although parameterization is considered in [1], the definition of higher-order processes in [1] includes both name-passing and process-passing, and the coincidence between context bisimulation and normal bisimulation is studied in the mixed language. So one does not know clearly whether the normal bisimulation can be extended to a purely higher-order setting with parameterization. To the best of our knowledge, it has not been discussed thoroughly whether such characterization, together with related techniques such as triggers, is still applicable in a purely higher-order calculus extended with the two kinds of parameterization (on names and on processes) respectively. In general, it is still not clear whether the characterization result can be extended to an enriched higher-order setting. Yet the question whether the context bisimulation has such a convenient characterization in terms of normal bisimulation in a parameterized setting is significant in broader research, as well as in improving the bisimulation theory itself (i.e., checking equivalence under all contexts is not required). It would be interesting to know the answer to this question also because it is already known that parameterization can strictly enhance the expressive power of basic process-passing [7]; thus potential difference in the behavioral theory of a more expressive calculus can be revealed.

- *Expressiveness.* There are basically two kinds of expressiveness [6]. One is usually known as absolute expressiveness, i.e., the comparison with traditional computational models such as Turing machines. To this end, Π

has been shown in [13, 16] to be Turing-complete (i.e., capable of encoding Turing machines). Some other work about the computational capability of higher-order processes can be found in [1, 15, 19, 21, 22, 23]; see [6] for a good survey in this respect. What we are interested here is relative expressiveness, i.e., given two concurrent models $\mathcal{M}_1$ and $\mathcal{M}_2$, how can we say that one model is more (or less) powerful than the other? In the remainder of this paper, whenever we use the word *expressiveness* we always mean the relative expressiveness. In earlier work on the expressiveness of higher-order processes [1, 2, 3], Sangiorgi provides a framework and some key techniques (e.g., triggers and factorization) for dealing with higher-order processes and for compiling the higher-order process calculi into the π-calculus (the first-order name-passing calculus [4]). In [1, 2], Sangiorgi proves that the compilation preserves reduction and is fully abstract. These works were then followed by more expressiveness study concerning higher-order process models. Vivas et al. [10, 24, 25] study an extension of the higher-order pi-calculus with a dynamic restriction operator, i.e., the binding of a local name is not static (and not switchable) so the scope extrusion disappears. In such a setting, Sangiorgi's strategy of encoding higher-order into first-order does not work anymore, and Vivas et al. work out a different strategy to translate higher-order processes with dynamic restriction into the first-order pi-calculus. In a sense, these results reveal that dynamic binding is more intricate than static binding, which may be the reason that the latter seems to be frequent in the field. Furthermore in [26], a hierarchy of higher-order calculi based on the order (i.e., the degree of abstraction) of processes is shown to correspond to the hierarchy of πI based on similar variation on names. The calculus πI is a variant of π that only admits private communications. The extensive study of the resulting hierarchy of these calculi and their relative expressiveness has been instructive for later work, in the sense that one knows from these works that name-passing appears to be more basic than process-passing. In [27] (later improved in [28]), a continuation-passing variant of Π (Homer) is put forward and shown to be able to encode π. Their encoding satisfies a few important properties, like the operational correspondence property (but only for silent actions). All the previous works on expressiveness benefit us in giving the insight that Π is not sufficiently expressive if it is not extended with more powerful operators, though these works do not depend on some uniform criterion for their conclusion (i.e., the related encodings satisfy different set of properties).

Regarding the extension of Π, in [9], Thomsen puts forth a strategy for translating π into Π^r (Π with the relabelling operator). However the Π^r model itself is debatable since relabelling is criticized for its excessively strong semantics of being able to change any channel names arbitrarily [29]. Comparatively, parameterization appears to be more reasonable in semantics and easy to manipulate in practice [1]. For example, among others, Sangiorgi and Walker propose a fully abstract encoding of a variant of first-order pi-calculus, i.e., the localized pi-calculus in which only the output capability of names can be communicated, into an asynchronous higher-order pi-calculus with the parameterization mechanism [5]. That said, it would be good to know whether parameterization promotes process-passing in expressiveness. Lanese et al. [7] show the answer is positive, based on another result, i.e., HO_{n+1} is strictly more expressive than HO_n (this actually pinpoints a striking difference from name-passing [5]), where HO_n denotes $n-$ary polyadic Π calculus. The work of [7] potentially offers an approach to increase the expressiveness of the basic calculus Π by enabling process parameterization, as well as a few important techniques of handling process parameterization. In these related work, however, little study is done about the expressivenss of name parameterization. It would be interesting and worthwhile to know more in this direction, especially the relationship between the two kinds of parameterization. *This leads to the second main purpose of this paper, that is, to compare parameterizations on names and processes.*

Last but not least, it is worth stressing some connections between the two aspects above. Concerning parameterization, normal bisimulation could be very helpful for relevant research topics. In particular, for expressiveness studies, a simpler yet equivalent form of context bisimulation would probably allow one to simplify the proof of the correctness of encodings (like soundness, i.e., two processes are equivalent only if their encodings are equivalent). Such a kind of proofs usually require the construction of context bisimulations. For instance, when comparing name-passing and process-passing [2, 30, 31], soundness demands that two name-passing processes are equivalent (e.g., early bisimilar) only if their translations to process-passing processes are context bisimilar. However as mentioned, context bisimulation is a particularly heavy burden to establish. In this case, instead of working directly with context bisimulation, establishing normal bisimulation (the so-called effective behavioral equivalence) would be far more convenient.

Contribution

In this paper, we work on bisimulation theory and expressiveness in the setting of purely higher-order processes with parameterization. We first explicitly examine the simpler characterization of context bisimulation in the settings of process parameterization separately, so that the essential difference from the basic calculus Π can be revealed. We then study the relative expressiveness of the two different parameterizations, and in this respect, we confine to second-order processes for simplicity. Here second-order, which is frequent in programming languages, basically means that any parameterization does not occur on another parameterization (no currying, i.e., a function does not return another one), and does not take another parameterization as its argument (i.e., a function does not take another one as its input). We do not know if our results on expressiveness can be extended to arbitrary case.

The contribution of this paper is twofold.

- We show that in the calculus Π_n^D, normal bisimulation characterizes context bisimulation. We prove the coincidence by re-exploiting the available methods from the field [1, 3]. A technical novelty here is that the proof is given in a more direct way, rather than first resorting to an intermediate bisimulation called triggered bisimulation by restricting to a sub-class of processes as described above. However, we do not know if the normal bisimulation exists in Π_n^d, about which we provide some discussion.

- We show that name parameterization is at least as expressive as process parameterization, in the sense that Π_n^D can be encoded by Π_n^d. The encoding strategy exploits the idea of triggers (as well as factorization) similar to that underlying normal bisimulation. We justify our encoding witha well-established set of criteria considered in the research on expressiveness, and shows the compliance of the encoding with the criteria.

Organization

The rest of the paper is organized as follows. In Section 2, we define the calculi used in this paper, and the context bisimulation. Section 3 discusses the characterization of context bisimulation in Π_n^D. In Section 4, we show that Π_n^D can be encoded in Π_n^d. Section 5 concludes the paper, and points out some directions for future work.

2. Preliminary

In this section, we give the formal definitions of the calculus Π, and its variants Π_n^D and Π_n^d. Lowercase letters represent channel names (ranged over by u, v, w). For convenience, names are divided into two disjoint subsets: name constants (ranged over by $a, b, c, ..., n$) and name variables (ranged over by x, y, z).

2.1. Calculus Π

The Π processes, denoted by uppercase letters ($A, E, P, Q, ...$) and their variant forms, are defined by the following grammar (Π_{seg} is used to provide convenience for defining the variants of Π). Notice that X, Y, Z stand for process variables, and U, K are reserved for defining Π_n^D and Π_n^d.

$$P ::= \Pi_{seg}$$
$$\Pi_{seg} ::= 0 \mid X \mid u(X).P \mid \overline{u}P.P \mid P \mid P \mid (c)P \mid !u(X).P \mid !\overline{u}P.P$$

The operators include: prefix ($u(X).P$ in which X is bound and $\overline{u}P'.P$), parallel composition ($P \mid P'$), restriction ($(c)P$) in which c is bound (or local); they have their standard meaning, and parallel composition has the least precedence. We also have guarded replication ($!u(X).P$ and $!\overline{u}P'.P$) that suffices for our work. In fact, it can be regarded as a derived operator [9, 13]: $!\phi.P \stackrel{\text{def}}{=} (c)(Q_c \mid \overline{c}Q_c)$, $Q_c \stackrel{\text{def}}{=} c(X).(\phi.(X \mid P) \mid \overline{c}X)$, where c is a fresh name in P and ϕ is a prefix. Yet for the sake of simplicity, we stick to the primitive replication with its standard semantics (see below). As usual some convenient notations are: a for $a(X).0$; $\overline{a}$ for $\overline{a}0.0$; $\overline{m}A$ for $\overline{m}A.0$; $\tau.P$ for $(a)(a(X).P \mid \overline{a})$ (X does not appear in P); sometimes $\overline{a}[A].P$ for $\overline{a}A.P$ (for the sake of clarity); $\widetilde{\cdot}$ for a finite sequence of some items (e.g., names, processes), and $\widetilde{cd}$ for the concatenation of $\widetilde{c}$ and $\widetilde{d}$. By standard definition [1], $fn(\widetilde{P})$, $bn(\widetilde{P})$, $n(\widetilde{P})$; $fv(\widetilde{P})$, $bv(\widetilde{P})$, $v(\widetilde{P})$ respectively denote free names (names that do not lie in the range of any restriction operator), bound names (names that lie in the range of some restriction operator), names (the union of both free names and bound names); free variables (variables that do not lie in the range of any input operator), bound variables (variables that lie in the

range of some input operator) and variables (the union of both free variables and bound variables) in $\widetilde{P}$. Closed processes contain no free variables, and are studied by default. A fresh name or variable is one that does not occur in the processes under consideration. Name substitution $P\{\widetilde{n}/\widetilde{m}\}$ and higher-order substitution $P\{\widetilde{A}/\widetilde{X}\}$ are defined structurally in the standard way [5]. $E[\widetilde{X}]$ denotes process E with free variables included in $\widetilde{X}$, and $E[\widetilde{A}]$ stands for $E\{\widetilde{A}/\widetilde{X}\}$. We work up to α-conversion with renaming possibly involved to avoid capture of free names.

The operational semantics is given in Figure 1. Symmetric rules are omitted. Symbols $\alpha, \beta, \lambda, ...$ denote actions. The subject of an action (e.g., a in action $a(A)$) indicates the channel name on which the action happens. Operations $fn(), bn(), n(), fv(), bv(), v()$ can be similarly defined on actions. The first two rules (PRE1 and PRE2) are the prefix rules for input and output actions. In higher-order input ($a(A)$), process A is received through channel a, and becomes part of the receiving environment through a substitution; in higher-order output ($(\widetilde{c})\overline{a}A$), process A is sent with a set of local names for prospective use in further communication. The third rule (RES1) is the restriction rule: whenever P can reach P' by performing an action λ, if c does not lie in the name set of λ, then the same action can be activated from $(c)P$ and leads to $(c)P'$, as c has no effect on the execution of λ. In the forth rule (RES2), the notation $fn(A)-\{\widetilde{c}, a\}$ means the free names of A excluding the names in $\widetilde{c}$ plus a, which rules out the possibility that d is a . Compared with the third rule (RES1), this rule (known as the open rule [4]) characterizes the scenario when one more secret name is carried by the output action. The fifth rule (PAR1) describes the situation that a component of a concurrent system has an action, i.e., whenever P (the component) has an action λ, $P \mid P_1$ (the system) can perform the same action with no effect on P_1. The side condition requires that the bound name of the action λ would not be a free name in P_1 so as to avoid capture in potential interaction [4], as depicted in the next rule (PAR2). The sixth rule (PAR2) characterizes interaction. It means if process P_1 can perform an input action through channel a and simultaneously another parallel process P_2 can perform an output action through the same channel, then P_1 and P_2 can communicate with each other (by performing the silent action τ). The last two rules (REP1 and REP2) take care of the replication operation. Intuitively, for example, $!a(X).P$ means that one has infinite many copies of $a(X).P$. Whenever a copy of $a(X).P$ is used (by performing the action $a(A)$), one gets its suffix P in parallel with the original process $!a(X).P$.

Relation $\Rightarrow$ is the reflexive transitive closure of the internal transition ($\xrightarrow{\tau}$), and $\stackrel{\lambda}{\Rightarrow}$ is $\Rightarrow\xrightarrow{\lambda}\Rightarrow$. Relation $\stackrel{\hat{\lambda}}{\Rightarrow}$ is $\Rightarrow$ when λ is τ and $\stackrel{\lambda}{\Rightarrow}$ otherwise. Notation $\xrightarrow{\tau}_k$ means k consecutive τ's. For a binary relation $\mathcal{R}$ over processes, $P\Rightarrow \cdot \mathcal{R}\, Q$ means $P\Rightarrow Q'$ for some Q' such that $Q'\,\mathcal{R}\,Q$ (i.e., $(Q', Q) \in \mathcal{R}$). We say that a relation $\mathcal{R}$ is closed under (variable) substitution if $(E\{A/X\}, E'\{A/X\}) \in \mathcal{R}$ for any A, X whenever $(E, E') \in \mathcal{R}$, in which E, E' (possibly) have free occurrence of variable X. A process diverges if it can perform an infinite τ sequence.

$$\text{(PRE1)}\ \frac{}{a(X).P \xrightarrow{a(A)} P\{A/X\}} \qquad \text{(PRE2)}\ \frac{}{\overline{a}A.P \xrightarrow{\overline{a}A} P}$$

$$\text{(RES1)}\ \frac{P \xrightarrow{\lambda} P'}{(c)P \xrightarrow{\lambda} (c)P'}\, c \notin n(\lambda) \qquad \text{(RES2)}\ \frac{P \xrightarrow{(\widetilde{c})\overline{a}[A]} P'}{(d)P \xrightarrow{(d\widetilde{c})\overline{a}[A]} P'}\, d \in fn(A)-\{\widetilde{c}, a\}$$

$$\text{(PAR1)}\ \frac{P \xrightarrow{\lambda} P'}{P \mid P_1 \xrightarrow{\lambda} P' \mid P_1}\, bn(\lambda) \cap fn(P_1) = \emptyset \qquad \text{(PAR2)}\ \frac{P_1 \xrightarrow{a(A)} P_1' \quad P_2 \xrightarrow{(\widetilde{c})\overline{a}[A]} P_2'}{P_1 \mid P_2 \xrightarrow{\tau} (\widetilde{c})(P_1' \mid P_2')}\, \{\widetilde{c}\} \cap fn(P_1) = \emptyset$$

$$\text{(REP1)}\ \frac{}{!a(X).P \xrightarrow{a(A)} P\{A/X\} \mid !a(X).P} \qquad \text{(REP2)}\ \frac{}{!\overline{a}A.P \xrightarrow{\overline{a}A} P \mid !\overline{a}A.P}$$

Figure 1: Semantics of Π

2.2. *Calculi* Π_n^D *and* Π_n^d

Parameterization (of arity $n \geq 1$) extends Π with the syntax and semantics in Figure 2 (recall we use Π_{seg} for the sake of convenience). We use U and K (and their indexed-version) to stand for the variable (process variable or name variable) in parameterization and the object (process or name) in application. Formally, $\langle U_1, U_2, ..., U_n\rangle P$ is an n-ary parameterization where $U_1, U_2, ..., U_n$ are the pairwise distinct formal parameters to be instantiated, and $P\langle K_1, K_2, ..., K_n\rangle$ is an n-ary application where $K_1, K_2, ..., K_n$ are the concrete objects. Application binds tighter than

prefixes and restriction. Sometimes we use $\widetilde{U}$ (respectively $\widetilde{K}$) instead of $U_1, U_2, ..., U_n$ (respectively $K_1, K_2, ..., K_n$) if the arity is clear. Accordingly we will use $|\widetilde{U}|$ (respectively $|\widetilde{K}|$) to stand for the arity of parameterization (respectively application). In the transition rule of Figure 2, $|\widetilde{U}|=|\widetilde{K}|$ means the sequence of parameters and the sequence of instantiating objects should be equally sized whenever an initialization is possible. This rule states that the parameterized process can do an action only after the application happens. It is now ready to define Π^D_n and Π^d_n:

- Calculus Π^D_n, which has process parameterization (or higher-order abstraction) of arity n, is defined by setting $\widetilde{U}, \widetilde{K}$ to be $\widetilde{X}, \widetilde{P}$ respectively. Calculus Π^D stands for the union of all Π^D_n.
- Calculus Π^d_n, which has name parameterization (or first-order abstraction) of arity n, is defined by setting $\widetilde{U}, \widetilde{K}$ to be $\widetilde{x}, \widetilde{u}$ respectively. Calculus Π^d stands for the union of all Π^d_n.

$$P ::= \Pi_{seg} \mid \langle U_1, U_2, ..., U_n \rangle P \mid P\langle K_1, K_2, ..., K_n \rangle$$

$$\frac{P\{\widetilde{K}/\widetilde{U}\} \xrightarrow{\lambda} P'}{P''\langle \widetilde{K} \rangle \xrightarrow{\lambda} P'} \quad \text{if } P'' \stackrel{\text{def}}{=} \langle \widetilde{U} \rangle P \ \ (|\widetilde{U}| = |\widetilde{K}| = n)$$

Figure 2: Π with parameterization of arity n

The forms of process expressions (or terms) can be stipulated by a type system provided in Sangiorgi's seminal work [1]. Such a system is however not important for this paper (for types do not play a role in reductions); interested readers can refer to [1, 5]. Terms of the form $\langle \widetilde{X} \rangle P$ or $\langle \widetilde{x} \rangle P$, in which $\widetilde{X}$ and $\widetilde{x}$ are not empty, are *parameterized processes*. Terms without outmost parameterization are *non-parameterized processes*, or simply processes. Only free variables can be effectively parameterized (i.e., it does not make sense to parameterize a bound variable); they become *bound* after parameterization. In the syntax, redundant parameterizations, for example $\langle X_1, X_2 \rangle P$ in which $X_2 \notin fv(P)$, are allowed. A Π^D_n ($n \geq 1$) process is therefore definable in Π^D_{n+1} (similar for Π^d_n); for example, $\langle \widetilde{X} \rangle P$ can be coded up by setting an additional fresh dummy variable Y (of no use) to obtain $\langle \widetilde{X}, Y \rangle P$.

For the purpose of this paper, we focus on two different variants in Sections 3 and 4. For the sake of simplicity in the study of bisimulation characterizations (Section 3), we stipulate that the processes of Π^D_n and Π^d_n are strictly abstraction-passing, i.e., all the transmitted objects are parameterized processes; accordingly, it is assumed that all the process variables would be instantiated by abstractions, so an occurrence of a variable X typically takes the form $X\langle P' \rangle$ in some context. This can be justified by two facts: firstly a non-parameterized process can be treated as a special case of parameterization; secondly, and more related to the aim of this paper, the characterization of context bisimulation in the case of non-parameterized processes has been examined in depth by Sangiorgi [3]. Separately, in the study of expressiveness in this paper (Section 4), as mentioned, we confine to second-order processes, in which parameterizing does not occur on parameterized processes, and application does not use parameterized processes as instances. That is, we stipulate that in $\langle \widetilde{U} \rangle P'$ and $P\langle \widetilde{P'} \rangle$, P' and the terms in $\widetilde{P'}$ are not parameterized processes. This prohibits processes such as $\langle X \rangle (\langle Y \rangle P)$ and $(\langle X \rangle (X\langle A \rangle))\langle A' \rangle$ in which A' is $\langle Y \rangle Y$; instead, in the former one could use $\langle X, Y \rangle P$, and in the latter the instance A could be sent to a service containing A' before retrieving the result (in programming practice second-order is most frequent and basically sufficient).

In summary, Section 3 restricts the transmitted objects to be abstractions, but does not require processes to be second-order. On the other hand, Section 4 restricts processes to be second-order, but allows the transmission of simple processes.

The semantics of parameterization makes it somewhat natural to deem application as some (extra) rule of structural congruence [32, 3], denoted by $\equiv$, which is defined in Figure 3. Notice that in addition to the standard algebraic laws (concerning parallel composition and restriction), the last rule of application is included. So the rule below can be used in place of that in Figure 2. We will use the resulting semantics in the remainder of the paper.

$$\frac{P \equiv P_1 \quad P_1 \xrightarrow{\alpha} P_2 \quad P_2 \equiv P'}{P \xrightarrow{\alpha} P'}$$

$P \equiv P'$ (if P and P' can be α-convertible to each other)

$P \mid 0 \equiv P, \quad P \mid (P' \mid P'') \equiv (P \mid P') \mid P'', \quad P \mid P' \equiv P' \mid P$

$(c)(d)P \equiv (d)(c)P$, $(c)\lambda.P \equiv 0$ whenever the subject of λ is c

$(c)(P \mid P') \equiv (c)P \mid P'$ whenever $c \notin fn(P')$

$(\langle \widetilde{U} \rangle P)\langle \widetilde{K} \rangle \equiv P\{\widetilde{K}/\widetilde{U}\}, |\widetilde{U}| = |\widetilde{K}| = n$

Figure 3: Structural congruence

2.3. Context Bisimulation

The bisimulation equivalence we intend to examine is context bisimulation. The basic form of its definition is the same for all the calculi defined above. Notice that we use '$\mathcal{L}-$process' to mean a process of $\mathcal{L}$, which can be Π, Π^D_n, or Π^d_n in this paper.

Definition 1 (Context bisimulation)**.** A symmetric binary relation $\mathcal{R}$ on closed $\mathcal{L}-$processes is a context bisimulation, if whenever $P \mathcal{R} Q$ the following properties hold:

1. If $P \xrightarrow{\alpha} P'$ and α is τ or $a(A)$, then $Q \stackrel{\hat{\alpha}}{\Rightarrow} Q'$ for some Q' and $P' \mathcal{R} Q'$;
2. If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$ then $Q \stackrel{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ for some $\widetilde{d}, B, Q'$, and for every $\mathcal{L}-$process $E[X]$ s.t. $\{\widetilde{c}, \widetilde{d}\} \cap fn(E) = \emptyset$ it holds that $(\widetilde{c})(P' \mid E[A]) \mathcal{R} (\widetilde{d})(Q' \mid E[B])$.

Process P is context bisimilar to Q, written $P \approx_{\mathcal{L}} Q$, if $P \mathcal{R} Q$ for some context bisimulation $\mathcal{R}$. Relation $\approx_{\mathcal{L}}$ is called context bisimilarity.

Context bisimulation can be extended to general (open) processes in the standard way, i.e., suppose $fv(P) \cup fv(P') \subseteq \widetilde{X}$, then $P \approx_{\mathcal{L}} P'$ if $P\{\widetilde{A}/\widetilde{X}\} \approx_{\mathcal{L}} P'\{\widetilde{A}/\widetilde{X}\}$ for all closed $\mathcal{L}$-processes $\widetilde{A}$. Similarly the extension to parameterized processes is: $\langle \widetilde{X} \rangle P \approx_{\mathcal{L}} \langle \widetilde{X} \rangle P'$ if $P\{\widetilde{A}/\widetilde{X}\} \approx_{\mathcal{L}} P'\{\widetilde{A}/\widetilde{X}\}$ for all closed $\mathcal{L}$-processes $\widetilde{A}$. Relation $\sim_{\mathcal{L}}$ is the strong version of $\approx_{\mathcal{L}}$ in the sense that every step of P should be simulated by exactly one step of Q and vice versa (i.e., replace all $\Rightarrow$ in the definition of $\approx$ with $\rightarrow$ and remove the hat on α if any). It is clear that $\equiv_{\mathcal{L}} \subseteq \sim_{\mathcal{L}} \subseteq \approx_{\mathcal{L}}$ [5] ($\equiv_{\mathcal{L}}$ denotes the structural congruence of $\mathcal{L}$). Sometimes we simply use $\approx$ (respectively $\sim$) when the corresponding language is clear from context.

It is well-known that context bisimilarity is an equivalence and a congruence; in Sangiorgi's higher-order π-calculus (which somewhat comprises Π^D, Π^d, and the name-passing π-calculus) context bisimilarity coincides with the contextual equivalence (i.e., barbed congruence); see [1, 2, 3, 26]. It is however unclear whether similar coincidence exists in Π^D or Π^d respectively.

Notice that $E[X]$ is a bit different from the well-known concept of context, which is standardly defined as a process with a hole $[\cdot]$ in place of 0 in the grammar defining process expressions. For the sake of completeness, below we define below the notion of context, written $G[\cdot]$, and $G[A]$ is the process obtained by filling the hole with A. Related concepts and operations (e.g., structural congruence and substitution) can be extended to contexts.

$$G[\cdot] ::= [\cdot] \mid b(Y).G[\cdot] \mid \overline{b}G[\cdot].P \mid \overline{b}P.G[\cdot] \mid (c)G[\cdot] \mid G[\cdot] \mid P \mid G[\cdot]\langle P \rangle \mid \langle Y \rangle G[\cdot]$$

A point here is that context allows name and variable capture, e.g., the case a free name falls into the scope of some restriction of the same name is allowed. On the other hand, $E[X]$, as used in context bisimulation, does not allow name capture and should use α-conversion to avoid that. Otherwise, for instance, α-convertible processes $(m)\overline{a}[m.0].\overline{m}.b$ and $(n)\overline{a}[n.0].\overline{n}.b$ would be distinguishable by context bisimilarity using $E[X] \equiv (m)X$ as the receiving environment in the output clause of the definition of context bisimulation (i.e., the latter can produce a visible action on b while the former cannot), which contradicts the fact that α-convertibility shall entail context bisimilarity [1].

The above context bisimulation is given in the *late* style [1, 2, 3, 26] . If we replace the second clause in Definition 1 with the following one:

> If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$ then for every $\mathcal{L}-$process $E[X]$, there exists some $\widetilde{d}, B, Q'$, $\{\widetilde{c}, \widetilde{d}\} \cap fn(E) = \emptyset$, s.t. $Q \overset{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ and $(\widetilde{c})(P' \mid E[A])\ \mathcal{R}\ (\widetilde{d})(Q' \mid E[B])$

and keep all the other requirements unchanged, then we get the so-called *early* context bisimulation, which will be used in Section 4. We use $\asymp$ to stand for early context bisimilarity.

3. Normal bisimulation for parameterized processes

This section deals with the bisimulation theory for parameterized processes, specifically the simpler characterization for context bisimulation in Π_n^D. In order to make the paper self-contained, we start by recalling in Section 3.1 the well-known work on Π by Sangiorgi [1], then present our work on Π_n^D in Section 3.2.

3.1. Normal bisimulation in Π

We exemplify the characterization of context bisimulation by means of normal bisimulation in basic higher-order processes [3]. Recall that $E[X]$ denotes a process with the free occurrence of X (i.e., not bound by an input prefix $a(X).P$), $E[A]$ denotes substituting A for all free occurrences of X in $E[X]$ (possibly renaming some restricted names in $E[X]$ if necessary to avoid capture), output action $(\widetilde{c})\overline{a}A$ means sending over name a a process A containing a set $\widetilde{c}$ of bound names, and the replication operation $!m.A$ is treated as primitive in the higher-order setting. We stick to the convention of using $\approx$ for context bisimilarity (i.e., the equivalence induced by context bisimulation), and $\cong$ for normal bisimilarity. First of all, the factorization theorem is stated as below (m is a fresh name, i.e., it does not occur in $E[A]$). As mentioned, the use of this theorem is to relocate a (sub-)process A to a new place by setting up a pointer to that new place, and the pointer is achieved by the trigger Tr_m.

$$E[A] \approx (m)(E[Tr_m] \mid !m.A), \qquad \text{where the trigger } Tr_m \overset{\text{def}}{=} \overline{m}.0$$

Intuitively, using the factorization theorem one can extract from $E[A]$ the process A that might cause difference to its behavior. For instance,

$$A \mid Q \approx (m)((\overline{m}.0 \mid Q) \mid !m.A)$$

Now suppose P and Q are basic higher-order processes, and are bisimilar with respect to either bisimulation equivalence (i.e., context bisimulation or normal bisimulation). We focus on the output clauses of context bisimulation and normal bisimulation, since the input case is similar. Note that $fn(E[X])$ returns the free names of $E[X]$ (i.e., not bound by restriction), and for simplicity in the following whenever we use $Q \overset{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ we always mean: there exists $\widetilde{d}, B$ and Q' such that after performing the action $(\widetilde{d})\overline{a}B$ along with some possible τ actions, process Q evolves into Q'.

- The output clause of context bisimulation is
  If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$, then $Q \overset{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ s.t. for every $E[X]$ with $fn(E[X]) \cap (\widetilde{c} \cup \widetilde{d}) = \emptyset$, $(\widetilde{c})(P' \mid E[A]) \approx (\widetilde{d})(Q' \mid E[B])$.
- The output clause of normal bisimulation is
  If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$, then $Q \overset{(\widetilde{d})\overline{a}B}{\Longrightarrow} Q'$ s.t. for some fresh m, $(\widetilde{c})(P' \mid !m.A) \cong (\widetilde{d})(Q' \mid !m.B)$.

The intuition behind the simplification is that one can use the factorization theorem to extract process A (respectively B) in $E[A]$ (respectively $E[B]$), so in context bisimulation one obtains the following, due to the congruence property of $\approx$,

$$\begin{array}{ccc} R_1 \overset{\text{def}}{=} (\widetilde{c})(P' \mid (m)(E[Tr_m] \mid !m.A)) & \overset{\approx}{\cdots\cdots} & (\widetilde{d})(Q' \mid (m)(E[Tr_m] \mid !m.B)) \overset{\text{def}}{=} R_2 \\ \vdots\ \approx & & \vdots\ \approx \\ (\widetilde{c})(P' \mid E[A]) & \overset{\approx}{\cdots\cdots} & (\widetilde{d})(Q' \mid E[B]) \end{array}$$

where each dotted line connects two processes that are related by context bisimilarity ($\approx$). Then by the congruence property of $\approx$, one can eliminate the common part in processes R_1 and R_2 (i.e., $E[Tr_m]$ and restriction on m), and thus arrive at the form of the output clause in normal bisimulation.

3.2. Normal bisimulation in Π_n^D

We now show that in Π_n^D normal bisimulation characterizes context bisimulation, thus extending the result in Π. For convenience, we focus on Π_1^D; the result can be readily extended to Π_n^D. We first define normal bisimulation, and then prove the coincidence theorem. Hereinafter $Tr_m \stackrel{\text{def}}{=} \langle Z\rangle\overline{m}Z$ denotes a trigger with (trigger name) m. The following definition portraits the normal bisimulation, in which the triggers and special environments are adapted to the setting allowing process parameterization. Notice that an alternative way to define this bisimulation is to use abstractions/concretions like in [3]. We stick to the original approach presented in [1] because this makes the definition and related arguments simpler.

Definition 2. A symmetric binary relation $\mathcal{R}$ on closed processes of Π_1^D is a normal bisimulation, if whenever $P\,\mathcal{R}\,Q$ the following properties hold:

1. If $P \xrightarrow{\tau} P'$, then $Q \Rightarrow Q'$ for some Q' s.t. $P'\,\mathcal{R}\,Q'$;
2. If $P \xrightarrow{a(Tr_m)} P'$ and $Tr_m \equiv \langle Z\rangle\overline{m}Z$ (m is fresh w.r.t. P and Q), then $Q \xRightarrow{a(Tr_m)} Q'$ for some Q' s.t. $P'\,\mathcal{R}\,Q'$;
3. If $P \xrightarrow{(\widetilde{c})\overline{a}A} P'$ then $Q \xRightarrow{(\widetilde{d})\overline{a}B} Q'$ for some $\widetilde{d}, B, Q'$, and it holds that (m is fresh)

$$(\widetilde{c})(P' \mid !m(Z).A\langle Z\rangle)\ \mathcal{R}\ (\widetilde{d})(Q' \mid !m(Z).B\langle Z\rangle)$$

which can be rephrased as $(\widetilde{c})(P' \mid E'[A])\ \mathcal{R}\ (\widetilde{d})(Q' \mid E'[B])$ where $E'[X] \stackrel{\text{def}}{=} !m(Z).X\langle Z\rangle$ is special, in contrast to context bisimulation (in which a universal $E[X]$ is required).

Process P is normal bisimilar to Q, written $P \cong Q$, if $P\,\mathcal{R}\,Q$ for some normal bisimulation $\mathcal{R}$. Relation $\cong$ is called normal bisimilarity.

Remark 1. The strong version of $\cong$ is denoted by $\simeq$. In general, $\cong_{\mathcal{L}}$ (respectively $\simeq_{\mathcal{L}}$) indicates the normal bisimilarity (respectively strong normal bisimilarity) of calculus $\mathcal{L}$, if any; we simply use $\cong$ (respectively $\simeq$) when there is no confusion. It can be shown in a standard way that normal bisimilarity is *an equivalence and a congruence*; see [1, 3] for a reference. Analogous to the case for Π, the form of normal bisimulation and the proof schema for Theorem 1 to follow stem from the triggers and the Factorization theorem (Theorem 4). To this end, we go beyond the related works [1, 3] in several respects. Firstly, the processes under inspection here are purely higher-order without name-passing, and capable of parameterization on processes only. Secondly, although the schema for the proof of the characterization of context bisimulation using normal bisimulation exploits those works, the technical details are not the same, that is, we prove the coincidence of normal bisimulation and context bisimulation in a more direct and easier way. More specifically, we avoid using the (indirect) approaches in [12], where the so-called index technique is essentially used to deal with actions, and in [1, 3], where an intermediate equivalence called triggered bisimulation is used and processes are first transformed into a subclass called triggered processes.

The rest of this section is mainly devoted to the (technical) proof of the following theorem.

Theorem 1. *In* Π_1^D*, normal bisimilarity coincides with context bisimilarity; that is,* $\cong = \approx$.

A key step in proving Theorem 1 is to prove the Factorization theorem (Theorem 4). First we need some preparation, i.e., the two lemmas below (Lemma 2 and Lemma 3) that are useful in the proof of the Factorization theorem. We use $fn(E_1, E_2, ..., E_n)$ as an abbreviation for $fn(E_1) \cup fn(E_2) \cup \cdots \cup fn(E_n)$. Intuitively, these two lemmas state some distributive laws concerning the replication $!m(Z).A\langle Z\rangle$ and the various process operators. Lemma 2 deals with the cases of prefixes and parallel composition, and Lemma 3 tackles the abstraction operation. Notice that in the first clause of Lemma 2, the output prefix does not have any free occurrence of m, differently from the second clause of that lemma. The proofs of the two lemmas are placed in Appendix A.

Lemma 2. *Suppose* $E[X], E_1[X], E_2[X]$ *belong to* Π_1^D*, and assume* $m \notin fn(E, E_1, E_2, A)$.

(1) If $m \notin fn(\alpha)$ and $bv(\alpha) \notin fv(A)$, then

$$(m)(\alpha.E[Tr_m] \mid !m(Z).A\langle Z\rangle) \approx \alpha.(m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

(2) It holds for output prefix that

$$(m)(\overline{a}B_1.E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \approx (m)(\overline{a}B_2.E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

where $B_1 \equiv E_2[Tr_m]$, $B_2 \equiv (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)$.

(3) It holds for parallel composition that

$$\begin{array}{ll} & (m)(E_1[Tr_m] \mid E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) \\ \approx & (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \mid (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) \end{array}$$

Lemma 3. *Suppose $E_1[X]$ belongs to Π_1^D. For all abstractions A, B, and m fresh (i.e., $m \notin fn(E_1, A, B)$), it holds that*

$$B\langle(m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)\rangle \approx (m)(B\langle E_1[Tr_m]\rangle \mid !m(Z).A\langle Z\rangle)$$

Now we show the Factorization theorem. As mentioned, this theorem offers some method to relocate a subprocess, which might cause difference in behavior, and set up a reference to it with the help of a trigger, while maintaining the equivalence with respect to context bisimilarity. The proof of Theorem 4 can be found in Appendix A.

Theorem 4 (Factorization)**.** *Given a $E[X]$ of Π_1^D, it holds for every abstraction A and fresh m (i.e., $m \notin fn(E, A)$) that*

(1) if $E[Tr_m]$ is non-parameterized, i.e., not an abstraction, then

$$E[A] \approx (m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

(2) else if $E[Tr_m] \equiv \langle Y_1\rangle \cdots \langle Y_k\rangle E'$ for some $k \geq 1$ and non-parameterized E', then

$$E[A] \approx \langle Y_1\rangle \cdots \langle Y_k\rangle((m)(E' \mid !m(Z).A\langle Z\rangle))$$

With the help of factorization theorem (Theorem 4), we can now give the proof of Theorem 1.

Proof of Theorem 1. The fact that $\approx$ implies $\cong$ barely needs argument, because the former demands more and the latter is actually a special case. So we focus on the other direction. To achieve this, we use the factorization theorem to show that the relation $\mathcal{R} \stackrel{\text{def}}{=} \{(P, Q) \mid P \cong Q\}$ (i.e., normal bisimilarity $\cong$) is a context bisimulation up-to $\approx$ (so that $\mathcal{R} \subseteq \approx$). The definition of the up-to technique is as usual: in each clause of Definition 1 replace each strong transition with weak transition, $\mathcal{R}$ with $\approx \mathcal{R} \approx$, and moreover — in order to be sound — $\hat{\alpha}$ with α (i.e., no nil simulation of internal moves could happen); see [5, 33] for the standard definition and its soundness. We focus on the case when the first result (1) of Theorem 4 applies, the case when the other applies can be handled in a similar way. There are several cases to analyze.

- Internal action. This case is trivial, because the clauses in context bisimulation and normal bisimulation are the same.
- Input. If $P \stackrel{a(A)}{\Longrightarrow} P'$, then we want to show that

$$Q \stackrel{a(A)}{\Longrightarrow} Q' \text{ for some } Q' \tag{1}$$
$$\text{and } P' \approx \mathcal{R} \approx Q' \tag{2}$$

  W.l.o.g., suppose $P' \equiv E[A]$ for some $E[X]$ (i.e., from $P \stackrel{a(X)}{\Longrightarrow}$ intuitively). So $P \stackrel{a(Tr_m)}{\Longrightarrow} E[Tr_m]$ for some fresh m. Since $P \cong Q$, we know

$$Q \stackrel{a(Tr_m)}{\Longrightarrow} F[Tr_m] \text{ for some } F$$

and

$$E[Tr_m] \cong F[Tr_m] \tag{3}$$

Thus

$$Q \overset{a(A)}{\Longrightarrow} F[A] \overset{\text{def}}{=} Q'$$

which fulfills (1). We know from (3) and the congruence properties of $\cong$ that

$$(m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle) \cong (m)(F[Tr_m] \mid !m(Z).A\langle Z\rangle)$$

Now by the factorization theorem (Theorem 4), we have

$$E[A] \approx (m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)\,\mathcal{R}\,(m)(F[Tr_m] \mid !m(Z).A\langle Z\rangle) \approx F[A]$$

which yields (2).

- Output. If $P \overset{(\widetilde{c})\bar{a}A}{\Longrightarrow} P'$, then we want to show that

$$Q \overset{(\widetilde{d})\bar{a}B}{\Longrightarrow} Q' \text{ for some } \widetilde{d}, B, Q' \tag{4}$$

$$\text{and for every } E[X]\ (\{\widetilde{c},\widetilde{d}\} \cap fn(E) = \emptyset),\ (\widetilde{c})(E[A] \mid P') \approx\ \mathcal{R}\ \approx (\widetilde{d})(E[B] \mid Q') \tag{5}$$

The argument can be conducted in a way pretty similar to that in the previous case for input; this time one attaches a process $E[Tr_m]$ and also uses the congruence properties of $\cong$.

Since $P \cong Q$, we have $Q \overset{(\widetilde{d})\bar{a}B}{\Longrightarrow} Q'$ which fulfills (4), and also

$$(\widetilde{c})(P' \mid !m(Z).A\langle Z\rangle) \cong (\widetilde{d})(Q' \mid !m(Z).B\langle Z\rangle) \tag{6}$$

We know from the congruence property and (6) that (notice we assume $\{\widetilde{c},\widetilde{d}\} \cap fn(E) = \emptyset$)

$$R_1 \overset{\text{def}}{=} (m)((\widetilde{c})(P' \mid !m(Z).A\langle Z\rangle) \mid E[Tr_m]) \cong (m)((\widetilde{d})(Q' \mid !m(Z).B\langle Z\rangle) \mid E[Tr_m]) \overset{\text{def}}{=} R_2 \tag{7}$$

By some simple structural adjustment and the factorization theorem (Theorem 4), we know R_1 and R_2 of (7) are equivalent to $(\widetilde{c})(E[A] \mid P')$ and $(\widetilde{d})(E[B] \mid Q')$ respectively. So we have

$$(\widetilde{c})(E[A] \mid P') \approx R_1\,\mathcal{R}\,R_2 \approx (\widetilde{d})(E[B] \mid Q')$$

This is exactly what (5) says.

The proof is now complete.

□

Remark 2. The success of the characterization of context bisimulation in terms of normal bisimulation in Π_n^D can be attributed to the fact that Π_n^D processes are purely higher-order, i.e., no names can be passed but only (parameterized) processes (carrying names), and moreover no names can be parameterized. Thus one can delay the instantiation of a process parameterization by moving it elsewhere with the help of triggers. We note that in [12], the index technique is utilized to show that strong normal bisimulation characterizes strong context bisimulation, within a calculus capable of both name-passing and process-passing but without any parameterization. A critical point there is that indices can be used to precisely pinpoint the matching of actions from the processes when going through some intermediate transformation (e.g., factorization). We believe that, by combining the technique in that paper with the approach in this section, one can further show the following result.

Conjecture 5. In the calculus Π_n^D, $\sim$ and $\simeq$ coincide.

Discussion

Roughly speaking, compared with process parameterization, name parameterization offers more flexibility in expressiveness (see Section 4), and how to achieve a simpler characterization of context bisimulation in Π_n^d remains an open issue. Intuitively, we do not believe the method of normal bisimulation in Π_n^D can be applied.

In some related work [34, 35], Lenglet et al. study higher-order processes with passivation and show that, among others, abstraction-free processes are insufficient in distinguishing abstractions by context bisimulation, and this actually implies that using non-parameterized processes would not give rise to a simpler normal-like characterization of context bisimulation. This somehow corresponds well with the simpler characterization in terms of normal bisimulation in Π_n^D, for the trigger there is the abstraction $\langle Z\rangle\overline{m}Z$. Unfortunately, the technique used in [34, 35] appears not useful to show the non-existence of normal-like characterization in Π_n^d, since (as mentioned in [35]) this technique is not intended to show the impossibility of simplifying context bisimulation, and moreover the languages are quite different. Passivation behaves like some dynamic (asynchronous) process emission, i.e., the process at the output position may evolve (akin to ambient calculus in a sense [36]). This makes the interaction between processes more involved and thus somewhat short of normal bisimulation. In [34, 35], only a restriction-free sub-calculus is reported to have normal-like bisimulation, but the target bisimulation is not exactly context bisimulation but rather a higher-order bisimulation similar to that proposed by Thomsen [9]. The choice of using the higher-order bisimulation makes the discussion simpler yet is basically sufficient because it coincides with contextual equivalence (viz., barbed congruence). To this end, It would be an interesting direction to further investigate the relationship between context bisimilarity and contextual equivalence in Π_n^D. Furthermore, the calculus used in [34, 35] is purely process-passing (i.e., no abstraction-passing), and the failure trial of normal bisimulation is discussed in such a setting and not extended to some richer setting allowing abstraction-passing (this would probably render the bisimulation even more intricate in presence of passivation).

For now, a simpler characterization of context bisimulation for higher-order processes with name-parameterization is still unknown. Our discussion here may provide some insight hopefully toward a potential solution.

4. Expressiveness comparison of parameterized processes

In this section, we show that on higher-order processes, name parameterization is at least as expressive as process parameterization, in the sense that the latter can be interpreted in the former. Specifically, we show that Π_n^D can be encoded in Π_n^d, in a second-order setting (i.e., parameterization are second-order as defined in Section 2). We believe the result in this section can be extended to general order though the extension would be highly coupled with the complexity of a process and unlikely to be trivial. The encoding we are going to present essentially exploits the idea of triggers (and factorization). We justify the encoding by showing its conformance to a variant version of the criteria set from [7] that is based on the well-known one proposed in [37]; this variant is a somewhat relaxed version on the basis of [38]. In the remainder of this section, we first introduce the criteria in Section 4.1, then present our encoding in Section 4.2.

4.1. Criteria for encodings

A process model $\mathcal{L}$ is a triplet $(\mathcal{P}, \rightarrow, \simeq)$, where $\mathcal{P}$ is the set of processes, $\rightarrow \subseteq \mathcal{P}\times\mathbb{A}\times\mathcal{P}$ is the transition relation with $\mathbb{A}$ as the set of actions, and $\simeq$ is a representative behavioral equivalence (e.g., context bisimilarity).

Given $\mathcal{L}_i \stackrel{\text{def}}{=} (\mathcal{P}_i, \rightarrow_i, \simeq_i)$ (i=1, 2), and a translation function: $[\![]\!] : \mathcal{P}_1 \rightarrow \mathcal{P}_2$, we will use the symbol $[\![\mathcal{L}_1]\!]$, to stand for the set of processes inside $\mathcal{L}_2$ which have reverse image in $\mathcal{L}_1$ under the function $[\![]\!]$. Obviously $[\![\mathcal{L}_1]\!] \subseteq \mathcal{P}_2$. As proposed in [38], we will use $\dot{\simeq}_2$ to stand for behavioural equivalence restricted to $[\![\mathcal{L}_1]\!]$. Naturally $\simeq_2$ is usually stronger than $\dot{\simeq}_2$ as there may be some processes in the target model $\mathcal{L}_2$ which are not the encoding of any process in $\mathcal{L}_1$. We say $\mathcal{L}_1$ can be encoded in $\mathcal{L}_2$, notation $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ if there is an encoding function $[\![\cdot]\!] : \mathcal{P}_1 \longrightarrow \mathcal{P}_2$ that satisfies the following set of criteria.

1. *Compositionality.* For any k-ary operator op of $\mathcal{L}_1$, and all $P_1, ..., P_k \in \mathcal{P}_1$, $[\![op(P_1, ..., P_k)]\!] = C_{op}[[\![P_1]\!], ..., [\![P_k]\!]]$ for some multi-hole context $C_{op}[\cdots] \in \mathcal{P}_2$. (A multi-hole context is a context that allows more than one holes to replace a number of 0s in a process expression [5].)
2. *Name invariance.* For any injective substitution σ of names, $[\![P\sigma]\!]$=$[\![P]\!]\sigma$.

3. *Operational correspondence (forth).* Whenever $P \stackrel{\beta}{\Rightarrow} P'$, it holds $[\![P]\!] \stackrel{\lambda}{\Rightarrow} \simeq_2 [\![P']\!]$, for some action λ with the same *subject* as that of β.
4. *Operational correspondence (back).* Whenever $[\![P]\!] \stackrel{\lambda}{\Rightarrow} T$, there exist P' and β with the same subject as that of λ s.t. $P \stackrel{\beta}{\Rightarrow} P'$ and $T \Rightarrow \simeq_2 [\![P']\!]$.
5. *Weak adequacy.* $P \simeq_1 P'$ implies $[\![P]\!] \dot{\simeq}_2 [\![P']\!]$. This is also known as (weak) *soundness*. The converse is known as (weak) *completeness*.
6. *Divergence reflection.* If $[\![P]\!]$ diverges, so does P, i.e. no new divergences are introduced by the encoding.

The above criteria set can be divided into two parts: compositionality and name invariance are syntactic conditions, and the rest are semantic conditions. If $\dot{\simeq}_2$ is replaced with $\simeq_2$, then we say that $\mathcal{L}_1$ is strongly encodable in $\mathcal{L}_1$, written $\mathcal{L}_1 \sqsubseteq^s \mathcal{L}_2$.

For Π_n^D, since we have proven that the context bisimulation has a simpler characterization in terms of normal bisimulation (Theorem 1), (late) context bisimulation thus coincides with its early version; see [1, 2, 3, 26] for a reference. Unfortunately for Π_n^d, such a characteriztion is not available. Hence we do not know if the early version coincides with the late one. To examine the expressiveness concerning Π_n^d, we will use early context bisimulation instead of late context bisimulation, for the sake of technical simplicity. Recall that the early context bisimulation, which brings forward the universal quantifier over context in the output clause, is defined at the end of Section 2, and $\asymp$ denotes early context bisimilarity. The relationship between early and late bisimulation in Π_n^d remains an open issue.

4.2. Encoding from Π_n^D to Π_n^d

We prove $\Pi_1^D \sqsubseteq \Pi_1^d$, and this can be extended to the general case in a straightforward way. The encoding from Π_1^D to Π_1^d is a *homomorphism* on all the operations except for those concerning process parameterization, as shown below.

$$
\begin{array}{lcll}
[\![\langle X\rangle P]\!] & \stackrel{\text{def}}{=} & \langle x\rangle [\![P]\!]\{\bar{x}/X\} & (x \text{ is fresh in } P) \\
[\![P\langle A\rangle]\!] & \stackrel{\text{def}}{=} & (m)([\![P]\!]\langle m\rangle \mid !m.[\![A]\!]) & (m \text{ is fresh in } P \text{ and } A)
\end{array}
$$

Intuitively, a process-parameterized process $\langle X\rangle P$ is encoded by a name-parameterized process (on variable x), with the parameter X replaced by a trigger $\bar{x}$ (i.e., $\bar{x}0$), which is waiting to activate the potential process instantiating the variable X. This should be understood together with the encoding of application $P\langle A\rangle$. When a process-parameterized process P is instantiated by a higher-order process A to be $P\langle A\rangle$, it is encoded by instantiating it with a restricted (fresh) trigger name m, which is used to activate a certain number of copies of $[\![A]\!]$. The following example demonstrates how the encoding works. Suppose $P\langle Q\langle R\rangle\rangle \in \Pi_1^D$ in which $P \stackrel{\text{def}}{=} \langle X\rangle(X|X|a(Y).Y)$, $Q \stackrel{\text{def}}{=} \langle Z\rangle(Z)$ and $R \stackrel{\text{def}}{=} \bar{a}0$. Then $P\langle Q\langle R\rangle\rangle \stackrel{\bar{a}0}{\longrightarrow} \bar{a}0|a(Y).Y$. The encoding process behaves as below.

$$
\begin{array}{rcl}
[\![P\langle Q\langle R\rangle\rangle]\!] & = & (m)(\ \langle x\rangle(\bar{x} \mid \bar{x} \mid a(Y).Y)\langle m\rangle \mid !m.(n)((\langle z\rangle\bar{z})\langle n\rangle \mid !n.\bar{a}0)\ ) \\
& \stackrel{\tau}{\longrightarrow} & (m)(\ (\bar{m} \mid a(Y).Y) \mid (n)((\langle z\rangle\bar{z})\langle n\rangle \mid !n.\bar{a}0) \mid !m.(n)((\langle z\rangle\bar{z})\langle n\rangle \mid !n.\bar{a}0)\ ) \\
& \stackrel{\tau}{\longrightarrow} & (m)(\ (\bar{m} \mid a(Y).Y) \mid (n)(\bar{a}0 \mid !n.\bar{a}0) \mid !m.(n)((\langle z\rangle\bar{z})\langle n\rangle \mid !n.\bar{a}0)\ ) \\
& \stackrel{\bar{a}0}{\longrightarrow} & (m)(\ (\bar{m} \mid a(Y).Y) \mid (n)(!n.\bar{a}0) \mid !m.(n)((\langle z\rangle\bar{z})\langle n\rangle \mid !n.\bar{a}0)\ ) \\
& \asymp & [\![\bar{a}0 \mid a(Y).Y]\!]
\end{array}
$$

4.3. Properties of the encoding

According to the encoding schema, it is not hard to verify that our encoding satisfies the syntactic conditions, i.e., the compositionality and name invariance condition. In order to show the soundness property of the encoding, we start with establishing the forward and backward operational correspondence. Notice that $\dot{\asymp}$ is based on $\asymp$ as described in 4.1.

Lemma 6 (Forth)**.** *Let P be a Π_1^D process, the following statements are valid.*

1. *If $P \Longrightarrow P'$, then $[\![P]\!] \Longrightarrow T \asymp [\![P']\!]$ for some T'.*
2. *If $P \overset{a(A)}{\Longrightarrow} P'$ then $[\![P]\!] \overset{a([\![A]\!])}{\Longrightarrow} T \asymp [\![P']\!]$ for some T'.*
3. *If $P \overset{(\widetilde{c})\bar{a}A}{\Longrightarrow} P'$, then for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$ we have $[\![P]\!] \overset{(\widetilde{d})\bar{a}B}{\Longrightarrow} T$ for some $\widetilde{d}$, B and T, and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and $(\widetilde{c})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (\widetilde{d})(E[B] \mid T)$.*

The proof Lemma 6 can be done by structural induction and is given in Appendix B.1.

Lemma 7 (Back)**.** *Let P be a Π_1^D process, the following statements are valid.*

1. *If $[\![P]\!] \overset{\tau}{\Longrightarrow} T$, then there exists some P' such that $P \Longrightarrow P'$ and $T \asymp [\![P']\!]$*
2. *If $[\![P]\!] \overset{a(A)}{\Longrightarrow} T$ and there is some B such that $[\![B]\!] \equiv A$, then $P \overset{a(B)}{\Longrightarrow} P'$ for some P' such that $T \asymp [\![P']\!]$.*
3. *If $[\![P]\!] \overset{(\widetilde{c})\bar{a}A}{\Longrightarrow} T$, then for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$ there are some $\widetilde{d}$, B, and P' such that $P \overset{(\widetilde{d})\bar{a}B}{\Longrightarrow} P'$ and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and $(\widetilde{c})(T \mid E[A]) \asymp (\widetilde{d})([\![P']\!] \mid E[[\![B]\!]])$.*

The proof of Lemma 7 can be found in Appendix B.2. Notice that the third clause of Lemma 6 (also Lemma 7) depicts the output correspondence in a slightly different form than the one in the criteria. It would be possible to tune, for example, the operational semantics to make it comply with the form given in the criteria [38]. We do not do this because the current form is sufficient for our discussion. With Lemma 6 and Lemma 7 available, we can show the soundness property stated in Lemma 8, whose proof is placed in Appendix B.3.

Lemma 8. *Suppose P, Q are Π_1^D processes, $P \asymp Q$ implies $[\![P]\!] \dot{\asymp} [\![Q]\!]$.*

The last semantic condition is *divergence reflection*. A key observation is that our encoding does not introduce unnecessary divergence, because P is guarded in the replication operation $!\phi.P$. By structural induction it is straightforward to show the divergence reflection property. As a result, we arrive at the following theorem.

Theorem 9. *There is an encoding from Π_1^D to Π_1^d.*

We end this section with some remarks.

Remark 3. Completeness of an encoding is basically simpler than soundness, and actually can be established in a uniform way (or somewhat a derivable property based on the criteria). This is why completeness is not mentioned in the criteria (and our discussion either). It is however harmless to brief how it can be proven, One typical way is to prove by contraposition, and this appears to be a frequent method [6, 37]. A relatively direct way is to argue in a way similar to that for soundness, but this time much simpler, because intuitively one virtually only needs to deal with process parameterization, which admits a simpler bisimulation theory.

As mentioned, the encoding works for second-order processes, i.e., a term like $X\langle P\rangle$ cannot occur within an abstraction $\langle X\rangle Q$. The encoding breaks if arbitrary processes are allowed. For example, suppose $R \overset{\text{def}}{=} \langle X\rangle(X\langle \overline{a}0.0\rangle)$. Then

$$\begin{array}{rcl} [\![R]\!] & \equiv & \langle x\rangle([\![X\langle \overline{a}0.0\rangle]\!])\{\overline{x}/X\} \\ & \equiv & \langle x\rangle((m)([\![X]\!]\langle m\rangle \mid !m.[\![\overline{a}0.0]\!]))\{\overline{x}/X\} \\ & \equiv & \langle x\rangle((m)(\overline{x}\langle m\rangle \mid !m.\overline{a}0.0)) \end{array}$$

Obviously the subterm $\overline{x}\langle m\rangle$ is not well-formed in Π^d. In future work, it would be interesting to find a general approach (which might need to look into the intrinsic information carried by X) of extending the current encoding strategy to the general higher-order setting.

Taking into account related work in the field, the current knowledge on expressiveness over higher order calculi is now expanded with the result from this section that Π_n^D is encodable in Π_n^d. From [1, 2] we know that both Π_n^D and Π_n^d can be embedded into π (the name-passing pi-calculus [4]), and from [7] we know Π_1^D is not encodable in Π. As syntactically Π_n^D subsumes Π_1^D, the same result in [7] carries over to the setting here. That said, it would be interesting to know whether Π_n^D is strongly encodable in Π_n^d. If so, then we can deduce that Π_n^d is more powerful than Π. Yet this remains unknown.

5. Conclusion and future work

This paper deepens the study of higher-order processes with parameterization. The results provide some insight into the bisimulation theory and expressiveness in presence of parameterization within the higher-order paradigm. We believe our results can be adapted to the calculi using abstractions/concretions as in [3], essentially treating an input $a(X).P$ as comprised by a channel a plus an abstraction $\langle X\rangle P$, and similarly for output. This is because it is more a matter of the choice of syntactical formalism than a semantical difference (so the bisimulation relations would essentially be the same). Below we summarize and highlight some future work.

On bisimulation theory. We show that when extended with process parameterization, higher-order processes enjoys the simpler characterization of context bisimulation in terms of normal bisimulation. In contrast, we do not know if this method of normal bisimulation is applicable in presence of name parameterization. This might indicate some essential difference between the two kinds of parameterization. Some future work includes:

- Finding some novel proof technique for the context bisimulation in Π^d_n. To some extent, Π^d_n is more useful than Π^D_n because it has name-handling primitives, though it is still a higher-order language (i.e., no name-passing). Such a proof technique would be particularly useful when studying related topics, e.g., expressiveness and applications concerning modelling and verification.

- Extending the study to other forms of bisimulation in more general higher-order models, for example, environmental bisimulation [14], which is shown to coincide with canonical bisimulations in various higher-order models. In such a setting, one may search for some general proof technique (e.g., up-to certain 'normal' feature) of different nature.

On expressiveness. We show that Π^D_n can be encoded by Π^d_n, and the encoding is proven to meet some well-known criteria in the field. The encoding somewhat stresses the atomic position of the capacity of manipulating names in dealing with higher-order processes, though both kinds of parameterization can be useful in practice. Some future work is as below.

- One question is about the encodability of π in Π^d_n (another relevant one is encoding π in Π^D_n). In light of related work in the field [1, 9, 7, 39], we have the following strategy (being homomorphism on the rest operations).

$$\begin{array}{rcll} [\![a(x).P]\!] & \stackrel{\text{def}}{=} & a(Y).Y\langle\langle x\rangle[\![P]\!]\rangle & (Y \text{ is fresh}) \\ [\![\overline{a}b.Q]\!] & \stackrel{\text{def}}{=} & \overline{a}[\langle Z\rangle(Z\langle b\rangle)].[\![Q]\!] & (Z \text{ is fresh}) \end{array}$$

  The encoding above utilizes both name parameterization and process parameterization, and appears to be sound. That is, the encoding of output 'transmits' the name to be sent (i.e., b) by means of a process parameterization (i.e., $\langle Z\rangle(Z\langle b\rangle)$) that, upon being received by the encoding of the input, is instantiated by a name-parameterized term (i.e., $\langle x\rangle[\![P]\!]$), which can then apply b on x in the encoding of P, thus achieving 'name-passing'. The more detailed properties of this encoding is among our ongoing work. That said, it is still not clear whether the encoding can be adapted (or rewritten) into one that only uses name parameterization. This is worthwhile for further investigation.

- A relevant conjecture is that Π^d_n is not encodable in Π^D_n, based on which the nonexistence of encodings from π to Π^D_n follows as a corollary by contraposition. Intuitively, a critical point is that with static binding, a context of Π^D_n has no way to dynamically form any channel without resorting to some global names previously agreed upon. This conjecture is worthy of further examination.

- In this paper, the languages in which we study the expressiveness of parameterization are small and 'abstract'; it would be interesting to explore parameterization in some 'less abstract' languages (in which the trigger technique might fail), such as those accommodating asynchronous communication [40, 41], reversibility and compensation [42], and explicit locations [43] etc.. These more practical features are increasingly common in modern distributed (concurrent) systems.

Acknowledgement The authors are particularly grateful to Alan Schmitt for his very helpful comments and constructive suggestions (e.g., the discussion on expressiveness in the last section finds source in an instructive encoding suggested by him) during the revision of this paper. We also thank Davide Sangiorgi and the anonymous referees for their useful comments.

References

[1] D. Sangiorgi, Expressing mobility in process algebras: First-order and higher-order paradigms, Phd thesis, University of Edinburgh (1992).

[2] D. Sangiorgi, From π-calculus to higher-order π-calculus—and back, in: Proceedings of Theory and Practice of Software Development (TAPSOFT '93), Vol. 668 of LNCS, Springer Verlag, 1992, pp. 151–166. doi:10.1007/3-540-56610-4_62.

[3] D. Sangiorgi, Bisimulation for higher-order process calculi, Information and Computation 131(2) (1996) 141–178. doi:10.1006/inco.1996.0096.

[4] R. Milner, J. Parrow, D. Walker, A calculus of mobile processes (parts i and ii), Information and Computation 100(1) (1992) 1–77. doi:10.1016/0890-5401(92)90008-4,10.1016/0890-5401(92)90009-5.

[5] D. Sangiorgi, D. Walker, The Pi-calculus: a Theory of Mobile Processes, Cambridge Universtity Press, 2001.

[6] J. A. Perez, Higher-order concurrency: Expressiveness and decidability results, Ph.D. thesis, Department of Computer Science, University of Bologna (2010).

[7] I. Lanese, J. A. Pérez, D. Sangiorgi, A. Schmitt, On the expressiveness of polyadic and synchronous communication in higher-order process calculi, in: Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2010), LNCS, Springer Verlag, 2010, pp. 442–453. doi:10.1007/978-3-642-14162-1_37.

[8] B. Thomsen, Calculi for higher order communicating systems, Phd thesis, Department of Computing, Imperial College (1990).

[9] B. Thomsen, Plain CHOCS, a second generation calculus for higher-order processes, Acta Informatica 30(1) (1993) 1–59. doi:10.1007/BF01200262.

[10] J.-L. Vivas, M. Dam, From higher-order pi-calculus to pi-calculus in the presence of static operators, in: Proceedings of the 9th International Conference on Concurrency Theory (CONCUR 1998), Vol. 1466 of LNCS, Springer Verlag, 1998, pp. 115–130. doi:10.1007/BFb0055619.

[11] A. Jeffrey, J. Rathke, Contextual equivalence for higher-order pi-calculus revisited, Logical Methods in Computer Science 1(1:4). doi:10.2168/LMCS-1(1:4)2005.

[12] Z. Cao, More on bisimulations for higher order π-calculus, in: Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structures (FOSSACS2006), Vol. 3921 of LNCS, Springer Verlag, 2006, pp. 63–78. doi:10.1007/11690634_5.

[13] I. Lanese, J. Perez, D. Sangiorgi, A. Schmitt, On the expressiveness and decidability of higher-order process calculi, in: Proceedings of the 23rd Annual IEEE Symposium on Logic in Computer Science (LICS 2008), IEEE Computer Society, 2008, pp. 145–155, journal version in [16]. doi:10.1109/LICS.2008.8.

[14] D. Sangiorgi, N. Kobayashi, E. Sumii, Environmental bisimulations for higher-order languages, ACM Transactions on Programming Languages and Systems 33(1) (2011) 5. doi:10.1145/1889997.1890002.

[15] U. D. Lago, S. Martini, D. Sangiorgi, Light logics and higher-order processes, in: Proceedings of Workshop on Expressiveness in Concurrency 2010 (EXPRESS2010), Vol. 41 EPTCS, 2010, pp. 46–60. doi:10.1017/S096012951400031O.

[16] I. Lanese, J. A. Pérez, D. Sangiorgi, A. Schmitt, On the expressiveness and decidability of higher-order process calculi, Information and Computation 209(2) (2011) 198–226. doi:10.1016/j.ic.2010.10.001.

[17] X. Xu, Expressing first-order π-calculus in higher-order calculus of communicating systems, Journal of Computer Science and Technology 24(1) (2009) 122–137. doi:10.1007/s11390-009-9210-y.

[18] R. Meyer, V. Khomenko, T. Strazny, A practical approach to verification of mobile systems using net unfoldings, in: Proceedings of the 29th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency (ATPN 2008), Vol. 5062 of LNCS, 2008, pp. 327–347. doi:10.1007/978-3-540-68746-7_22.

[19] C. D. Giusto, J. A. Pérez, G. Zavattaro, On the expressiveness of forwarding in higher-order communication, in: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC 2009), Vol. LNCS 5684, Springer Verlag, 2009, pp. 155 155 155–169. doi:10.1007/978-3-642-03466-4_10.

[20] L. Parrow, Expressiveness of process algebras, Electronic Notes in Theoretical Computer Science 209 (2008) 173–186. doi:10.1016/j.entcs.2008.04.011.

[21] R. M. Amadio, L. Leth, B. Thomsen, From a concurrent lambda-calculus to the pi-calculus, in: Proceedings of the 10th International Symposium Fundamentals of Computation Theory (FCT1995), Vol. 965 of LNCS, Springer Verlag, 1995, pp. 106–115.

[22] M. Bundgaard, J. C. Godskesen, B. Haagensen, H. Hüttel, Decidable fragments of a higher order calculus with locations, in: Proceedings of the 15th Workshop on Expressiveness in Concurrency (EXPRESS 2008), Vol. 242(1) of Electronic Notes in Theoretical Computer Science, Elsevier, 2008, pp. 113–138. doi:10.1016/j.entcs.2009.06.016.

[23] X. Xu, On bisimulation theory in linear higher-order pi-calculus, Transactions on Petri Nets and Other Models of Concurrency III 5800 (2009) 244–274. doi:10.1007/978-3-642-04856-2_10.

[24] J.-L. Vivas, N. Yoshida, Dynamic channel screening in the higher order pi-calculus, in: Proceedings of Foundations of Wide Area Network Computing (F-WAN, ICALP 2002 Satellite Workshop), Vol. 66(3) of Electronic Notes in Theoretical Computer Science, Elsevier, 2002, pp. 170–184. doi:10.1.1.16.7470.

[25] J.-L. Vivas, Dynamic binding of names in calculi for mobile processes, Ph.D. thesis, KTH-Royal Institute of Technology (2001).

[26] D. Sangiorgi, Pi-calculus, internal mobility and agent-passing calculi, Theoretical Computer Science 167(1-2) (1996) 235–274. doi:10.1016/0304-3975(96)00075-8.

[27] M. Bundgaard, T. Hildebrandt, J. C. Godskesen, A CPS encoding of name-passing in higher-order mobile embedded resources, Theoretical Computer Science 356 (2006) 422–439. doi:10.1016/j.tcs.2006.02.006.

[28] M. Bundgaard, J. C. Godskesen, T. Hildebrandt, Encoding the pi-calculus in higher-order calculi, Tech. Rep. TR-2008-106, IT University of Copenhagen (2008).
[29] R. Milner, Communication and Concurrency, Prentice Hall, 1989.
[30] X. Xu, Distinguishing and relating higher-order and first-order processes by expressiveness, Acta Informatica 49(7-8) (2012) 445–484. doi:10.1007/s00236-012-0168-9.
[31] Q. Yin, H. Long, Process passing calculus, revisited, Journal of Shanghai Jiaotong University (Science) 18 (1) (2013) 29–36. doi:10.1007/s12204-013-1365-6.
[32] R. Milner, Functions as processes, Mathematical Structures in Computer Science 2(2) (1992) 119–141. doi:10.1017/S0960129500001407.
[33] D. Sangiorgi, On the bisimulation proof method, Mathematical Structures in Computer Science 8(6) (1998) 447–479. doi:10.1017/S0960129598002527.
[34] S. Lenglet, A. Schmitt, J.-B. Stefani, Normal bisimulations in calculi with passivation, in: Proceedings of the 12th International Conference on Foundations of Software Science and Computational Structures (FOSSACS 2009), Vol. 5504 of LNCS, Springer Verlag, 2009, pp. 257–271. doi:10.1007/978-3-642-00596-1_19.
[35] S. Lenglet, A. Schmitt, J.-B. Stefani, Characterizing contextual equivalence in calculi with passivation, Information and Computation 209 (2011) 1390–1433. doi:10.1016/j.ic.2011.08.002.
[36] L. Cardelli, A. Gordon, Mobile ambients, Theoretical Computer Science 240 (2000) 177–213. doi:10.1016/S0304-3975(99)00231-5.
[37] D. Gorla, Towards a unified approach to encodability and separation results for process calculi, in: Proceedings of the 19th International Conference on Concurrency Theory (CONCUR 2008), Vol. 5201 of LNCS, Springer Verlag, 2008, pp. 492–507. doi:10.1007/978-3-540-85361-9_38.
[38] D. Gorla, On the relative expressive power of calculi for mobility, Electronic Notes in Theoretical Computer Science 249 (2009) 269–286. doi:10.1016/j.entcs.2009.07.094.
[39] X. Xu, Q. Yin, H. Long, On the computation power of name parameterization in higher-order processes, in: Proceedings of the 8th Interaction and Concurrency Experience (ICE 2015), Vol. 189 of EPTCS, 2015, pp. 114–127. doi:10.4204/EPTCS.189.10.
[40] K. Honda, M. Tokoro, An object calculus for asynchronous communication, in: Proceedings of the European Conference on Object-Oriented Programming (ECOOP 1991), Vol. 512, Springer Verlag, 1991, pp. 133–147. doi:10.1007/BFb0057019.
[41] C. Palamidessi, Comparing the expressive power of the synchronous and the asynchronous pi-calculus, Mathematical Structures in Computer Science 13 (2003) 685–719. doi:10.1017/S0960129503004043.
[42] I. Lanese, M. Lienhardt, C. A. Mezzina, A. Schmitt, J.-B. Stefani, Concurrent flexible reversibility, in: Proceedings of the 22nd European Symposium on Programming (ESOP 2013), Vol. 7792 of LNCS, Springer Verlag, 2013, pp. 370–390. doi:10.1007/978-3-642-37036-6_21.
[43] M. Hennessy, A Distributed Pi-Calculus, Cambridge University Press, 2007.
[44] R. De Nicola, U. Montanari, F. Vaandrager, Back and forth bisimulations, in: Proceedings of Theories of Concurrency: Unification and Extension (CONCUR 1990), Vol. 458 of LNCS, Springer Verlag, 1990, pp. 152–165. doi:10.1007/BFb0039058.
[45] D. Sangiorgi, R. Milner, The problem of weak bisimulation up-to, in: Proceedings of the 3rd International Conference on Concurrency (CONCUR 1992), Vol. 630 of LNCS, Springer Verlag, 1992, pp. 32–46. doi:10.1007/BFb0084781.

Appendix A. Proof of section 3

Appendix A.1. Proof of Lemma 2

Proof. (1). Let $P_1 \stackrel{\text{def}}{=} (m)(\alpha.E[Tr_m] \,|\, !m(Z).A\langle Z\rangle)$ and $Q_1 \stackrel{\text{def}}{=} \alpha.(m)(E[Tr_m] \,|\, !m(Z).A\langle Z\rangle)$. We define $\mathcal{R}_1 \stackrel{\text{def}}{=} \{(P_1, Q_1)\} \cup \mathcal{I}d$ in which $\mathcal{I}d \stackrel{\text{def}}{=} \{(P, P)\}$ is the identity relation. We show $\mathcal{R}_1$ is a strong context bisimulation up-to $\equiv$. Obviously the only action from P_1 is α. We make a case analysis on α (the situation of the other way is similar).

- α is τ. Then $P_1 \xrightarrow{\tau} (m)(E[Tr_m] \,|\, !m(Z).A\langle Z\rangle)$ is matched by $Q_1 \xrightarrow{\tau} (m)(E[Tr_m] \,|\, !m(Z).A\langle Z\rangle)$.
- α is $c(Y)$ for some c. Then $P_1 \xrightarrow{c(R)} (m)(E[Tr_m]\{R/Y\} \,|\, !m(Z).A\langle Z\rangle)$ is matched by $Q_1 \xrightarrow{c(R)} ((m)(E[Tr_m] \,|\, !m(Z).A\langle Z\rangle))\{R/Y\} \equiv (m)(E[Tr_m]\{R/Y\} \,|\, !m(Z).A\langle Z\rangle)$, since $bv(\alpha) \notin fv(A)$.
- α is $\overline{b}B$ for some b and B. Then $P_1 \xrightarrow{\overline{b}B} (m)(E[Tr_m] \,|\, !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} R_1$ is matched by $Q_1 \xrightarrow{\overline{b}B} R_1$, and clearly we have $(E[B] \,|\, R_1)\,\mathcal{R}\,(E[B] \,|\, R_1)$ for every $E[X]$.

(2). Let $P_2 \stackrel{\text{def}}{=} (m)(\overline{a}B_1.E_1[Tr_m] \,|\, !m(Z).A\langle Z\rangle)$ and $Q_2 \stackrel{\text{def}}{=} (m)(\overline{a}B_2.E_1[Tr_m] \,|\, !m(Z).A\langle Z\rangle)$. We define

$$\mathcal{R}_2 \stackrel{\text{def}}{=} \left\{ \Big((m)(G[B_1] \,|\, !m(Z).A\langle Z\rangle), (m)(G[B_2] \,|\, !m(Z).A\langle Z\rangle) \Big) \;\middle|\; \text{for some } G[\cdot], E_2, \text{and } A \right\} \cup \approx \tag{A.1}$$

Notice that E_2 appears in B_1, B_2. Obviously $(P_2, Q_2) \in \mathcal{R}_2$. We show $\mathcal{R}_2$ is a strong context bisimulation up-to $\sim$ by induction on G. For convenience, we denote the element in $\mathcal{R}_2$ as: $P_3 \stackrel{\text{def}}{=} (m)(G[B_1] \,|\, !m(Z).A\langle Z\rangle)$ and $Q_3 \stackrel{\text{def}}{=} (m)(G[B_2] \,|\, !m(Z).A\langle Z\rangle)$.

- $G[\cdot]$ is $[\cdot]$. In this case we have P_3 and Q_3 as below.

$$\begin{array}{lcl} P_3 & \equiv & (m)(B_1 \mid !m(Z).A\langle Z\rangle) \\ Q_3 & \equiv & (m)(B_2 \mid !m(Z).A\langle Z\rangle) \end{array}$$

  Since $m \notin fn(E_2, A)$, we have $Q_3 \sim B_2 \equiv P_3$, which closes this case.

- $G[\cdot]$ is $b(Y).G_1[\cdot]$. Notice that $Y \notin fv(A)$ (A is closed). In this case we have P_3 and Q_3 as below.

$$\begin{array}{lcl} P_3 & \equiv & (m)(b(Y).G_1[B_1] \mid !m(Z).A\langle Z\rangle) \\ Q_3 & \equiv & (m)(b(Y).G_1[B_2] \mid !m(Z).A\langle Z\rangle) \end{array}$$

  Suppose P_3 makes an action (the case Q_3 does is similar). Obviously the only action from P_3 is $b(A')$ (from $G[B_1]$), that is,

$$\begin{array}{l} P_3 \xrightarrow{b(A')} (m)((G_1[B_1])\{A'/Y\} \mid !m(Z).A\langle Z\rangle) \equiv (m)((G'_1[B'_1]) \mid !m(Z).A\langle Z\rangle) \\ \text{where } G'_1 \stackrel{\text{def}}{=} G_1\{A'/Y\},\ B'_1 \stackrel{\text{def}}{=} B_1\{A'/Y\} \equiv (E_2[Tr_m])\{A'/Y\} \equiv E'_2[Tr_m] \text{ for some } E'_2[X] \end{array}$$

  Then Q_3 corresponds with the following move

$$\begin{array}{l} Q_3 \xrightarrow{b(A')} (m)((G_1[B_2])\{A'/Y\} \mid !m(Z).A\langle Z\rangle) \equiv (m)((G'_1[B'_2]) \mid !m(Z).A\langle Z\rangle) \\ \text{where } B'_2 \stackrel{\text{def}}{=} B_2\{A'/Y\} \equiv (m)(E'_2[Tr_m] \mid !m(Z).A\langle Z\rangle) \end{array}$$

  Then this case is closed by taking G and E_2 in (A.1) respectively as G'_1 and E'_2.

- $G[\cdot]$ is $\overline{b}[G_1[\cdot]].T$. In this case we have P_3 and Q_3 as below.

$$\begin{array}{lcl} P_3 & \equiv & (m)(\overline{b}[G_1[B_1]].T \mid !m(Z).A\langle Z\rangle) \\ Q_3 & \equiv & (m)(\overline{b}[G_1[B_2]].T \mid !m(Z).A\langle Z\rangle) \end{array}$$

  Suppose P_3 makes an action (the case Q_3 does is similar). Obviously the only action from P_3 is $(m)\overline{b}[G_1[B_1]]$ (from $G[B_1]$), that is,

$$P_3 \xrightarrow{(m)\overline{b}\left[G_1[B_1]\right]} T \mid !m(Z).A\langle Z\rangle$$

  Then Q_3 matches by making the following move

$$Q_3 \xrightarrow{\overline{b}\left[G_1[B_2]\right]} (m)(T \mid !m(Z).A\langle Z\rangle)$$

  Now for every $F[X]$ s.t. $\{m\} \cap fn(F) = \emptyset$ and $bn(F) \cap (fn(B_1) \cup fn(B_2)) = \emptyset$, we want to have

$$(m)(F[G_1[B_1]] \mid T \mid !m(Z).A\langle Z\rangle)\ \mathcal{R}_2\ F[G_1[B_2]] \mid (m)(T \mid !m(Z).A\langle Z\rangle) \sim (m)(F[G_1[B_2]] \mid T \mid !m(Z).A\langle Z\rangle)$$

  where the $\sim$ may need some α-conversion in G_1. Then this case is closed by taking G as $F[G_1[\cdot]] \mid T$ in (A.1).

- $G[\cdot]$ is $\overline{b}A.G_1[\cdot]$. This case is similar to the last one.

- $G[\cdot]$ is $G_1[\cdot] \mid T$. In this case we have P_3 and Q_3 as below.

$$\begin{array}{lcl} P_3 & \equiv & (m)(G_1[B_1] \mid T \mid !m(Z).A\langle Z\rangle) \\ Q_3 & \equiv & (m)(G_1[B_2] \mid T \mid !m(Z).A\langle Z\rangle) \end{array}$$

  Suppose P_3 does an action α (the other way around is similar). There are several cases about where α comes from: 1) T; 2) G_1; 3) B_1; 4) interaction between G_1, B_1, T and $!m(Z).A\langle Z\rangle$(5 subcases). We take two cases. The rest is similar (or simpler).

– First, the case α is from interaction between B_1 and T (not on m since T is not aware of m). We take the subcase B_1 makes an output and T makes an input. That is

$$B_1 \xrightarrow{(\widetilde{c})\overline{d}A'} B_1' \qquad T \xrightarrow{d(A')} T' \qquad P_3 \xrightarrow{\tau} \sim (m)((\widetilde{c})(G_1[B_1'] \mid T') \mid !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} P_3'$$

Since d is not m, the action by B_1 must come from E_2, i.e., $E_2[Tr_m] \xrightarrow{(\widetilde{c})\overline{d}A'} E_2'[Tr_m] \equiv B_1'$. So $B_2 \xrightarrow{(\widetilde{c})\overline{d}A'} B_2' \equiv (m)(E_2'[Tr_m] \mid !m(Z).A\langle Z\rangle)$, and

$$Q_3 \xrightarrow{\tau} \sim (m)((\widetilde{c})(G_1[B_2'] \mid T') \mid !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} Q_3'$$

Now we have $(P_3', Q_3') \in \mathcal{R}_2$ by treating G and E_2 in (A.1) respectively as $(\widetilde{c})(G_1[\cdot] \mid T')$ and E_2'.

– Second, the case α is from interaction between B_1 and $!m(Z).A\langle Z\rangle$. That is

$$\begin{array}{l} B_1 \equiv E_2[Tr_m] \xrightarrow{(\widetilde{c})\overline{m}A'} B_1' \qquad !m(Z).A\langle Z\rangle \xrightarrow{m(A')} \sim A\langle A'\rangle \mid !m(Z).A\langle Z\rangle \\ P_3 \xrightarrow{\tau} \sim (m)((\widetilde{c})(G_1[B_1'] \mid A\langle A'\rangle) \mid T \mid !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} P_3' \end{array}$$

Since B_1 is at a firable position (i.e., not underneath a prefix), we have (assuming no name capture concerning $\widetilde{c}$)

$$\begin{array}{c} P_3' \sim (m)(G_1[B_1''] \mid T \mid !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} P_3'' \\ \text{where } B_1'' \stackrel{\text{def}}{=} (\widetilde{c})(B_1' \mid A\langle A'\rangle) \end{array}$$

Moreover, because Tr_m has been consumed in B_1, we can write B_1' as $E_2'[Tr_m]$ for some $E_2'[X]$ (in which $X \notin fv(E_2')$). Thus

$$B_1'' \equiv E_2''[Tr_m] \qquad \text{where} \qquad E_2''[X] \stackrel{\text{def}}{=} (\widetilde{c})(E_2'[X] \mid A\langle A'\rangle)$$

Then

$$\begin{array}{l} B_2 \equiv (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) \xrightarrow{\tau} \sim (m)((\widetilde{c})(B_1' \mid A\langle A'\rangle) \mid !m(Z).A\langle Z\rangle) \sim (m)(B_1'' \mid !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} B_2'' \\ Q_3 \xrightarrow{\tau} (m)(G_1[B_2''] \mid T \mid !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} Q_3'' \end{array}$$

In summary,

$$\begin{array}{ll} P_3 \xrightarrow{\tau} \sim (m)(G_1[B_1''] \mid T \mid !m(Z).A\langle Z\rangle) \equiv P_3'' & \text{where} \quad B_1'' \equiv E_2''[Tr_m] \\ Q_3 \xrightarrow{\tau} \sim (m)(G_1[B_2''] \mid T \mid !m(Z).A\langle Z\rangle) \equiv Q_3'' & \text{where} \quad B_2'' \equiv (m)(E_2''[Tr_m] \mid !m(Z).A\langle Z\rangle) \end{array}$$

Now we have $(P_3'', Q_3'') \in \mathcal{R}_2$ by treating G and E_2 in (A.1) respectively as $G_1[\cdot] \mid T$ and E_2''.

- $G[\cdot]$ is $(c)G_1[\cdot]$. In this case we have P_3 and Q_3 as below.

$$\begin{array}{lcl} P_3 & \equiv & (m)((c)G_1[B_1] \mid !m(Z).A\langle Z\rangle) \\ Q_3 & \equiv & (m)((c)G_1[B_2] \mid !m(Z).A\langle Z\rangle) \end{array}$$

  Suppose P_3 does an action α (the other way around is similar). Like the previous case, there are several cases about where α is originated: 1) G_1; 2) B_1; 3) interaction between G_1, B_1 and $!m(Z).A\langle Z\rangle$(3 subcases). The discussion can be conducted in a way similar to the previous case (only notice the design of G in (A.1) may involve the restriction on c).

- $G[\cdot]$ is $G_1[\cdot]\langle B\rangle$. This means G_1 is of the form $\langle Y\rangle G_2[\cdot]$, i.e., $G[\cdot] \equiv (\langle Y\rangle G_2[\cdot])\langle B\rangle$. So $G[\cdot] \equiv G_2\{B/Y\}[\cdot]$. Then this case eventually falls into one of the other cases.

- $G[\cdot]$ is $\langle Y\rangle G'[\cdot]$. This case is trivial because P_3 and Q_3 do not exhibit any action.

(3). Let $P_3 \stackrel{\text{def}}{=} (m)(E_1[Tr_m] \mid E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)$ and $Q_3 \stackrel{\text{def}}{=} (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \mid (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)$. This follows from (2) in this lemma, since $(P_3, Q_3) \in \mathcal{R}_2$ in (A.1) by taking G as $E_1[Tr_m] \mid [\cdot]$ (actually we prove a more general result in there).

□

Appendix A.2. Proof of Lemma 3

Proof. Suppose $B \equiv \langle Y \rangle T$, it amounts to proving

$$\begin{array}{rl} & P_1 \stackrel{\text{def}}{=} T\{T_1/Y\} \approx (m)(T\{T_2/Y\} \mid !m(Z).A\langle Z\rangle) \stackrel{\text{def}}{=} Q_1 \\ \text{where} & T_1 \stackrel{\text{def}}{=} (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \text{ and } T_2 \stackrel{\text{def}}{=} E_1[Tr_m] \end{array}$$

To this end, we define

$$\mathcal{R}_3 \stackrel{\text{def}}{=} \left\{ \Big(G[T_1], (m)(G[T_2] \mid !m(Z).A\langle Z\rangle) \Big) \;\middle|\; \text{for some } G[\cdot] \text{ and } E_1, m \notin fn(G, E_1, A) \right\} \cup \approx \tag{A.2}$$

Notice that E_1 appears in T_1, T_2. Obviously $(P_1, Q_1) \in \mathcal{R}_3$. We can show $\mathcal{R}_3$ is a strong context bisimulation up-to $\sim$ by induction on G. We save the details because they are very similar to that of Lemma 2.

□

Appendix A.3. Proof of Theorem 4: Factorization

Proof of Theorem 4: Factorization. The proof is by induction on E. The cases 1,2,3 are the base cases.

1. E is 0 or E is Y and $Y \neq X$. These cases are trivial.
2. E is X. Notice that this time $E[Tr_m]$ is $Tr_m \equiv \langle Z \rangle \overline{m} Z \equiv \langle Y \rangle \overline{m} Y$ (up-to alpha-conversion), so the second statement of this theorem applies and the two terms of interest are

   $$A \quad \text{and} \quad \langle Y \rangle((m)(\overline{m}Y \mid !m(Z).A\langle Z\rangle)$$

   Suppose $A \equiv \langle Z' \rangle F$, then it is easy to verify that for each B one has

   $$F\langle B \rangle \approx (m)(\overline{m}B \mid !m(Z).F\{Z/Z'\}$$

   which completes this case.
3. E is $X\langle E_1 \rangle$. This can be proven by showing the following relation $\mathcal{R}$ is a context bisimulation up-to $\sim$ (this technique and similar ones are standard [3, 29, 5] and we omit the definition here and in what follows; some related explanation can be found at the beginning of the proof for Theorem 1).

   $$\mathcal{R} \stackrel{\text{def}}{=} \{(A\langle E_1 \rangle, (m)(Tr_m\langle E_1 \rangle \mid !m(Z).A\langle Z\rangle) \;\big|\; m \text{ is fresh w.r.t. } A \text{ and } E_1\} \cup \approx$$

   Let $(A\langle E_1 \rangle, (m)(Tr_m\langle E_1 \rangle \mid !m(Z).A\langle Z\rangle) \in \mathcal{R}$ and $A \equiv \langle Z' \rangle T$; so the pair is actually

   $$(T\{E_1/Z'\}, (m)(\overline{m}E_1 \mid !m(Z).T\{Z/Z'\})$$

   There are mainly two cases to consider.

   - $(m)(\overline{m}E_1 \mid !m(Z).T\{Z/Z'\}) \xrightarrow{\alpha} T'$. Then α must be τ, and thus

     $$T' \equiv (m)(T\{E_1/Z'\} \mid !m(Z).T\{Z/Z'\})$$

     By $T\{E_1/Z'\} \Rightarrow T\{E_1/Z'\}$ (null transition), since m is fresh, it can be easily seen that,

     $$T\{E_1/Z'\} \sim T\{E_1/Z'\}\, \mathcal{R}\, T\{E_1/Z'\} \sim T'$$

   - $T\{E_1/Z'\} \xrightarrow{\alpha} T_1$. Then this is simulated by

     $$\begin{array}{ll} & (m)(\overline{m}E_1 \mid !m(Z).T\{Z/Z'\}) \\ \xrightarrow{\tau} & (m)(T\{E_1/Z'\} \mid !m(Z).T\{Z/Z'\}) \\ \xrightarrow{\alpha} & (m)(T_1 \mid !m(Z).T\{Z/Z'\}) \stackrel{\text{def}}{=} T_2 \end{array}$$

     So it holds in a straightforward way that

     $$T_1 \sim T_1\, \mathcal{R}\, T_1 \sim T_2$$

The following cases 4,5,6,7,8,9 are cases involving the induction hypothesis (ind. hyp. for short). Notice that we will only consider the case when statement (1) in the theorem applies, and the case for (2) can be dealt with similarly. Also the congruence property would be used implicitly.

4. E is $\langle Y\rangle E_1$. Then we have by ind. hyp.

$$\begin{array}{rl} & E[A] \equiv \langle Y\rangle E_1[A] \\ \approx & \langle Y\rangle((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)) \end{array}$$

   To conclude this case, we have to show

$$\langle Y\rangle((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)) \approx \langle Y\rangle((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle))$$

   This is immediate and the right-hand-side is exactly the second claim (2) of this theorem.

5. E is $\overline{a}E_2.E_1$. Then we have

$$\begin{array}{rll} & E[A] \equiv \overline{a}E_2[A].E_1[A] & \\ \approx & \overline{a}[(m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)].((m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)) & \text{(ind. hyp.)} \\ \approx & (m)(\overline{a}[(m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle)].E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(Lemma 2(1))} \\ \approx & (m)(\overline{a}E_2[Tr_m].E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(Lemma 2(2))} \end{array}$$

   The last equation gives exactly what we expect.

6. E is $a(Y).E_1$. This is similar (and easier) than the previous case, and straightforward from Lemma 2(1).

7. E is $E_1 \mid E_2$. Then we have

$$\begin{array}{rll} & E[A] \equiv E_1[A] \mid E_2[A] & \\ \approx & (m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle) \mid (m)(E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(ind. hyp.)} \\ \approx & (m)(E_1[Tr_m] \mid E_2[Tr_m] \mid !m(Z).A\langle Z\rangle) & \text{(Lemma 2(3))} \end{array}$$

   The last equation completes this case.

8. E is $(c)E_1$. This is similar to the previous case.

9. E is $E_2\langle E_1\rangle$. This case can be reduced to the case E is $Y\langle E_1\rangle$ and $Y \neq X$, because if E_2 is not a variable (i.e., an abstraction) or is X, then it can be handled in a way that falls into one of the previous cases (up-to structural congruence). So we have

$$\begin{array}{rll} & E[A] & \\ \equiv & Y\langle E_1[A]\rangle & \\ \approx & Y\langle(m)(E[Tr_m] \mid !m(Z).A\langle Z\rangle)\rangle \stackrel{\text{def}}{=} T & \text{(ind. hyp.)} \end{array}$$

   We want to show that $T \approx (m)(Y\langle E_1[Tr_m]\rangle \mid !m(Z).A\langle Z\rangle)$ i.e., for every B,

$$B\langle(m)(E_1[Tr_m] \mid !m(Z).A\langle Z\rangle)\rangle \approx (m)(B\langle E_1[Tr_m]\rangle \mid !m(Z).A\langle Z\rangle)$$

   This is immediate from Lemma 3.

The proof is now completed.

□

Appendix B. Proof of section 4

Appendix B.1. Proof of Lemma 6 (Forward Operational Correspondence)

The proofs to the following lemmas are routine structural induction.

Lemma 10. *If $P \xrightarrow{a(A)} P_1$, then there is some $F[X]$ such that $P_1 = F[A]$ and for all B, $P \xrightarrow{a(B)} F[B]$.*

Lemma 11. *Let P be a Π_1^D process and $[\![P]\!] \Rightarrow \xrightarrow{(\widetilde{c})\bar{a}B} T$, then there is some $A \in \Pi_1^D$ such that $[\![A]\!] = B$.*

To prove Lemma 6, we first focus on the strong version of transition correspondence, as stated in the following lemma, since moving from strong version to weak version in Lemma 6 involves somewhat regular arguments (see [1] for a reference).

Lemma 12 (Strong Forward Operation Correspondence)**.** *Let P be a Π_1^D process, the following statements are valid*

1. *If $P \xrightarrow{\tau} P'$, then $[\![P]\!] \Rightarrow \xrightarrow{\tau} \Rightarrow T \asymp [\![P']\!]$ for some T.*
2. *If $P \xrightarrow{a(A)} P'$ then $[\![P]\!] \Rightarrow \xrightarrow{a([\![A]\!])} \Rightarrow T \asymp [\![P']\!]$ for some T.*
3. *If $P \xrightarrow{(\widetilde{c})\bar{a}A} P'$, then for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$ there are some $\widetilde{d}$, B and T such that $[\![P]\!] \Rightarrow \xrightarrow{(\widetilde{d})\bar{a}B} \Rightarrow T$ and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and $(\widetilde{c})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (\widetilde{d})(E[B] \mid T)$.*

Proof. The proof proceeds by induction on the structure of P. We consider the following cases.

- Case 1. P is of the form $a(X).P_1$. P can only do an input action such that $P \xrightarrow{a(A)} P_1\{A/X\}$, then we have

$$[\![P]\!] \equiv a(X).[\![P_1]\!] \xrightarrow{a([\![A]\!])} \equiv [\![P_1]\!]\{[\![A]\!]/X\} \asymp [\![P_1\{A/X\}]\!].$$

- Case 2. P is of the form $\bar{a}A.P_1$. P can only do an output action such that $P \xrightarrow{\bar{a}A} P_1$, then for all $E[X]$, we have

$$[\![P]\!] \equiv \bar{a}[\![A]\!].[\![P_1]\!] \xrightarrow{\bar{a}[\![A_1]\!]} [\![P_1]\!],$$

  and it clearly holds that $(E[[\![A]\!]] \mid [\![P_1]\!]) \asymp (E[[\![A]\!]] \mid [\![P_1]\!])$.

- Case 3. P is of the form $(c)P_1$. There are 4 subcases.
  - $(c)P_1 \xrightarrow{\tau} P'$. It must be the case that $P_1 \xrightarrow{\tau} P_1'$ for some P_1' and $P' \equiv (c)P_1'$. By ind. hyp.

$$[\![P_1]\!] \Rightarrow \xrightarrow{\tau} \Rightarrow T_1 \asymp [\![P_1']\!]$$

    for some T_1, then $[\![P]\!] \equiv (c)[\![P_1]\!] \Rightarrow \xrightarrow{\tau} \Rightarrow (c)T_1 \asymp (c)[\![P_1']\!] \equiv [\![P']\!]$.
  - $(c)P_1 \xrightarrow{a(A)} P'$. This subcase is analogous to the above one.
  - $(c)P_1 \xrightarrow{(c,\widetilde{d})\bar{a}A} P'$ and $c \in fn(A)$. Then it must be the case that $P_1 \xrightarrow{(\widetilde{d})\bar{a}A} P'$. Now for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{d}, c\} = \emptyset$, by ind. hyp. $[\![P_1]\!] \Rightarrow \xrightarrow{(\widetilde{e_1})\bar{a}B_1} \Rightarrow T_1$ for some $\widetilde{e_1}$, B_1 and T_1, and it holds that $fn(E[X]) \cap \{\widetilde{e_1}\} = \emptyset$ and

$$(\widetilde{d})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (\widetilde{e_1})(E[B_1] \mid T_1) \tag{B.1}$$

    If $c \in fn(B_1)$, then $[\![P]\!] = (c)[\![P_1]\!] \Rightarrow \xrightarrow{(c,\widetilde{e_1})\bar{a}B_1} \Rightarrow T_1$. By (B.1) and congruence we are done. If $c \notin fn(B_1)$, then let c' be a fresh name, and $(c)[\![P_1]\!] \Rightarrow \xrightarrow{(\widetilde{e_1})\bar{a}B_1} \Rightarrow (c')(T_1\{c'/c\})$, then by congruence and α-conversion we have

$$(c, \widetilde{d})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (c, \widetilde{e_1})(E[B_1] \mid T_1) \asymp (\widetilde{e_1})(E[B_1] \mid (c')T_1\{c'/c\}). \tag{B.2}$$

– $(c)P_1 \xrightarrow{(\widetilde{d})\bar{a}A} (c)P'$, $c \notin fn(A)$, then $P_1 \xrightarrow{(\widetilde{d})\bar{a}A} P'$. Now by ind. hyp. for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{d}, c\} = \emptyset$, there are some $\widetilde{e}$, B and T such that $[\![P_1]\!] \Rightarrow \xrightarrow{(\widetilde{e})\bar{a}B} \Rightarrow T$, and it holds that $fn(E[X]) \cap \{\widetilde{e}\} = \emptyset$ and

$$(\widetilde{d})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (\widetilde{e})(E[B] \mid T) \tag{B.3}$$

If $c \in fn(B)$, then let c' be some fresh name such that $[\![P]\!] = (c)[\![P_1]\!] \Rightarrow \xrightarrow{(c',\widetilde{e})\bar{a}B_1} \Rightarrow T_1 = T\{c'/c\}$, where $B_1 = B\{c'/c\}$. Clearly $fn(E[X]) \cap \{c', \widetilde{e}\} = \emptyset$, by (B.3) and congruence we have

$$(\widetilde{d})(E[[\![A]\!]] \mid (c)[\![P']\!]) \asymp (c, \widetilde{d})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (c, \widetilde{e})(E[B] \mid T) \asymp (c', \widetilde{e})(E[B_1] \mid T_1) \tag{B.4}$$

If $c \notin fn(B)$, then $[\![P]\!] = (c)[\![P_1]\!] \Rightarrow \xrightarrow{(\widetilde{e})\bar{a}B} \Rightarrow (c)T$ and by congruence and α-conversion we have

$$(\widetilde{d})(E[[\![A]\!]] \mid (c)[\![P']\!]) \asymp (c, \widetilde{d})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (c, \widetilde{e})(E[B] \mid T) \asymp (\widetilde{e})(E[B] \mid (c)T) \tag{B.5}$$

- Case 4. P is of the form $P_1 \mid P_2$. There are 2 subcases.

1. $P \xrightarrow{\lambda} P'$ is caused by P_1 or P_2 alone, where $\lambda = a(A)$, $(\widetilde{c})\bar{a}A$, or τ. This subcase is trivial.
2. $P \xrightarrow{\tau} P'$ and this τ action is caused by a communication between P_1 and P_2. W.l.o.g., we assume that this is caused by $P_1 \xrightarrow{a(A)} P_1'$, $P_2 \xrightarrow{(\widetilde{c})\bar{a}A} P_2'$ and $P' \equiv (c)(P_1' \mid P_2')$. By ind. hyp. for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$ there are some $\widetilde{d}$, B and T_2 such that $[\![P_2]\!] \Rightarrow \xrightarrow{(\widetilde{d})\bar{a}B} \Rightarrow T_2$, and it holds that

$$(\widetilde{d})(E[B] \mid T_2) \asymp (\widetilde{c})(E[[\![A]\!]] \mid [\![P_2']\!]) \tag{B.6}$$

$fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$. By Lemma 11 there is some $B' \in \Pi^D$ such that $[\![B']\!] = B$. By Lemma 10, there are some $F[X]$ such that $P_1' = F[A]$ and $P_1 \xrightarrow{a(B')} F[B']$. By α-conversion we can require that $fn(F[X]) \cap \{\widetilde{c}\} = \emptyset$. By ind. hyp. $[\![P_1]\!] \Rightarrow \xrightarrow{a([\![B']\!])} \Rightarrow T_1$ and $T_1 \asymp [\![F[B']]\!] = [\![F]\!][[\![B']\!]] = [\![F]\!][B]$. Now we can take $E[X] = [\![F]\!][X]$ and we have $[\![P_1]\!] \mid [\![P_2]\!] \xrightarrow{\tau} (\widetilde{d})(T_1 \mid T_2)$ and it holds that

$$(\widetilde{d})(T_1 \mid T_2) \asymp (\widetilde{d})([\![F]\!][B] \mid T_2) \asymp (\widetilde{c})([\![F]\!][[\![A]\!]] \mid [\![P_2']\!]) = [\![(\widetilde{c})(P_1' \mid P_2')]\!] \tag{B.7}$$

- Case 5. P is of the form $(\langle X_1 \rangle P_1)\langle Q_1 \rangle$. Let $P_0 = P$, w.l.o.g, we can assume that there is some $n \geq 0$ such that for all $0 \leq i \leq n$ we have $P_i = (\langle X_{i_1} \rangle P_{i+1})\langle Q_{i+1} \rangle$ and $P_{n+1}\{Q_{n+1}/X_{n+1}\}$ is not in application form. Then if $P \xrightarrow{\lambda} P'$, we must have

$$P_{n+1}\{Q_{n+1}/X_{n+1}\}\{Q_n/X_n\} \ldots \{Q_1/X_1\} \xrightarrow{\lambda} P' \tag{B.8}$$

We only deal with the output case here. Assume $\lambda = (\widetilde{c})\bar{a}A$, then by case 4, for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$, there are some $\widetilde{d}$, B_1 and T_1 such that $[\![P_{n+1}\{Q_{n+1}/X_{n+1}\}\{Q_n/X_n\} \ldots \{Q_1/X_1\}]\!] \Rightarrow \xrightarrow{(\widetilde{d})\bar{a}B_1} \Rightarrow T_1$ and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and

$$(\widetilde{c})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (\widetilde{d})(E[B_1] \mid T_1). \tag{B.9}$$

It is not hard to show by induction that $[\![P]\!] \asymp [\![P_{n+1}\{Q_{n+1/X_{n+1}}\}\{Q_n/X_n\} \ldots \{Q_1/X_1\}]\!]$. Then there are some $\widetilde{e}$, B and T such that $[\![P]\!] \Rightarrow \xrightarrow{(\widetilde{e})\bar{a}B} \Rightarrow T$ and it holds that $fn(E[X]) \cap \{\widetilde{e}\} = \emptyset$ and

$$(\widetilde{c})(E[[\![A]\!]] \mid [\![P']\!]) \asymp (\widetilde{d})(E[B_1] \mid T_1) \asymp (\widetilde{e})(E[B] \mid T) \tag{B.10}$$

□

Proof of Lemma 6

Proof. We only prove the (3) since (1) and (2) are straightforward by Lemma 12. Suppose that $P \Rightarrow P_1 \xrightarrow{(\widetilde{c})\bar{a}A} P_2 \Rightarrow P'$ for some P_1 and P_2, then by (1) there are T_1 such that $[\![P]\!] \Rightarrow T_1 \asymp [\![P_1]\!]$. By Lemma 12, for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$ there are some $\widetilde{e}$, C, T_2 such that

$$[\![P_1]\!] \Rightarrow \xrightarrow{(\widetilde{e})\bar{a}C} \Rightarrow T_2 \text{ with } (\widetilde{c})([\![P_2]\!] \mid E[[\![A]\!]]) \asymp (\widetilde{e})(T_2 \mid E[C]) \text{ and } fn(E[X]) \cap \{\widetilde{e}\} = \emptyset. \tag{B.11}$$

Now $T_1 \asymp [\![P_1]\!]$, then there are some $\widetilde{d}$, B and T_3 such that

$$T_1 \Rightarrow \xrightarrow{(\widetilde{d})\bar{a}B} \Rightarrow T_3 \text{ with } (\widetilde{d})(T_3 \cap E[B]) \asymp (\widetilde{e})(T_2 \cap E[C]) \text{ and } fn[E[X]] \cap \{\widetilde{d}\} = \emptyset. \tag{B.12}$$

Let $F = b.E[X]$ for some fresh name b. By (B.11) and (B.12) and arbitrariness of E we have

$$(\widetilde{c})([\![P_2]\!] \mid b.E[[\![A]\!]]) \asymp (\widetilde{d})(T_2 \mid b.E[B]) \tag{B.13}$$

Now $P_2 \Rightarrow P'$, then by (1) there are some T_4 such that $[\![P_2]\!] \Rightarrow T_4 \asymp [\![P']\!]$ which implies that

$$(\widetilde{c})([\![P_2]\!] \mid b.E[[\![A]\!]]) \Rightarrow (\widetilde{c})(T_4 \mid b.E[[\![A]\!]]) \asymp (\widetilde{c})([\![P']\!] \mid b.E[[\![A]\!]]). \tag{B.14}$$

The above transition can be matched by

$$(\widetilde{d})(T_3 \mid b.E[B]) \Rightarrow (\widetilde{d})(T \mid b.E[B]) \asymp (\widetilde{c})([\![P']\!] \mid b.E[[\![A]\!]]), \tag{B.15}$$

and for some T such that $T_3 \Rightarrow T$. It remains to show $(\widetilde{c})([\![P']\!] \mid E[[\![A]\!]]) \asymp (\widetilde{d})(T \mid E[B])$. Now assume that $(\widetilde{d})(T \mid b.E[B]) \xrightarrow{b(0)} (\widetilde{d})(T \mid E[B])$ is matched by

$$(\widetilde{c})([\![P']\!] \mid b.E[[\![A]\!]]) \Rightarrow (\widetilde{c})(P_3 \mid b.E[[\![A]\!]]) \xrightarrow{b(0)} (\widetilde{c})(P_3 \mid E[[\![A]\!]]) \Rightarrow P_4 \tag{B.16}$$

for some P_3 and P_4 which implies that $(\widetilde{c})([\![P']\!] \mid E[[\![A]\!]]) \Rightarrow (\widetilde{c})(P_3 \mid E[[\![A]\!]]) \Rightarrow P_4 \asymp (\widetilde{d})(T \mid E[B])$. Similarly we have $(\widetilde{d})(T \mid E[B]) \Rightarrow \asymp (\widetilde{c})([\![P']\!] \mid E[[\![A]\!]])$. It follows by Bisimulation Lemma [44] that $(\widetilde{d})(T \mid E[B]) \asymp (\widetilde{c})([\![P']\!] \mid E[[\![A]\!]])$. □

Appendix B.2. Proof of Lemma 7 (Backward Operational Correspondence)

In order to make the proof smooth, we first introduce an auxiliary notion, *weak context expansion*. Intuitively weak context expansion is an asymmetric version of weak context bisimulation [45].

Definition 3. A binary relation $\mathcal{R}$ is weak context expansion if $\mathcal{R}$ is a weak context bisimulation and the following statements are valid:

1. If $Q\mathcal{R}^{-1}P \xrightarrow{\tau} P'$, then either $P'\mathcal{R}Q$ or $P'\mathcal{R}Q'$ for some Q' such that $Q \xrightarrow{\tau} Q'$.
2. If $Q\mathcal{R}^{-1}P \xrightarrow{a(A)} P'$, then $Q \xrightarrow{a(A)} Q'\mathcal{R}^{-1}P'$.
3. If $Q\mathcal{R}^{-1}P \xrightarrow{(\widetilde{c})\bar{a}A} P'$, then for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$, there are some $\widetilde{d}$, B and Q' such that $Q \xrightarrow{(\widetilde{d})\bar{a}B} Q'$ and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and $(\widetilde{c})(P' \mid E[A])\mathcal{R}(\widetilde{d})(Q' \mid E[B])$.

It is easy to see that if $\mathcal{R}_1$ and $\mathcal{R}_2$ are weak context expansion, then $\mathcal{R}_1;\mathcal{R}_2$ and $\mathcal{R}_1 \cup \mathcal{R}_2$ are both weak context expansion. The largest weak context expansion contained in weak context bisimilarity is denoted by $\asymp^{\triangleright}$. We say P expands Q, denoted $P \asymp^{\triangleright} Q$ if there is some weak context expansion $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$. Clearly if $P \asymp^{\triangleright} Q$, then $P \asymp Q$. The following lemma is useful; see [45, 5] for a relevant proof.

Lemma 13. *If $P \asymp^{\triangleright} Q$, then for all c and R it holds that $(c)P \asymp^{\triangleright} (c)Q$, $P \mid R \asymp^{\triangleright} Q \mid R$ and $R \mid P \asymp^{\triangleright} R \mid Q$.*

Lemma 14. *Assume m is a fresh name and $fv(P) \subseteq \{X\}$, then*

$$(m)(P\{\bar{m}/X\} \mid !m.Q) \asymp^{\triangleright} P\{Q/X\}.$$

Proof. Clearly $(m)(P\{\bar{m}/X\} \mid !m.Q) \asymp P\{Q/X\}$. Now it is a routine task to check the relation $\mathcal{R}$ defined as

$$\mathcal{R} = \{((m)(P\{\bar{m}/X\} \mid !m.Q) \mid T,\ P\{Q/X\} \mid T) \mid m \text{ is a fresh name}\}$$

is a weak context expansion. □

Like the case of forward operational correspondence, we first deal with the strong version.

Lemma 15 (Strong Backward Operational Correspondence)**.** *Let P be a Π_1^D process, the following statements are valid.*

1. *If $[\![P]\!] \xrightarrow{\tau} T$, then there exists some P' such that $T \asymp^{\triangleright} [\![P']\!]$ and either $P \xrightarrow{\tau} P'$ or $P \equiv P'$.*
2. *If $[\![P]\!] \xrightarrow{a(A)} T$ and there is some B such that $[\![B]\!] \equiv A$, then $P \xrightarrow{a(B)} P'$ for some P' such that $T \asymp^{\triangleright} [\![P']\!]$.*
3. *If $[\![P]\!] \xrightarrow{(\widetilde{c})\bar{a}A} T$, then for all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$ there are some $\widetilde{d}$, B, and P' such that $P \xrightarrow{(\widetilde{d})\bar{a}B} P'$ and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and*

$$(\widetilde{c})(T \mid E[A]) \asymp^{\triangleright} (\widetilde{d})([\![P']\!] \mid E[[\![B]\!]]).$$

Proof. The proof proceeds by induction on the structure of P. We consider the following cases.

- Case 1. P is of the form $a(X).P_1$. In this case $[\![P]\!]$ can only perform an input action such that

$$[\![P]\!] = a(X).[\![P_1]\!] \xrightarrow{a(A)} T = [\![P_1]\!]\{A/X\}$$

  for some A and T and there is some $B \in \Pi_1^D$ such that $[\![B]\!] = A$. Then $P \xrightarrow{a(B)} P_1\{B/X\}$ and

$$[\![P_1\{B/X\}]\!] = [\![P_1]\!]\{A/X\} \asymp^{\triangleright} T.$$

- Case 2. P is of the form $\bar{a}A.P_1$. $[\![P]\!]$ and P can only perform an output action such that

$$[\![P]\!] = \bar{a}[\![A]\!].[\![P_1]\!] \xrightarrow{\bar{a}[\![A]\!]} [\![P_1]\!] \text{ and } P \xrightarrow{\bar{a}A} P_1.$$

- Case 3. P is of the form $(c)P_1$. The analysis of this case is analogous to Case 3 of proof for Lemma 12.
- Case 4. The analysis of this case is analogous to Case 4 of proof for Lemma 12.
- Case 5. P is of the form $(\langle X_1 \rangle P_1)\langle Q_1 \rangle$. Let $P_0 = P$, w.l.o.g, we can assume that there is some $n \geq 0$ such that for all $0 \leq i \leq n$ we have $P_i = (\langle X_{i+1} \rangle P_{i+1})\langle Q_{i+1} \rangle$ and $R = P_{n+1}\{Q_{n+1}/X_{n+1}\} \ldots \{Q_1/X_1\}$ is not in application form. The by Lemma 14 and by induction we have $[\![P]\!] \asymp^{\triangleright} [\![R]\!]$. Now suppose that $[\![P]\!] \xrightarrow{\lambda} T$, then there are there subcases to consider: $\lambda = a(B)$, $\lambda = (\widetilde{c})\bar{a}B$ and $\lambda = \tau$.

  (a) $\lambda = \tau$. By definition of $\asymp^{\triangleright}$, either $T \asymp^{\triangleright} [\![R]\!]$ or $[\![R]\!] \xrightarrow{\tau} T'$ and $T \asymp^{\triangleright} T'$. If $T \asymp^{\triangleright} [\![R]\!]$, then we have $P \equiv R$. Otherwise by previous cases there are some R' such that $T \asymp^{\triangleright} T' \asymp^{\triangleright} [\![R']\!]$ and either $R \equiv R'$ or $R \xrightarrow{\tau} R'$. As $P \equiv R$, by transitivity of $\asymp^{\triangleright}$ there is some P' such that $T \asymp^{\triangleright} [\![P']\!]$ and either $P \equiv P'$ or $P \xrightarrow{\tau} P'$.

  (b) $\lambda = a(A)$. This subcase is analogous to subcase (a).

  (c) $\lambda = (\widetilde{c})\bar{a}A$. For all $E[X]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$, there are some $\widetilde{d_1}$, B_1 and T_1 such that $[\![R]\!] \xrightarrow{(\widetilde{d_1})\bar{a}B_1} T_1$ and it holds that $fn(E[X]) \cap \{d_1\} = \emptyset$ and $(\widetilde{c})(E[A] \mid T) \asymp^{\triangleright} (\widetilde{d_1})(E[B_1] \mid T_1)$. By analysis of the previous cases, there are some $\widetilde{d}$, B and P' such that $P \xrightarrow{(\widetilde{d})\bar{a}B} P'$, and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and

$$(\widetilde{c})(E[A] \mid T) \asymp^{\triangleright} (\widetilde{d_1})(E[B_1] \mid T_1) \asymp^{\triangleright} (\widetilde{d})(E[[\![B]\!]] \mid [\![P']\!]) \tag{B.17}$$

□

The following property is immediate from the lemmas above.

Corollary 16. *If $[\![P]\!] \Rightarrow T$, then there are some P' such that $P \Rightarrow \equiv P'$ and $T \asymp^{\triangleright} [\![P']\!]$.*

Now with the help of weak context expansion and Lemma 15, it is straightforward to establish the weak backward operational correspondence, in a way similar to that for forward operational correspondence. We thus skip the details.

Appendix B.3. Proof of Lemma 8

Proof. We prove the relation $\mathcal{R}$ defined by

$$\mathcal{R} \stackrel{\text{def}}{=} \{([\![P]\!], [\![Q]\!]) \mid P \asymp Q\}$$

is a context bisimulation up-to $\doteq$ [5] in $[\![\Pi^D]\!]$, and thus $\mathcal{R} \subseteq \doteq$. Suppose that $[\![P]\!] \stackrel{\lambda}{\Rightarrow} T$, there are three cases to consider.

- Case 1. $\lambda = a(A)$ and there is some $B \in \Pi^D$ s.t. $[\![B]\!] = A$. By lemma 7, $P \stackrel{a(B)}{\Longrightarrow} P'$ and $T \doteq [\![P']\!]$. Since $P \asymp Q$, then Q can simulate P with $Q \stackrel{a(B)}{\Longrightarrow} Q' \asymp P'$. By lemma 6 we know that $[\![Q]\!] \stackrel{a([\![B]\!])}{\Longrightarrow} T'$ and $T' \doteq [\![Q']\!]$. As a result, we have $[\![Q]\!] \stackrel{a(A)}{\Longrightarrow} T'$ and

$$T \doteq [\![P']\!] \; \mathcal{R} \; [\![Q']\!] \doteq T' \tag{B.18}$$

- Case 2. $\lambda = (\widetilde{c})\bar{a}A$. By lemma 7, for all $E[X] \in [\![\Pi^D]\!]$ with $fn(E[X]) \cap \{\widetilde{c}\} = \emptyset$ there exist some $\widetilde{d}$, B s.t. $P \stackrel{(\widetilde{d})\bar{a}B}{\Longrightarrow} P'$ and it holds that $fn(E[X]) \cap \{\widetilde{d}\} = \emptyset$ and

$$(\widetilde{c})(E[A] \mid T) \doteq (\widetilde{d})(E[[\![B]\!]] \mid [\![P']\!]). \tag{B.19}$$

  Now let $F[X]$ be the context in Π^D s.t. $[\![F]\!][X] = E[X]$. Clearly $fn(F[X]) = fn(E[X])$. Since $P \asymp Q$, then Q can simulate P with $Q \stackrel{(\widetilde{e})\bar{a}B'}{\Longrightarrow} Q'$ for some $\widetilde{e}$ and B' s.t. $fn(F[X]) \cap \{\widetilde{e}\} = \emptyset$ and

$$(\widetilde{d})(F[B] \mid P') \asymp (\widetilde{e})(F[B'] \mid Q'). \tag{B.20}$$

  By Lemma 6 , there exist some $\widetilde{f}$, A' and T' s.t. $[\![Q]\!] \stackrel{(\widetilde{f})\bar{a}[\![A']\!]}{\Longrightarrow} T'$ and it holds that $fn(E[X]) \cap \{\widetilde{f}\} = \emptyset$ and

$$(\widetilde{e})(E[[\![B']\!]] \mid [\![Q']\!]) \doteq (\widetilde{f})(E[A'] \mid T') \tag{B.21}$$

  Now by (B.19,B.20,B.21), we have

$$(\widetilde{c})(E[A] \mid T) \doteq [\![(\widetilde{d})(F[B] \mid P')]\!] \; \mathcal{R} \; [\![(\widetilde{e})(F[B'] \mid Q')]\!] \doteq (\widetilde{f})(E[A'] \mid T'). \tag{B.22}$$

- Case 3. $\lambda = \tau$. This case is similar to Case 1.

The proof is now complete. □