On the Expressiveness of Parameterization in Process-passing

Xian Xu[1] ⋆, Qiang Yin[2] ⋆, and Huan Long[2] ⋆

[1] East China University of Science and Technology, China. xuxian@ecust.edu.cn
[2] Shanghai Jiao Tong University, China.
yinqiang.sjtu@gmail.com,longhuan@sjtu.edu.cn

Abstract. This paper studies higher-order processes with the capability of parameterization (or abstraction), which has been proven to be an effective measure of lifting the mere process-passing in expressiveness. We contribute to the understanding of two kinds of parameterization: name-parameterization and process-parameterization, particularly in a second-order setting (no currying of parameterization). Firstly, we show that in the expressiveness hierarchy of process-parameterization, n-ary parameterization can faithfully translate (n+1)-ary parameterization in a linear setting where each received process can be used only once. Secondly, the two kinds of parameterization are compared. We prove that name-parameterization is more basic than process-parameterization, i.e. the former can encode the union of them. As a result, name-parameterization can strictly promote the expressiveness of mere process-passing.

Keywords: Parameterization, Expressiveness, Higher-order, Processes

1 Introduction

Higher-order process model is one of the important formal methods in the area of concurrency theory [22] [21]. It differs from the classic first-order model when dealing with the formalization of applications including object-oriented computing and (distributed) service-oriented design and programming, because higher-order processes are capable of passing an integrate program instead of a reference to its code. One of the fundamental problems concerning higher-order programming is the expressiveness of higher-order processes. Recent years have seen increased importance of both positive and negative results in the studies of expressiveness. On the whole, the endeavor dwells on the relationship to and from: (1) classical computational models (e.g. Turing machines or the equivalent [8,11,12]); (2) interactive models (e.g. first-order process calculi [17,18,23–25] and Petri Nets [13]); (3) variants of higher-order processes [3, 10, 12]. An important

⋆ The authors are supported by the National Nature Science Foundation of China (61261130589,61202023,61173048), and the ANR project 12IS02001 (PACE).

way to promote the expressiveness of pure higher-order processes is by extending it with certain name-handling mechanism. Related results also contribute to the study of programming in higher-order languages, which seems to inevitably need both process passing and name handling in addition to some recursion mechanism. One advantage of modeling (or programming) with higher-order processes, among others (e.g. recursion), is that it is simple to reproduce some process (e.g. using $a(X).(X|X)$ that inputs a process over port a and clones another copy of it), whereas this seems not so straightforward with first-order processes. On the other hand, sometimes arbitrary reproduction is not allowed in practice (e.g. security, copyright issues). To this end, linearity regulates the replicating ability of higher-order processes by requiring that a received process can be used only once [22] [26] (thus $a(X).(X|X)$ is ruled out). Linearity is witnessed in distributed computing, e.g. video on demand, online bank's paying session, (mobile) code update, and is an important topic in process calculi [22].

In this paper, we focus on the higher-order process models equipped with the *parameterization* operation. Parameterization (i.e., abstraction) [17] 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 functions that map a variable (higher-order or first-order) to an object (process or name). From a certain standpoint, for example the construction of a compiler, parameterization is a soundly realized apparatus of instantiation. The mechanism of such apparatus cannot be achieved using mere process-passing, as is proven in [10] that process-passing is strictly less expressive than the passing of higher-order parameterization. However, available work in the field did not look *into* the two kinds of parameterization. So it would be interesting to know more about the expressiveness *inside* the parameterization operation, which forms the key point of this paper. More review of related work that motivates our work is provided in Section 5.

We use Π to represent the basic higher-order pi-calculus [22]. Π^D_n is Π extended with n-ary parameterization on processes ($n \in \mathbb{N}$), where $\mathbb{N}$ is the set of natural numbers; when $n = 0$, it is simply Π. Π^D is $\bigcup_{n\in\mathbb{N}} \Pi^D_n$. Π^d_n is Π extended with n-ary parameterization on names ($n \in \mathbb{N}$). Π^d is $\bigcup_{n\in\mathbb{N}} \Pi^d_n$. Π^{Dl}_n is linear Π^D_n (i.e. imposing on Π^D_n the concept of linearity as described above), similar for Π^{dl}_n. $\Pi^{D,d}_n$ stands for the union of Π^D_n and Π^d_n, similar for other combinations.

In theoretical and practical scenarios, most processes are second-order, that is, any parameterization does not occur on another parameterization (no currying). So we focus on the expressiveness of second-order processes. This is further motivated below: (1)Practically, it is not a desirable programming style to define parameterization on themselves. Particularly, all the (currying) parameters can be put together to be a vector of parameters and thus become second-order. (2)Methodologically, the encoding strategies, as mentioned in the contribution below, reveal some programming approach using parameterized processes (in the second-order setting), and is potentially applicable to related work. For instance, the second contribution implies that using name-parameterization in distributed computing is sufficient, at least in expressiveness, thought sometimes

using process-parameterization may offer certain conciseness. This differs from the well-known result of encoding Π into π (name-passing calculus of Milner et al. [15]), because here all the processes are strictly higher-order, i.e. no names can be communicated.

Contribution This paper makes the following contributions. ($\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ denotes calculus $\mathcal{L}_1$ is encodable in calculus $\mathcal{L}_2$ w.r.t. some criteria defined in Section 2.)

1. The expressiveness hierarchy of process-parameterization is studied: we show that $\Pi^{Dl}_{n+1} \sqsubseteq \Pi^{D}_{n}$. This reveals that lowering the arity in a setting of linearity and process parameterization somewhat amounts to relinquishing linearity in the meanwhile without losing expressiveness.
2. The relationship between Π^{D}_{n} and Π^{d}_{n} is examined: we show that $\Pi^{D,d}_{n} \sqsubseteq \Pi^{d}_{n}$. This is somehow surprising: it says that adding Π^{D}_{n} to Π^{d}_{n} is no more interesting than using Π^{d}_{n} alone. Moreover, an important corollary is that Π^{d}_{n} is strictly more expressive than Π.

The paper is structured as follows. Section 2 defines the relevant calculi and the encoding criteria. Section 3 establishes $\Pi^{Dl}_{n+1} \sqsubseteq \Pi^{D}_{n}$. Section 4 shows $\Pi^{D,d}_{n} \sqsubseteq \Pi^{d}_{n}$. Section 5 concludes this paper, mentions future work, and describes related work.

2 The calculi and the notion of encoding

2.1 The calculi

Calculus Π

Π processes, denoted by uppercase letters ($T, P, Q, R, A, B, E, F...$), are defined by the following grammar. Lowercase letters stand for channel names. Letters X, Y, Z represent process variables. We use Π_{seg} for convenience.

$$\Pi_{seg} ::= 0 \mid X \mid u(X).T \mid \overline{u}T'.T \mid T|\,T' \mid (c)T$$

The operators are: prefix:$u(X).T, \overline{u}T'.T$; composition: $T|\,T'$; restriction: $(c)T$ in which c is bound. Parallel composition has the least precedence. As usual some 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 $(a)(a.P|\,\overline{a})$; sometimes $\overline{a}[A].T$ for $\overline{a}A.T$; $\widetilde{\cdot}$ for a finite sequence of something. By standard definition, $fn(\widetilde{T})$, $bn(\widetilde{T})$, $n(\widetilde{T})$; $fv(\widetilde{T})$, $bv(\widetilde{T})$, $v(\widetilde{T})$ respectively denote free names, bound names, all names; free variables, bound variables and all variables in $\widetilde{T}$. Closed processes contain no free variables. A fresh name or variable is one that does not occur in the processes under consideration. Name substitution $T\{\widetilde{n}/\widetilde{m}\}$ and higher-order substitution $T\{\widetilde{A}/\widetilde{X}\}$ are defined structurally in a standard way. $E[\widetilde{X}]$ denotes E with (at most) variables $\widetilde{X}$, and $E[\widetilde{A}]$ stands for $E\{\widetilde{A}/\widetilde{X}\}$. We work up-to α-conversion and always assume no capture. We use input-guarded replication as a derived operator [12, 24]: $!\phi.P \stackrel{def}{=} (c)(Q_{c,\phi,P}|\,\overline{c}Q_{c,\phi,P})$, $Q_{c,\phi,P} \stackrel{def}{=} c(X).(\phi.(X|\,P)|\,\overline{c}X)$, where ϕ is a prefix.

The operational semantics is given in Figure 1. Symmetric rules are omitted. The rules are mostly self-explanatory. For instance, in higher-order input $a(A)$, the received process A becomes part of the receiving environment through a substitution. $\alpha, \beta, \lambda, ...$ denote action, whose subject indicates the channel name on which it happens. Operations $fn(), bn(), n()$ can be similarly defined on actions. $\Longrightarrow$ is the reflexive transitive closure of internal actions τ, and $\stackrel{\lambda}{\Longrightarrow}$ is $\Longrightarrow \stackrel{\lambda}{\rightarrow} \Longrightarrow$. $\stackrel{\hat{\lambda}}{\Longrightarrow}$ is $\Longrightarrow$ when λ is τ and $\stackrel{\lambda}{\Longrightarrow}$ otherwise. $\stackrel{\tau}{\rightarrow}_k$ means k consecutive τ's. "$P \stackrel{\hat{\lambda}}{\Longrightarrow} P'$" abbreviates "there is a process P' such that $P \stackrel{\hat{\lambda}}{\Longrightarrow} P'$". $P \Longrightarrow \cdot \mathcal{R}\, Q$ means $P \Longrightarrow Q'$ and $Q' \,\mathcal{R}\, Q$ (i.e. $(Q', Q) \in \mathcal{R}$), where $\mathcal{R}$ is a binary relation. We say relation $\mathcal{R}$ is closed under (variable) substitution if $(E\{A/X\}, F\{A/X\}) \in \mathcal{R}$ for any A whenever $(E, F) \in \mathcal{R}$. A process diverges if it can perform an infinite τ sequence.

$$\frac{}{a(X).T \xrightarrow{a(A)} T\{A/X\}} \quad \frac{}{\overline{a}A.T \xrightarrow{\overline{a}A} T} \quad \frac{T \xrightarrow{\lambda} T'}{(c)T \xrightarrow{\lambda} (c)T'}\, c \notin n(\lambda) \quad \frac{T \xrightarrow{\lambda} T'}{T \mid T_1 \xrightarrow{\lambda} T' \mid T_1}\, bn(\lambda) \cap fn(T_1) = \emptyset$$

$$\frac{T \xrightarrow{(\widetilde{c})\overline{a}[A]} T'}{(d)T \xrightarrow{(d)(\widetilde{c})\overline{a}[A]} T'}\, d \in fn(A) - \{\widetilde{c}, a\} \qquad \frac{T_1 \xrightarrow{a(A)} T_1', T_2 \xrightarrow{(\widetilde{c})\overline{a}[A]} T_2'}{T_1 \mid T_2 \xrightarrow{\tau} (\widetilde{c})(T_1' \mid T_2')}\, \widetilde{c} \cap fn(T_1) = \emptyset$$

Fig. 1. Semantics of Π

Calculi Π_n^D ***and*** Π_n^d

Parameterization extends Π with the syntax and semantics in Figure 2. $\langle U_1, U_2, ..., U_n \rangle T$ is n-ary parameterization where $U_1, U_2, ..., U_n$ are the formal parameters to be instantiated by the n-ary application $T\langle K_1, K_2, ..., K_n \rangle$ where the parameters are instantiated by concrete objects $K_1, K_2, ..., K_n$. In the rule, $|\widetilde{U}| = |\widetilde{K}| = n$ requires the parameters and the instantiating objects should be equally sized (equal to n). It also expresses that the parameterized process can do an action only after the application happens. Calculus Π_n^D, which has process parameterization (or higher-order abstraction), is defined by taking $\widetilde{U}, \widetilde{K}$ as $\widetilde{X}, \widetilde{T'}$ respectively. Calculus Π_n^d, which has name parameterization (or first-order abstraction), is defined by taking $\widetilde{U}, \widetilde{K}$ as $\widetilde{x}, \widetilde{u}$ respectively. For convenience, names (ranged over by u, v, w) are handled dichotomically: name constants (ranged over by $a, b, c, ..., m, n$); name variables (ranged over by x, y, z).

$$T ::= \Pi_{seg} \mid \langle U_1, U_2, ..., U_n \rangle T \mid T\langle K_1, K_2, ..., K_n \rangle$$
$$\frac{T\{\widetilde{K}/\widetilde{U}\} \xrightarrow{\lambda} T'}{F\langle \widetilde{K} \rangle \xrightarrow{\lambda} T'} \quad \text{if } F \stackrel{def}{=} \langle \widetilde{U} \rangle T \;\; (|\widetilde{U}| = |\widetilde{K}| = n)$$

Fig. 2. Π with parameterization

Process expressions (or terms) of the form $\langle\widetilde{X}\rangle P$ or $\langle\widetilde{x}\rangle P$, in which $\widetilde{X}$ or $\widetilde{x}$ is not empty, are *parameterized processes*. Terms without outmost parameterization are *non-parameterized processes*, or simply processes. We mainly focus on processes in the encodings to be presented. Sometimes related notations are slightly abused if no confusion is caused. Only free variables can be effectively parameterized; they become *bound* after parameterization. In the syntax, null-parameterizations, for example $\langle X_1, X_2\rangle P$ in which $X_2 \notin fv(P)$, are allowed. A $\Pi^D_n(n \geq 1)$ process is definable in Π^D_{n+1} (similar for Π^d_n) by making use of fresh dummy variable. The semantics of parameterization renders it somewhat natural to deem application as extra rule of structural congruence, denoted by $\equiv$, which is defined in the standard way [14, 19]. In addition to the standard rules (e.g. monoid rules for composition), it includes the application rules: $(\langle\widetilde{X}\rangle P)\langle\widetilde{A}\rangle \equiv P\{\widetilde{A}/\widetilde{X}\}$ and $(\langle\widetilde{x}\rangle P)\langle\widetilde{u}\rangle \equiv P\{\widetilde{u}/\widetilde{x}\}$.

Type systems for Π^D_n and Π^d_n (also $\Pi^{D,d}_n$) can be routinely defined in a similar way to that in [17], where the concept of *order* is also formalized. As mentioned, here we require that only second-order processes be admitted. So parameterizing does not occur on parameterized processes, and application does not use parameterized processes as instance. That is, we stipulate that in $\langle\widetilde{U}\rangle T'$ and $T\langle\widetilde{T'}\rangle$, T' are not parameterized processes. This prohibits processes such as $\langle X\rangle(\langle Y\rangle T)$ and $(\langle X\rangle(X\langle A\rangle))\langle B\rangle$ in which $B \equiv \langle Y\rangle Y$; instead, in the former one could use $\langle X, Y\rangle T$, and in the latter the instance A could be sent to a service containing B before retrieving the result. The main reason we restrict to second-order processes is that it serves well in related study (e.g. the encodings in this paper and work from [17]) and can also simplify type consistency in programming. Technically this rules out a trivial encoding in section 3 (some more remark is given in its end), simplifies the encoding in section 4 because it is currently not obvious how to extend to the general case.

Linearity

Linearity originates from resource-sensitiveness, and requires that a received resource (e.g. a piece of update code) can be used only *once*. We define here linear Π^D_n, notation Π^{Dl}_n (Π^{dl}_n can be defined similarly). The only difference is that in the syntax of output and (parallel) composition, we impose linearity, i.e.

$$\overline{u}T'.T, \quad T|\,T', \text{ where } fv(T) \cap fv(T') = \emptyset$$

So in Π^{Dl}_n, $a(X).\overline{b}X.X$ is illegal because it tries to forward the received resource (perhaps secretly) while keeping a copy of it, whereas $a(X).b(Y).\overline{c}X.Y$ is legal. The intuition is that a same variable cannot appear more than once on concurrent position (viz. output and composition), so that once that variable is instantiated by a resource it is obliged to be used merely one time. Accordingly, in the semantics, we need to control the appearance of variables to ensure linearity during the evolution of a process. This can be achieved with some syntactical constraint or type system (see [26]), which we do not report here because it is not important for the study of expressiveness.

2.2 Context bisimulation and the encoding criteria

The well-established bisimulation equivalence of higher-order processes is the context bisimulation [17].

Definition 1. *A symmetric binary relation $\mathcal{R}$ on (closed) 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 $Q\overset{\hat{\alpha}}{\Longrightarrow}Q'$ for some Q' and $P'\,\mathcal{R}\,Q'$;*
2. *If $P\xrightarrow{(\widetilde{c})\overline{a}A}P'$ then $Q\overset{(\widetilde{d})\overline{a}B}{\Longrightarrow}Q'$ for some $\widetilde{d}, B, Q'$, and for every process $E[X]$ s.t. $\{\widetilde{c},\widetilde{d}\}\cap fn(E)=\emptyset$ it holds that $(\widetilde{c})(E[A]|\,P')\;\mathcal{R}\;(\widetilde{d})(E[B]|\,Q')$.*

Context bisimilarity, written $\approx_{Ct}$, is the largest context bisimulation.

Context bisimulation can be extended to general (open) processes in a standard way [17] so that more processes can be compared. Relation $\sim_{ct}$ is the strong version of $\approx_{Ct}$. For clarity, we use $=_{\mathcal{L}}$ (resp. $\sim_{\mathcal{L}}$) to denote the context bisimilarity (resp. strong context bisimilarity) of calculus $\mathcal{L}$ (Π, Π_n^D, Π_n^d, Π_n^{Dl}); we simply use $=$ (resp. $\sim$) when it is clear from context. Context bisimilarity is *an equivalence and a congruence*, and is characterized by normal bisimilarity and coincident to barbed bisimilarity; see [17–20].

Now we are ready for the notion of encoding. A process model $\mathcal{L}$ is a triplet $(\mathcal{P},\rightarrow,\approx)$, where $\mathcal{P}$ is the set of processes, $\rightarrow$ is the LTS with a set $\mathcal{A}$ of actions, and $\approx$ is a behavioral equivalence. Given $\mathcal{L}_i \overset{def}{=} (\mathcal{P}_i,\rightarrow_i,\approx_i)$ $(i{=}1,2)$, an encoding from $\mathcal{L}_1$ to $\mathcal{L}_2$ is a function $[\![\cdot]\!] : \mathcal{P}_1 \longrightarrow \mathcal{P}_2$ that satisfies a set of criteria. The following criteria set used in this paper is from [10] (a variant based on [4]). We use it instead of that in [4] because it yields better properties like transitivity of (qualified) encodings. $\mathcal{L}_1 \sqsubseteq \mathcal{L}_2$ means there is an encoding from $\mathcal{L}_1$ to $\mathcal{L}_2$. $\mathcal{L}_1 \not\sqsubseteq \mathcal{L}_2$ means otherwise.

Definition 2 (Criteria for encodings [10]). *There are two categories.*

***Static 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 context $C_{op}[\cdots]\in\mathcal{P}_2$;
(2) Name invariance. For any injective substitution σ of names, $[\![P\sigma]\!] = [\![P]\!]\sigma$.
***Dynamic criteria**:*
(1) Operational correspondence (forth). Whenever $P\overset{\beta}{\Longrightarrow}P'$, it holds $[\![P]\!]\overset{\lambda}{\Longrightarrow}\approx_2 [\![P']\!]$, for some action λ with the same *subject* as that of β;
(2) Operational correspondence (back). Whenever $[\![P]\!]\overset{\lambda}{\Longrightarrow}T$, there exist P' and β with the same *subject* as that of λ s.t. $P\overset{\beta}{\Longrightarrow}P'$ and $T\Longrightarrow\approx_2 [\![P']\!]$;
(3) Adequacy. $P\approx_1 P'$ implies $[\![P]\!]\approx_2[\![P']\!]$. This is also known as *soundness*. The converse is known as *completeness*;
(4) Diverge-reflecting. If $[\![P]\!]$ diverges, so does P.

Below are some **derived criteria**.

(1) Barb preservation. $P\Downarrow \nu$ (ν is a or $\overline{a}$, meaning $P \Longrightarrow P'$ for some P' and $P' \xrightarrow{\lambda} P''$ for some P'', with the subject of λ as ν) iff $[\![P]\!]\Downarrow \nu$.
(2) Free names preservation. If $a \in fn(P)$*, then* $a \in fn([\![P]\!])$.
(3) Transitivity. If $[\![\cdot]\!]_1$ is an encoding from $\mathcal{L}_1$ to $\mathcal{L}_2$, and $[\![\cdot]\!]_2$ is an encoding from $\mathcal{L}_2$ to $\mathcal{L}_3$, then the composition $[\![[\![\cdot]\!]_1]\!]_2$ is an encoding from $\mathcal{L}_1$ to $\mathcal{L}_3$.

3 On the expressiveness layer of process-parameterization

In this section we show that in the expressiveness hierarchy of process-parameterization, it holds that $\Pi^{Dl}_{n+1} \sqsubseteq \Pi^{D}_{n}$. This reveals that in a linear setting, the decrease of arity of parameterized variables can be compensated by relaxing the requirement of linearity. We use a kind of choice encoded as below (the definition is homomorphic on the other operators): $[\![a_1.P_1 + a_2.P_2]\!]_+ \stackrel{def}{=} \overline{a_1}[\![P_1]\!]_+ | \, \overline{a_2}[\![P_2]\!]_+$; the choice making is $[\![\bar{a_1}]\!]_+ \stackrel{def}{=} a_2(X_2).a_1(X_1).X_1, [\![\bar{a_2}]\!]_+ \stackrel{def}{=} a_1(X_1).a_2(X_2).X_2$. The encoding is correct as long as the choice making is done locally [12]. We use this choice operation for the sake of convenience.

3.1 $\Pi^{Dl}_{n+1} \sqsubseteq \Pi^{D}_{n}$

We focus on the encoding $\Pi^{Dl}_{2} \sqsubseteq \Pi^{D}_{1}$, and the encoding strategy can be extended to the general case (e.g. utilize the last parameter of a Π^{D}_{n} abstraction to interpret the last two parameters in an abstraction of Π^{Dl}_{n+1}). The central point of the translation is the encoding of (conveying) a Π^{Dl}_{2} abstraction concerning various environment. To this end, an abstraction can be regarded as some kind of premeditated input action consequence. To encode a Π^{Dl}_{2} abstraction, the "controller" is on the sender part, that is, the sender delivers a Π^{D}_{1} abstraction which is designed to work on the receiver's side to execute some protocol agreed upon initially. Such a protocol consists of a series of communication to achieve the effect of the communication of a Π^{Dl}_{2} abstraction. Moreover the encoding uses local names to control the sequence of the communication (activating the instance of a potential concrete process applied on the abstraction). Also the encoding must take into account potential interferences from its environment.

Intuitively, the encoding from Π^{Dl}_{2} to Π^{D}_{1} is homomorphism on all the operators except the following canonical scenario, which clarifies how the encoding should work. Notice we suppose the two encoded processes are in Π^{Dl}_{2}.

$$
\begin{aligned}
&[\![\overline{a}[\langle X_1, X_2\rangle E].P]\!] \stackrel{def}{=} (m)(\overline{a}[\langle Z\rangle \overline{m} Z].[\![P]\!] \,|\, m(X_1).(X_1 |\, m(X_2).[\![E]\!]) \\
&[\![a(Y).(Y\langle B_1, B_2\rangle |\, Q)]\!] \\
&\stackrel{def}{=} a(Y).((fgh)(\overline{g}|\, Y\langle g.(\overline{f}|\, \overline{h}) + f.([\![B_1]\!]|\, \overline{f})\rangle |\, h.Y\langle [\![B_2]\!]\rangle) |\, [\![Q]\!])
\end{aligned}
$$

The basic idea is that instead of sending a binary abstraction, $\langle X_1, X_2\rangle E$, to be applied on concrete parameters, $\langle B_1, B_2\rangle$, in the receiving environment, we send a special unary abstraction, $\langle Z\rangle \overline{m} Z$ that is a trigger [17, 18] carrying a pointer (trigger name) m to the process E, and it is then used by the receiver to achieve

the effect of the original application by successively sending the parameters, B_1 and B_2, to process E to instantiate its variables, X_1 and X_2, in the form of input prefixes. The technical procedure is:
– Firstly, the process $g.(\overline{f}|\,\overline{h}) + f.([\![B_1]\!]\,|\,\overline{f})$ is transmitted along the pointer m to instantiate the variable X_1 in $X_1|\,m(X_2).[\![E]\!]$, in which the first X_1 other than those possibly occurring in E now acts as an initiator of the next step;
– An internal communication happens on local name g, releasing $\overline{f}|\,\overline{h}$, which separately activates two different procedures. The one through $\overline{f}$ communicates with f in other places of E where X_1 once occurs and brings about a due instance of B_1, and this continues because another $\overline{f}$ is produced in the meanwhile. The one through $\overline{h}$ activates the remaining sub-process $h.Y\langle[\![B_2]\!]\rangle$ to transmit the second parameter B_2 to E to instantiate its second variable X_2. The communications on f and h may interleave, but it would not have any negative effect on the final result since they would never intervene with each other.

Notice we need two copies of Y (which renders the encoding process non-linear) to maintain the order between the two arguments to be instantiated. The protocol appears not possible to simply further, at least in its rationale. It could be if we had operation like $Y\langle B_1\rangle.Y\langle B_2\rangle$, but we do not.

Example Suppose $P|\,Q \in \Pi_2^{Dl}$ and $P \stackrel{def}{=} \overline{a}[\langle X_1, X_2\rangle(A|\,X_1|\,X_2)].P_1$, $Q \stackrel{def}{=} a(Y).(Y\langle B_1, B_2\rangle|\,Q_1)$, so $Y \notin fv(Q_1)$, $X_i \notin fv(A), i = 1, 2$. We have

$$\begin{aligned}
P|\,Q &\xrightarrow{\tau} P_1|\,\langle X_1, X_2\rangle(A|\,X_1|\,X_2)\langle B_1, B_2\rangle|\,Q_1 \equiv P_1|\,Q_1|\,A|\,B_1|\,B_2\\
[\![P]\!] &\stackrel{def}{=} (m)(\overline{a}[\langle Z\rangle\overline{m}Z].[\![P_1]\!]\,|\,m(X_1).(X_1|\,m(X_2).(X_1|\,X_2|\,[\![A]\!])))\\
[\![Q]\!] &\stackrel{def}{=} a(Y)((fgh)(\overline{g}|\,Y\langle g.(\overline{f}|\,\overline{h}) + f.([\![B_1]\!]\,|\,\overline{f})\rangle|\,h.Y\langle[\![B_2]\!]\rangle)|\,[\![Q_1]\!])
\end{aligned}$$

f, g, h should be fresh to secure the correct activation of the variables defined in the original abstraction. The encoding of $P|\,Q$ and its transitions are as below. T is defined as $g.(\overline{f}|\,\overline{h}) + f.([\![B_1]\!]\,|\,\overline{f})$.

$$\begin{aligned}
[\![P|\,Q]\!] &= [\![P]\!]\,|\,[\![Q]\!]\\
&\xrightarrow{\tau} \equiv (mfgh)([\![P_1]\!]\,|\,m(X_1).(X_1|\,m(X_2).(X_1|\,X_2|\,[\![A]\!]))|\,\overline{g}|\\
&\quad (\langle Z\rangle\overline{m}Z)\langle g.(\overline{f}|\,\overline{h}) + f.([\![B_1]\!]\,|\,\overline{f})\rangle|\,h.(\langle Z\rangle\overline{m}Z)\langle[\![B_2]\!]\rangle|\,[\![Q_1]\!])\\
&\equiv (mfgh)([\![P_1]\!]\,|\,m(X_1).(X_1|\,m(X_2).(X_1|\,X_2|\,[\![A]\!]))|\,\overline{g}|\\
&\quad \overline{m}[g.(\overline{f}|\,\overline{h}) + f.([\![B_1]\!]\,|\,\overline{f})]|\,h.\overline{m}[[\![B_2]\!]]|\,[\![Q_1]\!])\\
&\xrightarrow{\tau} \equiv (mfgh)([\![P_1]\!]\,|\,(g.(\overline{f}|\,\overline{h}) + f.([\![B_1]\!]\,|\,\overline{f}))|\\
&\quad m(X_2).(T|\,X_2|\,[\![A]\!])|\,\overline{g}|\,h.\overline{m}[[\![B_2]\!]]|\,[\![Q_1]\!])\\
&\xrightarrow{\tau} \equiv (mfgh)([\![P_1]\!]\,|\,(\overline{f}|\,\overline{h})|\,m(X_2).(T|\,X_2|\,[\![A]\!])|\,h.\overline{m}[[\![B_2]\!]]|\,[\![Q_1]\!])\\
&\quad \xrightarrow{\tau} (mfgh)([\![P_1]\!]\,|\,\overline{f}|\,m(X_2).(T|\,X_2|\,[\![A]\!])|\,\overline{m}[[\![B_2]\!]]|\,[\![Q_1]\!])\\
&\quad \xrightarrow{\tau} (mfgh)([\![P_1]\!]\,|\,\overline{f}|\,T|\,[\![B_2]\!]\,|\,[\![A]\!]\,|\,[\![Q_1]\!])\\
&\xrightarrow{\tau} \equiv (mfgh)([\![P_1]\!]\,|\,[\![B_1]\!]\,|\,\overline{f}|\,[\![B_2]\!]\,|\,[\![A]\!]\,|\,[\![Q_1]\!])\\
&\quad = [\![P_1]\!]\,|\,[\![Q_1]\!]\,|\,[\![A]\!]\,|\,[\![B_1]\!]\,|\,[\![B_2]\!]\\
&\quad \equiv [\![P_1|\,Q_1|\,A|\,B_1|\,B_2]\!]
\end{aligned}$$

We have the complete encoding defined on processes in Figure 3. For a Π_2^{Dl} process, applications of forms other than $Y\langle B_1, B_2\rangle$ are immediately firable and

$$
\begin{array}{rcll}
[\![0]\!] & \stackrel{def}{=} & 0 & \\
[\![X]\!] & \stackrel{def}{=} & X & \\
[\![a(Y).P]\!] & \stackrel{def}{=} & a(Y).[\![P]\!] & \\
[\![\overline{a}F.P]\!] & \stackrel{def}{=} & \begin{cases} \overline{a}[\![F]\!].[\![P]\!], & \text{if } F \text{ is non-parameterized} \\ (m)(\overline{a}[\langle Z\rangle \overline{m}Z].[\![P]\!] \,|\, [\![F]\!]), & \text{if } F \equiv \langle X_1, X_2\rangle R, \\ & \text{where } [\![F]\!] \text{ is } m(X_1).(X_1 \,|\, m(X_2).[\![R]\!]) \end{cases} \\
[\![P \,|\, Q]\!] & \stackrel{def}{=} & [\![P]\!] \,|\, [\![Q]\!] & \\
[\![(c)P]\!] & \stackrel{def}{=} & (c)[\![P]\!] & \\
[\![Y\langle B_1, B_2\rangle]\!] & \stackrel{def}{=} & (fgh)(\overline{g} \,|\, Y\langle g.(\overline{f} \,|\, \overline{h}) + f.([\![B_1]\!] \,|\, \overline{f})\rangle \,|\, h.Y\langle [\![B_2]\!]\rangle) & \\
 & & & \text{where } f, g, h \notin fn(Y\langle B_1, B_2\rangle)
\end{array}
$$

Fig. 3. Encoding from Π_2^{Dl} to Π_1^D

become structurally equivalent to a not (outmost) parameterized process. Hence without loss of generality we can assume all the remaining applications are of the above form, i.e. $Y\langle B_1, B_2\rangle$, and thus we focus on it in the encoding strategy. Apart from the core parts depicted above, the rest is structural. Notice in the encoding of output, it suffices to have only one copy of the encoding of the communicated process F, since it may be used only once in the receiving environment. The encoding satisfies the static criteria of encodability (Definition 2). The definition can be extended to contexts by leaving the holes as they are and the rest is nearly the same.

3.2 Properties of the encoding

Establishing the operational correspondence between the encoded Π_2^{Dl} process and its encoding counterpart in Π_1^D process is the first step toward the soundness and completeness of the encoding, that is the full abstraction property underneath the correctness of the encoding strategy.

Operational correspondence The following lemma states the relationship from the actions before to those after the encoding. The definition of $=$ is the one given in Section 2.2. The proof can be found in Appendix A.1.

Lemma 1 (Forth). *Let P be a Π_2^{Dl} process.*

(1.1) If $P \xrightarrow{a(A)} P'$ and A is non-parameterized, then $[\![P]\!] \xrightarrow{a([\![A]\!])} T = [\![P']\!]$;

(1.2) If $P \xrightarrow{a(G)} P' \equiv F\{G/Y\}$ for some F and $G \equiv \langle X_1, X_2\rangle E$, then $[\![P]\!] \xrightarrow{a(H)} T = [\![F]\!]\{H/Y\}$ and $H \equiv \langle Z\rangle \overline{m}Z$;

(2.1) If $P \xrightarrow{(\widetilde{c})\overline{a}E} P'$ and E is non-parameterized, then $[\![P]\!] \xrightarrow{(\widetilde{c})\overline{a}[\![E]\!]} [\![P']\!]$;

(2.2) If $P \xrightarrow{(\widetilde{c})\overline{a}[\langle X_1, X_2\rangle E]} P'$, then $[\![P]\!] \xrightarrow{(m)\overline{a}[\langle Z\rangle \overline{m}Z]} T = (\widetilde{c})([\![P']\!] \,|\, [\![\langle X_1, X_2\rangle E]\!])$;

(3) If $P \xrightarrow{\tau} P'$, then $[\![P]\!] \stackrel{\tau}{\Longrightarrow} T = [\![P']\!]$.

The following lemma is the converse of Lemma 1.

Lemma 2 (Back). *Let P be a Π_2^{Dl} process.*

(1.1) If $[\![P]\!] \xrightarrow{a(T')} T$ and T' is non-parameterized, then $P \xrightarrow{a(A)} P'$, $[\![A]\!] = T'$ and $T = [\![P']\!]$;

(1.2) If $[\![P]\!] \xrightarrow{a(H)} T \equiv S\{H/Y\}$ and $H \equiv \langle Z\rangle \overline{m} Z$, then $P \xrightarrow{a(G)} P' \equiv F\{G/Y\}$ for some F and $G \equiv \langle X_1, X_2\rangle E$, and $S = [\![F]\!]$;

(2.1) If $[\![P]\!] \xrightarrow{(\widetilde{c})\overline{a}T'} T$ and T' is non-parameterized, then $P \xrightarrow{(\widetilde{c})\overline{a}E}$, $[\![E]\!] = T'$ and $[\![P']\!] = T$;

(2.2) If $[\![P]\!] \xrightarrow{(m)\overline{a}[\langle Z\rangle \overline{m} Z]} T$, then $P \xrightarrow{(\widetilde{c})\overline{a}[\langle X_1, X_2\rangle E]} P'$ and $T = (\widetilde{c})([\![P']\!] \mid [\![\langle X_1, X_2\rangle E]\!])$;

(3) If $[\![P]\!] \xrightarrow{\tau} T$, then $P \xrightarrow{\tau} P'$, and $T \Longrightarrow \cdot = [\![P']\!]$ (or simply $T = [\![P']\!]$).

The proof is symmetric to Lemma 1 and can be done in a similar way. The difference lies in the induction steps. Note Lemma 1 and Lemma 2 still hold even if we replace the strong transition $\xrightarrow{\lambda}$ with the weak version $\stackrel{\lambda}{\Longrightarrow}$. The proofs are based on the strong version, using (routine) induction on the transitions of the encoding process. For convenience, we will simply point to these two lemmas. A similar case occurs in characterizing the encoding in section 4.

Soundness and completeness Intuitively, soundness says equal processes are translated into equal ones, whereas completeness indicates no unequal processes will be translated into equal ones. Suppose P, Q are Π_2^{Dl} processes.

Lemma 3 (Soundness). *$P = Q$ implies $[\![P]\!] = [\![Q]\!]$.*

Lemma 4 (Completeness). *$[\![P]\!] = [\![Q]\!]$ implies that $P = Q$.*

Notice $=$ denotes the context bisimilarity of corresponding calculi. The soundness of the encoding is the most involved part in showing the correctness. The proof of Lemma 3 is placed in Appendix A.1. In general, the completeness is relatively less tricky to deal with, and the proof of Lemma 4 can be conducted in a similar and easier way than that of Lemma 3, so we skip the details.

All the discussions and proofs above boil down to the satisfaction of the dynamic conditions (Definition 2), which also states essentially that the encoding fulfills the criteria for a correct encoding, as the following proposition states. The satisfying of diverge-reflecting follows from the observation that our encoding does not introduce divergence, i.e., Lemma 1 and Lemma 2.

Proposition 1. *There is an encoding from Π_2^{Dl} to Π_1^{D} meeting the criteria given in Section 2.*

Remark 1. One point about this section worthy of emphasis is that we focus on second-order processes, which does away with the naive (currying) strategy: translating $\langle X_1, X_2, ..., X_n\rangle P$ as $\langle X_1\rangle ... \langle X_n\rangle P$. As mentioned in the introduction, it is a more desirable programming style. Another point worth mentioning is that the result of this section leads to several relevant questions: (1) $\Pi_{n+1}^{Dl} \sqsubseteq \Pi_n^{Dl}$? (2) $\Pi_{n+1}^{D} \sqsubseteq \Pi_n^{Dl}$? (3) $\Pi_{n+1}^{D} \sqsubseteq \Pi_n^{D}$? Our intuition is that the first two of them are negative, while the last one seems positive. We leave their examination as future work. More related discussion is given in Section 5.

4 On the relationship between $\Pi_n^{D,d}$ and Π_n^d

In this section, we show that $\Pi_n^{D,d} \sqsubseteq \Pi_n^d$. The encoding borrows idea from that of translating a higher-order process to a first-order one [17]. Recall $\Pi_n^{D,d}$ is the union of Π_n^D and Π_n^d. We prove that $\Pi_1^{D,d} \sqsubseteq \Pi_1^d$ instead of the general case, which can be extended to in a straightforward way.

4.1 Encode $\Pi_1^{D,d}$ in Π_1^d

The encoding function from $\Pi_1^{D,d}$ to Π_1^d is a homomorphism except for higher-order abstraction and higher-order application, as shown below.

$$\begin{aligned}[\![\langle X\rangle P]\!] &\stackrel{def}{=} \langle x\rangle[\![P]\!]\{\bar{x}/X\} \quad x \text{ is fresh}\\ [\![P\langle A\rangle]\!] &\stackrel{def}{=} (m)([\![P]\!]\langle m\rangle \,|\, !m.[\![A]\!])\end{aligned}$$

Intuitively, a higher-order-parameterized process $\langle X\rangle P$ is encoded by a first-order-parameterized process (on variable x) and then replacing the parameter X with a trigger $\bar{x}$ (i.e. $\bar{x}0$), which is 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 higher-order-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 $P\langle Q\langle R\rangle\rangle \in \Pi_1^{D,d}$ in which $P \stackrel{def}{=} \langle X\rangle(X|X|a(Y).Y)$, $Q \stackrel{def}{=} \langle Z\rangle(Z)$ and $R \stackrel{def}{=} \bar{a}0$. Then $P\langle Q\langle R\rangle\rangle \xrightarrow{\bar{a}0} \bar{a}0|a(Y).Y$. The encoding process behaves as below.

$$\begin{aligned}[\![P\langle Q\langle R\rangle\rangle]\!] &\equiv (m)(\ \langle x\rangle(\bar{x}|\bar{x}|a(Y).Y)\langle m\rangle \mid !m.(n)((\langle z\rangle\bar{z})\langle n\rangle|!n.\bar{a}0)\ )\\ &\xrightarrow{\tau} \equiv (m)(\ (\bar{m}|a(Y).Y) \mid (n)((\langle z\rangle\bar{z})\langle n\rangle|!n.\bar{a}0)|!m.(n)((\langle z\rangle\bar{z})\langle n\rangle|!n.\bar{a}0)\ )\\ &\xrightarrow{\tau} \equiv (m)(\ (\bar{m}|a(Y).Y) \mid \bar{a}0 \mid !m.(n)((\langle z\rangle\bar{z})\langle n\rangle|!n.\bar{a}0)\ )\\ &\xrightarrow{\bar{a}0} (m)(\ (\bar{m}|a(Y).Y) \mid !m.(n)((\langle z\rangle\bar{z})\langle n\rangle|!n.\bar{a}0)\ )\\ &= [\![\bar{a}0|a(Y).Y]\!]\end{aligned}$$

4.2 Properties of the encoding

In order to show the soundness and completeness property of the encoding we start by establishing the operational correspondence between a $\Pi_1^{D,d}$ process and its encoding in Π_1^d. Again $=$ stands for context bisimilarity.

Lemma 5 (Forth). *Let P be a $\Pi_1^{D,d}$ process.*
1. Strong correspondence:
(1) If $P \xrightarrow{a(A)} P'$ then $[\![P]\!] \Longrightarrow \xrightarrow{a([\![A]\!])} T = [\![P']\!]$; (2) If $P \xrightarrow{(\widetilde{c})\bar{a}A} P'$, then $[\![P]\!] \Longrightarrow \xrightarrow{(\widetilde{c})\bar{a}[\![A]\!]} T = P'$; (3) If $P \xrightarrow{\tau} P'$, then $[\![P]\!] \Longrightarrow \xrightarrow{\tau} T = [\![P']\!]$.

2. Weak correspondence:
(1) If $P \overset{a(A)}{\Longrightarrow} P'$ *then* $[\![P]\!] \overset{a([\![A]\!])}{\Longrightarrow} T = [\![P']\!]$*; (2) If* $P \overset{(\widetilde{c})\bar{a}A}{\Longrightarrow} P'$*, then* $[\![P]\!] \overset{(\widetilde{c})\bar{a}[\![A]\!]}{\Longrightarrow} T = P'$*; (3) If* $P \Longrightarrow P'$*, then* $[\![P]\!] \Longrightarrow T = [\![P']\!]$*.*

The strong transition part of lemma 5 can be proven by induction on the derivation of the transition; the weak transition part is a corollary of the strong transition part. The proof is placed in Appendix A.2. Using similar analysis we are able to prove the backward operational correspondence.

Lemma 6 (Back). *Let* P *be a* $\Pi_1^{D,d}$ *process.*
1. Strong correspondence:
(1) If $[\![P]\!] \xrightarrow{a(A)} T$*, then* $P \xrightarrow{a(A)} P'$ *and* $T = [\![P']\!]$*; (2) If* $[\![P]\!] \xrightarrow{(\widetilde{c})\bar{a}A} T$*, then there exists some* A' *s.t.* $P \xrightarrow{(\widetilde{c})\bar{a}A'} P'$ *,* $T = [\![P']\!]$ *and* $[\![A']\!] \equiv A$*; (3) If* $[\![P]\!] \xrightarrow{\tau} T$*, then either* $T = [\![P]\!]$ *or* $P \xrightarrow{\tau} P'$ *and* $T = [\![P']\!]$*.*
2. Weak correspondence:
(1) If $[\![P]\!] \overset{a(A)}{\Longrightarrow} T$*, then* $P \overset{a(A)}{\Longrightarrow} P'$ *and* $T = [\![P']\!]$*; (2) If* $[\![P]\!] \overset{(\widetilde{c})\bar{a}A}{\Longrightarrow} T$*, then exists some* A' *s.t.* $P \overset{(\widetilde{c})\bar{a}A'}{\Longrightarrow} P'$ *,* $T = [\![P']\!]$ *and* $[\![A']\!] \equiv A$*; (3) If* $[\![P]\!] \Longrightarrow T$*, then* $P \Longrightarrow P'$ *and* $T = [\![P']\!]$*.*

With the help of lemma 5 and lemma 6 we can establish the soundness and completeness property of the encoding. Suppose P, Q are $\Pi_1^{D,d}$ processes.

Lemma 7 (Soundness). $P = Q$ *implies* $[\![P]\!] = [\![Q]\!]$*.*

Lemma 8 (Completeness). $[\![P]\!] = [\![Q]\!]$ *implies* $P = Q$*.*

The proof of the soundness is placed in Appendix A.2. Likewise, we omit the proof of the completeness since it uses a similar approach and is easier. Since the satisfaction of diverge-reflecting follows straightforward from Lemma 7 and Lemma 8, we arrive at the following proposition.

Proposition 2. *There is an encoding from* $\Pi_1^{D,d}$ *to* Π_1^d *meeting the criteria given in Section 2.*

Remark 2. It should be stressed that the basic encoding strategy from $\Pi_n^{D,d}$ to Π_n^d employs the idea of triggers. This makes the proof of the correctness of the strategy somewhat akin to that of encoding Π into π [17, 30]. However the fact is that the technical proof is not so trivial as expected. The reason is that when proving the soundness of the encoding, the corresponding bisimulation relation is hard to design, in particular concerning the case of output, due to the difficulty in finding the original image of certain encoding process. This is why we use $\Pi_n^{D,d}$ instead of Π_n^D. We believe $\Pi_n^D \sqsubseteq \Pi_n^d$ can be obtained from $\Pi_n^D \sqsubseteq \Pi_n^{D,d}$ and transitivity by a fine reasoning on the controlling of contexts (and potentially some proof technique like up-to technique [22]). This would be an interesting further work. A related question is whether there is an encoding from Π_n^d to Π_n^D. One may think of trying a similar strategy than that of $\pi \sqsubseteq \Pi^r$ (Π with relabelling operator) [24, 25]. Unfortunately, this does not seem to work out, as far as we are concerned. Section 5 provides more related further work.

5 Conclusion

This paper contributes toward clarifying the expressiveness of parameterization operation in higher-order processes (in a second-order setting). The first is that in process parameterization Π_n^D is able to encode Π_{n+1}^{Dl}. This shows that decreasing the arity of process parameterization somewhat correlates with the absence of linearity. The second is process parameterization does not surpass name parameterization in expressiveness. This emphasizes the intuition that name-handling is more basic than process-handling in process models. We have mentioned some further work concerning the first result in Section 3. For the second result, the generalization to arbitrary-order processes may be an interesting further job worth investigation. The framework of the proofs would be useful for that further study and other related work on similar calculi.

Figure 4 concludes from several aspects the expressiveness of the related calculi. π represents the first-order pi-calculus [15], and "$\longrightarrow$" means there exists an encoding while "$\text{—}\times\!\!\rightarrow$" means the opposite, and "$\text{—}?\!\!\rightarrow$" means open.

(1) The positive. The arrows respectively from $\Pi_n^{D,d}$, Π_n^d to π are from [17,18]. The arrow from $\Pi_n^{D,d}$ to Π_n^d is from this paper. The arrows from Π respectively to $\Pi_n^{D,d}$, Π_n^d naturally hold.

(2) The negative. That $\Pi_n^{D,d}$ is not encodable in Π is from [10]. A corollary is Π_n^d is not encodable in Π. This exhibits that first-order abstraction can strictly lift the expressiveness of Π.

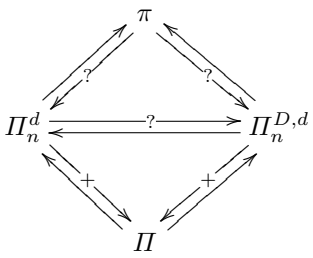

Fig. 4. Expressiveness relations

(3) There are three question marks that denotes uncertainty. One is on the arrow from π to Π_n^d (another relevant is from π to $\Pi_n^{D,d}$). To our intuition, the encoding, if any, is somewhat akin to that in [24]; some initial work is in [28]. The question mark on the arrow from Π_n^d to $\Pi_n^{D,d}$ may be more difficult than it may appear, as mentioned. The reason is that the contexts in the two calculi are not the same, so the soundness is unlikely to be obtained trivially, but we believe it can be achieved. These would be interesting future work.

There are two more potential directions deserving further consideration.

- One conjecture not shown in Figure 4 is Π_n^D cannot encode Π_n^d, based on which the nonexistence of an encoding from π to Π_n^D follows by contraposition. Intuitively, the crux is that the role of a (parameterized) name cannot be faithfully translated by a (parameterized) process, because the (auxiliary) actions accountable for switching the role of a name (input or output) have no way to stay secret, toward the soundness of a correct encoding. Moreover, with static binding, a context of Π^D has no way to dynamically form any channel without resorting to some global names previously agreed upon.
- The languages in which we study parameterization in this paper are small and "abstract"; it would be most interesting to explore parameterization in some "less abstract" languages, such as those accommodating asynchronous

communication [7] [16], reversibility and compensation [9], explicit locations [6]. These more practical features are increasingly common in modern distributed (concurrent) systems.

Related Work Here we review more relevant work in the literature that motivates ours here.

- Some relatively recent work is by Lanese et al. [10], which inspires our work. Particularly, in [10] it is shown that the passing of process abstractions is strictly more expressive than mere process-passing. This potentially offers an approach to elevate the expressiveness of the basic calculus Π, as well as a few important techniques of handling abstractions. It is proven based on another contribution $HO_{n+1} \not\sqsubseteq HO_n$, where HO_n denotes $n-$ary polyadic Π calculus. This pinpoints a striking difference from name-passing processes. An exposed question after these results is what could be the limit of the increasing chain HO_n. The fact that the extension of process abstractions is more powerful than process-passing alone presents little hope that expressiveness could be enhanced *greatly* through allowing more processes to be sent in one communication. In addition, a thought-provoking encoding scheme from HO_n to Π^D (actually only Π^D_1) is given and discussed. In these results of [10], however, not so much concentration is put on the intrinsic characteristic of process abstraction, and no name abstraction is studied. It would be interesting and worthwhile to know more about the *inside* of process abstraction, as well as their relationship with name abstraction, which could also reveal more essence about name abstraction. The work of this paper originates from these ideas.
- In earlier work on the expressiveness of higher-order processes, Sangiorgi provides a framework and some key techniques (e.g. normal bisimulation and triggers) for dealing with higher-order processes [19] and for compiling Π (extended with abstractions) into π [17, 18]. The correctness proofs for the encodings are discussed without explicit criteria for qualified encodings (due to chronological reason); some recent work is from [27,30]. Furthermore in [20], a hierarchy of higher-order calculi based on the order of processes are shown to correspond to the hierarchy of πI based on similar variation on names. πI is a variant of π that can only engage private communications. The extensive study of the resulting hierarchy of these calculi and their relative expressiveness are quite instructive to later work. One knows from these work that name-passing appears more basic than program-passing.
- In [2] (later improved in [1]), a variant of Π (Homer) is put forward and shown to be able to encode π. Albeit the semantics of the capability of continuation-passing seems rather strong and few explicit criteria is resorted to, one insight from the result is obvious, i.e. Π is not reasonably expressive if not extended with more serviceable operator. In [24], Thomsen puts forth a strategy of translating π into Π^r (Π with the relabelling operator). Later more analysis is given [25]. However these results are still debatable because relabelling is criticized by many researchers for its excessively strong semantics. Yet abstraction is usually considered more reasonable in semantics

and easy to control in practice. So it is good to know that abstraction is a proper operation that may promote process-passing in expressiveness. Moreover, it is not necessary to worry about computational capability, because Π is shown to be Turing complete in [11, 12]. More about the computation capacity of higher-order processes can be found in [3, 8, 26].

Acknowledgement The authors thank the members of the BASICS lab and the anonymous referees for their comments and suggestions.

References

1. Bundgaard, M., Godskesen, J.C., Hildebrandt, T.: Encoding the pi-calculus in higher-order calculi. Tech. Rep. TR-2008-106, IT University of Copenhagen (2008)
2. Bundgaard, M., Hildebrandt, T., Godskesen, J.C.: A cps encoding of name-passing in higher-order mobile embedded resources. Theoretical Computer Science 356, 422–439 (2006)
3. Giusto, C.D., Pérez, J.A., Zavattaro, G.: On the expressiveness of forwarding in higher-order communication. In: Proceedings of the 6th International Colloquium on Theoretical Aspects of Computing (ICTAC '09). vol. LNCS 5684, pp. 155–169 (2009)
4. Gorla, D.: Towards a unified approach to encodability and separation results for process calculi. In: Proceedings of CONCUR' 08. LNCS, vol. 5201, pp. 492–507 (2008), journal version in [5]
5. Gorla, D.: Towards a unified approach to encodability and separation results for process calculi. Information and Computation 208(9), 1031–1053 (2010)
6. Hennessy, M.: A Distributed Pi-Calculus. Cambridge University Press (2007)
7. Honda, K., Tokoro, M.: An object calculus for asynchronous communication. In: America, P. (ed.) Proceedings of the European Conference on Object-Oriented Programming (ECOOP). vol. 512, pp. 133–147. Springer-Verlag (1991)
8. Lago, U.D., Martini, S., Sangiorgi, D.: Light logics and higher-order processes. In: Proceedings of Workshop on Expressiveness in Concurrency 2010 (EXPRESS 2010). vol. EPTCS 41, pp. 46–60 (2010)
9. Lanese, I., Lienhardt, M., Mezzina, C.A., Schmitt, A., Stefani, J.B.: Concurrent flexible reversibility. In: Proceedings of ESOP 2013. LNCS, vol. 7792, pp. 370–390 (2013)
10. Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness of polyadic and synchronous communication in higher-order process calculi. In: Proceedings of ICALP 2010. pp. 442–453. LNCS, Springer Verlag (2010)
11. Lanese, I., Pérez, J.A., Sangiorgi, D., Schmitt, A.: On the expressiveness and decidability of higher-order process calculi. Information and Computation 209(2), 198–226 (2011)
12. Lanese, I., Perez, J., Sangiorgi, D., Schmitt, A.: 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). pp. 293–302 (2008)
13. Meyer, R., Khomenko, V., Strazny, T.: 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). LNCS, vol. 5062, pp. 327–347 (2008)

14. Milner, R.: Functions as processes. Journal of Mathematical Structures in Computer Science 2(2), 119–141 (1992), research Report 1154, INRIA, Sofia Antipolis, 1990
15. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes (parts i and ii). Information and Computation 100(1), 1–77 (1992), academic Press
16. Palamidessi, C.: Comparing the expressive power of the synchronous and the asynchronous pi-calculus. Mathematical Structures in Computer Science 13, 685–719 (2003)
17. Sangiorgi, D.: Expressing Mobility in Process Algebras: First-order and Higher-order Paradigms. Phd thesis, University of Edinburgh (1992)
18. Sangiorgi, D.: From π-calculus to higher-order π-calculus—and back. In: Proceedings TAPSOFT '93. LNCS, vol. 668, pp. 151–166. Springer Verlag (1992), orsay, France, April 13-17
19. Sangiorgi, D.: Bisimulation for higher-order process calculi. Information and Computation 131(2), 141–178 (1996), preliminary version in proceedings PROCOMET'94 (IFIP Working Conference on Programming Concepts, Methods and Calculi), pages 207-224 ,North Holland, 1994
20. Sangiorgi, D.: Pi-calculus, internal mobility and agent-passing calculi. Theoretical Computer Science 167(2), 235–274 (1996), extracts of parts of the material contained in this paper can be found in the Proceedings of TAPSOFT'95 and ICALP'95
21. Sangiorgi, D.: Introduction to Bisimulation and Coinduction. Cambridge University Press (2011)
22. Sangiorgi, D., Walker, D.: The Pi-calculus: a Theory of Mobile Processes. Cambridge Universtity Press (2001)
23. Thomsen, B.: Calculi for Higher Order Communicating Systems. Phd thesis, Department of Computing, Imperial College (1990)
24. Thomsen, B.: Plain chocs, a second generation calculus for higher-order processes. Acta Informatica 30(1), 1–59 (1993)
25. Xu, X.: Expressing first-order π-calculus in higher-order calculus of communicating systems. Journal of Computer Science and Technology 24(1), 122–137 (2009)
26. Xu, X.: On bisimulation theory in linear higher-order pi-calculus. Transactions on Petri Nets and Other Models of Concurrency III 5800, 244–274 (2009)
27. Xu, X.: Distinguishing and relating higher-order and first-order processes by expressiveness. Acta Informatica 49, 445–484 (2012)
28. Xu, X.: On the expressiveness of higher-order processes with name parameterization. Tech. rep., East China University of Science and Technology (2012), presented and discussed at the 4th NSFC-JSPS Joint Workshop on Formal Methods (Nara Japan) 2011
29. Xu, X.: On context bisimulation for parameterized higher-order processes. In: Proceedings of 6th Interaction and Concurrency Experience (ICE 2013), Satellite workshop of DisCoTec 2013 (2013), to appear in EPTCS
30. Yin, Q., Long, H.: Process passing calculus, revisited. Journal of Shanghai Jiaotong University (Science) 18, 29–36 (2013)

Appendix

A Proofs

A.1 Proofs for Section 3

Proof of Lemma 1

Proof. (1.1) and (2.1) are trivial according to the encoding. (3) becomes easy in a straightforward way from (1.1), (1.2), (2.1), (2.2). The most critical parts are (1.2) and (2.2). Since their proof strategies are somewhat similar, we focus on (1.2). To prove that clause, we need to undergo an induction on the structure of P. Thus there are several cases: 0, $a(Y).P_1$, $\overline{a}E.P_1$, $P_1|\,Q_1$, $(c)P_1$, $\langle X_1, X_2\rangle P_1$, $P_1\langle B_1, B_2\rangle$. We will expand the analysis on three of them, and the rest are similar or easier.

- $P \equiv a(Y).P_1$. In this case $P \xrightarrow{a(G)} P_1\{G/Y\}$ and $F \equiv P_1$. According to the encoding, $[\![P]\!] \equiv a(Y).[\![P_1]\!] \xrightarrow{a(H)} [\![P_1]\!]\{H/Y\} \equiv T$, in which $H \equiv \langle Z\rangle\overline{m}Z$. Since $F \equiv P_1$, we immediately have $T = [\![F]\!]\{H/Y\}$.
- $P \equiv P_1\langle B_1, B_2\rangle$. The only countable case is $P \equiv Y\langle B_1, B_2\rangle$, when $[\![P]\!] \equiv (fgh)(\overline{g}|\,Y\langle g.(\overline{f}|\,\overline{h}) + f.([\![B_1]\!]|\,\overline{f})\rangle|\,h.Y\langle[\![B_2]\!]\rangle)$. So this case is impossible.
- $P \equiv P_1|\,Q_1$. W.l.o.g., suppose $P_1 \xrightarrow{a(G)} P_1' \equiv F_1\{G/Y\}$ for some F_1 and Y is fresh. So $P \equiv P_1|\,Q_1 \xrightarrow{a(G)} P_1'|\,Q_1 \equiv F_1\{G/Y\}|\,Q_1 \equiv F\{G/Y\}$ in which $F \stackrel{def}{=} F_1|\,Q_1$. By induction hypothesis, $[\![P_1]\!] \xrightarrow{a(H)} T_1 = [\![F_1]\!]\{H/Y\}$ where $H \equiv \langle Z\rangle\overline{m}Z$. Thus

$$\begin{aligned}[\![P]\!] \equiv [\![P_1]\!]|\,[\![Q_1]\!] \xrightarrow{a(H)} T_1|\,[\![Q_1]\!] &= [\![F_1]\!]\{H/Y\}|\,[\![Q_1]\!] \\ &= ([\![F_1]\!]|\,[\![Q_1]\!])\{H/Y\} \\ &= ([\![F_1|\,Q_1]\!])\{H/Y\} \\ &= ([\![F]\!])\{H/Y\}\end{aligned}$$

□

Proof of Lemma 3

Proof. Based on the operational correspondence, our target is to prove $P = Q$ implies $[\![P]\!] = [\![Q]\!]$, through showing

$$\mathcal{R} \stackrel{def}{=} \{([\![P]\!], [\![Q]\!]) \mid P = Q\} \quad \text{which is closed under (variable) substitution}$$

is a normal bisimulation up-to $=$ (see [19] [22] for up-to techniques). Remember "closed under (variable) substitution" means $P\{A/X\}\ \mathcal{R}\ Q\{A/X\}$ for every A whenever $P\mathcal{R}Q$ and they are (open) processes with occurrence of variable X. Notice in the calculus Π_n^D, normal bisimulation characterizes context bisimulation; see [17] [19] [29] for proof reference. In normal bisimulation, the communicated process only needs to be confined to triggers ($\langle Z\rangle\overline{m}Z.0$) and the testing environment in the output clause only needs to be $!n(X).[\,]\langle X\rangle$.

There are several cases needing analysis, among which the most crucial parts are concerning the communication of parameterized processes.

1. $[\![P]\!] \stackrel{a(H)}{\Longrightarrow} T$ in which $H \equiv \langle Z\rangle\overline{m}Z$;
2. $[\![P]\!] \stackrel{(m)\overline{a}T_1}{\Longrightarrow} T$ in which $T_1 \equiv \langle Z\rangle\overline{m}Z$;

3. $[\![P]\!] \stackrel{\tau}{\Longrightarrow} T$

We below proceed with their analysis.

– Case 1. From the premise, $T \equiv S\{H/Y\}$ for some S, and by Lemma 2

$$P \stackrel{a(G)}{\Longrightarrow} P' \equiv F\{G/Y\}$$

for some F and $G \equiv \langle X_1, X_2\rangle E$, and also $S = [\![F]\!]$. Because $P = Q$, Q can simulate P in the following way.

$$Q \stackrel{a(G)}{\Longrightarrow} Q' \equiv F'\{G/Y\}$$

for some F', and $P' = Q'$. Clearly we also have $F = F'$. By Lemma 1,

$$[\![Q]\!] \stackrel{a(H)}{\Longrightarrow} T' = [\![F']\!]\{H/Y\},$$

then

$$T = [\![F]\!]\{H/Y\} \mathcal{R} [\![F']\!]\{H/Y\} = T'$$

because $[\![F]\!] \mathcal{R} [\![F']\!]$ due to $F = F'$.

– Case 2. From the premise and Lemma 2,

$$P \stackrel{(\widetilde{c})\overline{a}[\langle X_1, X_2\rangle E]}{\Longrightarrow} P'$$

and $T = (\widetilde{c})([\![P']\!] \,|\, [\![\langle X_1, X_2\rangle E]\!])$. Since $P = Q$, Q can simulate P by

$$Q \stackrel{(\widetilde{d})\overline{a}G}{\Longrightarrow} Q'$$

and

$$(\widetilde{c})(P' \,|\, C[\langle X_1, X_2\rangle E]) = (\widetilde{d})(Q' \,|\, C[G]) \tag{1}$$

for every Π_2^D context $C[\,]$. There are two cases: whether G is a parameterization or not. Actually they can be similarly dealt with, so we only consider the case $G \equiv \langle X_1, X_2\rangle F$ which is a little more complicated. Then by Lemma 1,

$$[\![Q]\!] \stackrel{(m)\overline{a}[\langle Z\rangle \overline{m} Z]}{\Longrightarrow} T' = (\widetilde{d})([\![Q']\!] \,|\, [\![\langle X_1, X_2\rangle F]\!])$$

We intend to show

$$(m)(T \,|\, D[\langle Z\rangle \overline{m} Z]) \mathcal{R} (m)(T' \,|\, D[\langle Z\rangle \overline{m} Z])$$

for a Π_1^D context $D[\,] \equiv !n(X).[\,]\langle X\rangle$ in which n is fresh. Or equivalently

$$\begin{aligned}&(m)((\widetilde{c})([\![P']\!] \,|\, [\![\langle X_1, X_2\rangle E]\!]) \,|\, !n(X).\overline{m}X)\\ \mathcal{R}\ &(m)((\widetilde{d})([\![Q']\!] \,|\, [\![\langle X_1, X_2\rangle F]\!]) \,|\, !n(X).\overline{m}X)\end{aligned} \tag{2}$$

In terms of (1), it is obvious that when fixing $C[\,] \equiv (m)([\,]|\,!n(X).\overline{m}X)$ we have

$$(\widetilde{c})(P'|\,(m)(\langle X_1, X_2\rangle E|\,!n(X).\overline{m}X)) = (\widetilde{d})(Q'|\,(m)(\langle X_1, X_2\rangle F|\,!n(X).\overline{m}X))$$

which becomes the following after applying some structural adjustment

$$(m)((\widetilde{c})(P'|\,\langle X_1, X_2\rangle E)|\,!n(X).\overline{m}X) = (m)((\widetilde{d})(Q'|\,\langle X_1, X_2\rangle F)|\,!n(X).\overline{m}X) \tag{3}$$

Now it is not hard to see that the left and right side of $=$ in (3) are respectively encoded into the left and right side of $\mathcal{R}$ in (2). That virtually proves the validity of (2) by the definition of $\mathcal{R}$.

– Case 3. From the premise and Lemma 2,

$$P \stackrel{\tau}{\Longrightarrow} \text{ and } T \Longrightarrow \cdot = [\![P']\!] \text{ or simply } T = [\![P']\!]$$

Since $P = Q$, we have

$$Q \stackrel{\tau}{\Longrightarrow} Q' \text{ and } P' = Q'$$

Then by Lemma 1,

$$[\![Q]\!] \stackrel{\tau}{\Longrightarrow} T' = [\![Q']\!]$$

Thus

$$T = [\![P']\!]\ \mathcal{R}\ [\![Q']\!] = T' \text{ due to } P' = Q'$$

Now the proof is completed. □

A.2 Proofs for Section 4

Proof of lemma 5

Proof. We only prove the strong transition correspondence part. The proof proceeds by induction on the structure of P, we consider the following cases.

- Case 1. $P \equiv a(X).P_1$. P can only do an input action, suppose that $P \xrightarrow{a(A)} P_1\{A/X\}$, then we have $[\![P]\!] \equiv a(X).[\![P_1]\!] \xrightarrow{a([\![A]\!])} \equiv [\![P_1]\!]\{[\![A]\!]/X\} = [\![P_1\{A/X\}]\!]$.
- Case 2. $P \equiv P_1|P_2$. There are 3 subcases.
  1. $P \xrightarrow{(\widetilde{c})\bar{a}A} P'$. W.l.o.g. we can assume this is caused by $P_1 \xrightarrow{(\widetilde{c})\bar{a}A} P_1'$ and $P' \equiv P_1'|P_2$. Then by I.H. we have $[\![P_1]\!] \Longrightarrow \xrightarrow{(\widetilde{c})\bar{a}[\![A]\!]} T_1$ and $T_1 = [\![P_1']\!]$. As a result we have $[\![P]\!] \equiv [\![P_1]\!]|[\![P_2]\!] \Longrightarrow \xrightarrow{(\widetilde{c})\bar{a}[\![A]\!]} T_1|[\![P_2]\!]$ and $T_1|[\![P_2]\!] = [\![P_1']\!]|[\![P_2]\!] = [\![P']\!]$.
  2. $P \xrightarrow{a(A)} P'$ this is case is similar to the above.
  3. $P \xrightarrow{\tau} P'$. 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 $P' \equiv (c)(P_1'|P_2')$. By I.H. we have $[\![P_1]\!] \Longrightarrow \xrightarrow{a([\![A]\!])} T_1 = [\![P_1']\!]$ and $[\![P_2]\!] \Longrightarrow \xrightarrow{(\widetilde{c})\bar{a}[\![A]\!]} T_2 = [\![P_2']\!]$. As a result $[\![P]\!] \equiv [\![P_1]\!]|[\![P_2]\!] \stackrel{\tau}{\Longrightarrow} T$ and $T \equiv (\widetilde{c})(T_1|T_2) = (\widetilde{c})([\![P_1']\!]|[\![P_2']\!]) \equiv [\![P']\!]$.

- Case 3. $P \equiv \bar{a}A.P_1$ or $P \equiv (c)P_1$. This case is similar to case 2.
- Case 4. $P \equiv A\langle P_1\rangle$. Suppose that $A \equiv \langle X\rangle Q$ and $P \xrightarrow{\lambda} P'$ there are three subcases to consider: $\lambda = a(B)$, $\lambda = (\widetilde{c})\bar{a}E$ and $\lambda = \tau$. We only deal with the case that $\lambda = a(B)$, the others are similar. Consider the different cause of $P \xrightarrow{a(B)} P'$.
  1. If $P \xrightarrow{a(B)} P'$ is caused by Q, then we can assume that $Q\{P_1/X\} \xrightarrow{a(A)} P' \equiv Q_1\{P_1/X\}\{A/Y\}$ and $Q\{\bar{m}0/X\} \xrightarrow{a(A)} Q_1\{\bar{m}0/X\}\{A/Y\}$. By I.H. $[\![Q\{\bar{m}0/X\}]\!] \Longrightarrow \xrightarrow{a([\![A]\!])} T'$ and $T' = [\![Q_1\{\bar{m}0/X\}\{A/Y\}]\!]$. Thus we have $[\![P]\!] \equiv (m)([\![Q]\!]\{\bar{m}0/X\}|!m.[\![P_1]\!]) \Longrightarrow \xrightarrow{a([\![A]\!])} T$ and

$$T = (m)(T'|!m.[\![P_1]\!]) = (m)([\![Q_1\{\bar{m}0/X\}\{A/Y\}]\!]|!m.[\![P_1]\!]) = [\![P']\!]$$

  2. If $P \xrightarrow{a(B)} P'$ is caused by $P_1 \xrightarrow{a(A)} P_2$ then we have $Q \equiv Q'|X$ for some Q' , $Q\{P_1/X\} \equiv Q'\{P_1/X\}|P_1$ and $P' \equiv Q'\{P_1/X\}|P_2$. By I.H. $[\![P_1]\!] \Longrightarrow \xrightarrow{a([\![A]\!])} T'$ and $T' = [\![P_2]\!]$. Then

$$\begin{array}{rcl} [\![P]\!] & \equiv & (m)([\![Q]\!]\{\bar{m}0/X\}\langle m\rangle|!m.[\![P_1]\!]) \\ & \equiv & (m)([\![Q'|X]\!]\{\bar{m}0/X\}\langle m\rangle|!m.[\![P_1]\!]) \\ & \xrightarrow{\tau} & (m)([\![Q']\!]\{\bar{m}0/X\}\langle m\rangle|[\![P_1]\!]|!m.[\![P_1]\!]) \\ \Longrightarrow \xrightarrow{a([\![A]\!])} & & (m)([\![Q']\!]\{\bar{m}0/X\}\langle m\rangle|T'|!m.[\![P_1]\!]) \\ & = & (m)([\![Q']\!]\{\bar{m}0/X\}\langle m\rangle|[\![P_2]\!]|!m.[\![P_1]\!]) \\ & \equiv & [\![P']\!] \end{array}$$

□

Proof of lemma 7

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

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

is a context bisimulation up-to $=$. Suppose that $P \xRightarrow{\lambda} T$, there are three cases to consider.

- Case 1. $\lambda = a(A)$. By lemma 6, $P \xRightarrow{a(A)} P'$ and $T = [\![P']\!]$. Since $P = Q$, then Q can simulate P with $Q \xRightarrow{a(A)} Q' = P'$. By lemma 5 we know that $[\![Q]\!] \xRightarrow{a([\![A]\!])} T'$ and $T' = [\![Q']\!]$. As A is an Π_1^d agent, $[\![A]\!] \equiv A$. In conclusion we got $[\![Q]\!] \xRightarrow{a(A)} T'$ and

$$T = [\![P']\!]\mathcal{R}[\![Q']\!] = T'$$

- Case 2. $\lambda = (\widetilde{c})\bar{a}A$. By lemma 6, there exists some $\widetilde{c}$, A' s.t. $P \overset{(\widetilde{c})\bar{a}A'}{\Longrightarrow} P'$ with $T = [\![P']\!]$ and $[\![A']\!] \equiv A$. Since $P = Q$, then Q can simulate P with $Q \overset{(\widetilde{d})\bar{a}B'}{\Longrightarrow} Q'$ for some $\widetilde{d}$ and B' s.t. for every process $E[\,]$ with $\widetilde{c}\widetilde{d} \cap fn(E[\,]) = \emptyset$ it holds that

$$(\widetilde{c})(E[A']|\, P') = (\widetilde{d})(E[B']|\, Q') \qquad (*)$$

  By lemma 5 , there exist some T' s.t. $Q \overset{(\widetilde{d})\bar{a}[\![B']\!]}{\Longrightarrow} T'$ and $T' = [\![Q']\!]$. And from (*) we know that for every process $F[\,]$ without higher-order abstraction and higher-order application s.t. $\widetilde{c}\widetilde{d} \cap fn(F[\,]) = \emptyset$ we have

$$(\widetilde{c})(F[A]|\, [\![P']\!])\, \mathcal{R}\, (\widetilde{d})(F[B]|\, [\![Q]\!]) \quad (\text{let } B \equiv [\![B']\!])$$

  In conclusion we have $Q \overset{(\widetilde{d})\bar{a}B}{\Longrightarrow} T'$ and for every process $F[\,]$ s.t. $\widetilde{c}\widetilde{d} \cap fn(F[\,]) = \emptyset$ we have

$$T = (\widetilde{c})(F[A]|\, [\![P']\!])\, \mathcal{R}\, (\widetilde{d})(F[B]|\, [\![Q]\!]) = T'$$

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

□