The C_π-calculus: a Model for Confidential Name Passing

Ivan Prokić
Faculty of Technical Sciences, University of Novi Sad, Serbia

Sharing confidential information in distributed systems is a necessity in many applications, however, it opens the problem of controlling information sharing even among trusted parties. In this paper, we present a formal model in which dissemination of information is disabled at the level of the syntax in a direct way. We introduce a subcalculus of the π-calculus in which channels are considered as confidential information. The only difference with respect to the π-calculus is that channels once received cannot be forwarded later on. By means of examples, we give an initial idea of how some privacy notions already studied in the past, such as group creation and name hiding, can be represented without any additional language constructs. We also present an encoding of the (sum-free) π-calculus in our calculus.

1 Introduction

Sharing sensitive information over the internet has become an everyday routine: sending personal data and/or credit card number for online shopping is just one of the examples where the sensitive information can be disposed to other parties. Such cases open the problem of controlling information sharing even among trusted parties. The problem of privacy can (and must) be perceived both from a legal and technological point of view. One of the first who explored privacy in the information age, a legal scholar, Alan Westin had recognized that "Building privacy controls into emerging technologies will require strong effort..." [20]. On the other hand, new technologies can also provide new ways to deal with privacy problems [17]. According to Solove [16], there are four types of privacy violation: invasions, information collection, information processing, and information dissemination (see also [10]). The focus of this paper will be on presenting the techniques for controlling information dissemination in distributed systems.

Although there is a further taxonomy for information dissemination violation by Solove, all these sub-types roughly speak about harms of revealing the personal data or threats of spreading information. In distributed systems where communication of entities is central, controlling the flow of confidential information poses some obstacles. The capability of forwarding, that makes it possible to disseminate received information, may be recognized as one problem in controlling such systems.

Even in the examples of well-structured communications between two parties, such as the ones respecting the protocols specified by session types [9], it can be permissible for any party to forward (i.e., delegate) its session end-point. The session delegation is crucial for establishing sessions between the two parties [4, 18], such as in

$$(\nu session)channel!session.Alice \quad | \quad channel?x.Bob$$

where *Alice* creates a fresh channel *session* and sends one end-point along *channel* to *Bob*. However, it is also possible that the receiving party forwards the channel (cf. session delegation), as we may specify $Bob = forward!x.Bob'$. Such forwarding capability may be appealing to have in some cases (e.g., forwarding tasks from a master to a slave process), but considering *session* is a channel created by *Alice*,

$$\begin{array}{lcl} \pi & ::= & a!k \quad | \quad a?x \quad | \quad [a=b]\pi \\ P & ::= & 0 \quad | \quad \pi.P \quad | \quad P \,|\, P \quad | \quad (\nu k)P \quad | \quad !P \end{array}$$

Table 1: Syntax of prefixes and processes.

pointing to some private data and shared exclusively with *Bob*, one might argue that *Bob* should not gain the capability of session delegation just by receiving *session*. Hence, if we consider the name of channel *session* to be confidential, *Alice* should be the one who decides whether to let a third party knows about the channel. In addition, we may argue that in some cases there is no predefined set of parties that may receive *session* in any future. Indeed, *Alice* should be able to send *session* end-point to any other party she decides. Therefore, we may conclude that confidential information can also be shared in open-ended systems, where the set of users of the information cannot be statically predefined.

In this paper, we present a formal model in which dissemination of information is disabled at the level of the syntax in a direct way. We build on the π-calculus [15], a process model tailored for communication-centric systems, by introducing a subcalculus which we call *Confidential π-calculus*, abbreviated C_π. The only information shared in our calculus are names of channels, so channels are the confidential information. This is the only difference of our model with respect to the π-calculus, names of channels are confidential and hence once received cannot be forwarded later on. By means of examples, the paper gives an initial idea of how some privacy notions already studied in the past, such as group creation and name hiding, can be represented without any additional language constructs. We also define the non-forwarding property and show that, naturally, all C_π processes satisfy this property. This result is then reused to differentiate the π-calculus processes that never forward received channels: if a π process is bisimilar to a C_π process then the π process satisfies the non-forwarding property. We also propose an encoding from π-calculus into C_π-calculus and show its completeness. The paper presents initial results of the investigation of the model, formalization of some of the results are left for an extended version of the paper.

The paper is organized as follows. In Section 2, we start by presenting the syntax and semantics of C_π-calculus, and we state some properties of the labeled transition system. In Section 3 we define a behavioral equivalence relation, called strong bisimilarity. Using the definition of strong bisimilarity we state and prove that the closed domains for channels are directly representable in C_π. Another consequence of this result is the possibility of creating channels with similar behavior to *CCS* channels [12]. Using the strong bisimilarity relation and non-forwarding of C_π processes, we also propose a method for differentiating π-calculus processes that never forward received channels. In addition Section 4 presents some further informal insights on C_π and several interesting scenarios which are naturally represented in our model. Even though non-forwarding property restricts the syntax of the π-calculus, in Section 5 we show the C_π is expressive enough to model the π-calculus. The base idea of the encoding is to create dedicated processes for each channel that handle sending the respective channels. In Section 6 we conclude and point to the related work.

2 Process Model

In this section, we present the syntax and semantics of C_π. The main difference with respect to the π-calculus processes is that in C_π names received in an input cannot be later used as an object of an output, hence disallowing forwarding. Apart from this difference, the remainder of this section should

$$\text{(OUT)}\quad k!l.P \xrightarrow{k!l} P \qquad \text{(IN)}\quad k?x.P \xrightarrow{k?l} P\{l/x\} \qquad \text{(MATCH)}\ \frac{\pi.P \xrightarrow{\alpha} P'}{[a=a]\pi.P \xrightarrow{\alpha} P'} \qquad \text{(RES)}\ \frac{P \xrightarrow{\alpha} P' \quad k \notin \mathsf{n}(\alpha)}{(\nu k)P \xrightarrow{\alpha} (\nu k)P'} \qquad \text{(OPEN)}\ \frac{P \xrightarrow{k!l} Q \quad k \neq l}{(\nu l)P \xrightarrow{(\nu l)k!l} Q}$$

$$\text{(PAR-L)}\ \frac{P \xrightarrow{\alpha} Q \quad \mathsf{bn}(\alpha) \cap \mathsf{fn}(R) = \emptyset}{P \mid R \xrightarrow{\alpha} Q \mid R} \qquad \text{(COMM-L)}\ \frac{P \xrightarrow{k!l} P' \quad Q \xrightarrow{k?l} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \text{(CLOSE-L)}\ \frac{P \xrightarrow{(\nu l)k!l} P' \quad Q \xrightarrow{k?l} Q' \quad l \notin \mathsf{fn}(Q)}{P \mid Q \xrightarrow{\tau} (\nu l)(P' \mid Q')}$$

$$\text{(REP-ACT)}\ \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \text{(REP-COMM)}\ \frac{P \xrightarrow{k!l} P' \quad P \xrightarrow{k?l} P''}{!P \xrightarrow{\tau} (P' \mid P'') \mid !P} \qquad \text{(REP-CLOSE)}\ \frac{P \xrightarrow{(\nu l)k!l} P' \quad P \xrightarrow{k?l} P'' \quad l \notin \mathsf{fn}(P)}{!P \xrightarrow{\tau} (\nu l)(P' \mid P'') \mid !P}$$

Table 2: LTS rules.

come as no surprise to a reader familiar with the π-calculus. To make a clear distinction between names of variables bound in input and names of channels, we introduce two disjoint countable sets $\mathscr{V}$ and $\mathscr{C}$, where $\mathscr{V}$ is the set of variables, ranged over by $x, y, z, \ldots$, and $\mathscr{C}$ is the set of channel names, ranged over by $k, l, m, \ldots$. We denote with $\mathscr{N}$ the union of sets $\mathscr{V}$ and $\mathscr{C}$, and we let $a, b, c, \ldots$ range over $\mathscr{N}$.

Syntax. Table 1 presents the syntax of the language. An inactive process is represented with 0. The prefixed process $\pi.P$ comprehends process $a!k.P$, which on name a sends channel k and then proceeds as P, process $a?x.P$ which on name a receives a channel and substitutes the received channel for x in P, and the last prefix $[a = b]\pi.P$ which exhibits $\pi.P$ only if a and b are the same name. Notice that in our language, differently from the π-calculus, there is a syntactic restriction of the objects in prefixes: only a channel (k) can be sent and only a variable (x) can be used as a placeholder for a channel to be received. For example, π-calculus process $a?x.b!x.0$ is not part of the C_π syntax. There is no restriction on subjects of prefixes and names to be matched, these can be either variables or channels (a). Parallel composition $P \mid P$ stands for two processes simultaneously active, that may interact. Channel restriction $(\nu k)P$ expresses that a new channel k, known only to process P, is created. Replicated process $!P$ introduces an infinite behavior. Intuitively, consider $!P$ stands for an infinite parallel composition of copies of process P (i.e., $P \mid P \mid \cdots$). Here, we do not consider the choice operator (i.e., the sum) since we believe it is not fundamental to our approach, but we remark choice can be added in expected lines.

In $(\nu k)P$ and $a?x.P$, the channel k and the variable x are binding with scope P. The set of bound names $\mathsf{bn}(P)$, for any process P, is defined as the union of bound channels and bound variables in P. The set of free names $\mathsf{fn}(P)$ and the set of names $\mathsf{n}(P)$, for any P, are defined analogously. In addition, we use $\mathsf{fo}(P)$ to denote the set of all free channels appearing as objects of output prefixes in process P.

Semantics. We present an operational semantics for our model in terms of the labeled transition system, which build on observable labeled actions α, defined as

$$\alpha ::= \quad k!l \quad \mid \quad k?l \quad \mid \quad (\nu l)k!l \quad \mid \quad \tau$$

Action $k!l$ sends the channel l on the channel k, while $k?l$ receives the channel l on channel k. In action $(\nu l)k!l$ the sent channel l is bound, and τ stands for internal action. Notice that, as in the π-calculus, names bound in the input (variables) cannot appear in labels of observable actions. To retain the same notation as for processes, we denote by $\mathsf{fn}(\alpha)$, $\mathsf{bn}(\alpha)$ and $\mathsf{n}(\alpha)$, the sets of free, bound and all names of observable α, respectively. As we noted above, these sets contain only channels, and not variables.

The transition relation is defined inductively by the rules given in Table 2. Notice that the action labels and transition rules are defined exactly as in the π-calculus [15]. The symmetric rules for (PAR-L), (COMM-L) and (CLOSE-L) are elided from the table. Rules (OUT), (IN) and (MATCH) are consistent with the explanations of the corresponding syntactic constructs. Rule (RES) ensures that the action of the process is the action of the process scoped over by channel restriction if the channel specified in restriction is not mentioned in the action. Rule (OPEN) opens the scope of the restricted channel, enabling the extrusion of its scope while ensuring that subject and the object of the action are different channels. Rule (PAR-L) lifts the action of one of the branches, and the side condition ensures that the channel bound in the action is not specified as free in the other branch. In rule (COMM-L) two processes performing dual actions, one sending and other receiving l along k, synchronize their actions in the respective parallel composition. In rule (CLOSE-L) the channel sent (l) by the left process is bound and after the synchronization with the right process (performing the dual action), the scope of l is closed while avoiding unintended name capture. Rules (REP-ACT), (REP-COMM) and (REP-CLOSE) describe the actions of a replicated process. The first rule lifts the action of a single copy of the replicated process and activates $!P$ in parallel. The second and the third rules show cases when two copies of replicated process synchronize their actions, either through communicating a free or bound channel, where, again, in both cases a copy of $!P$ is activated in parallel. As usual, we identify α-convertible processes, and thus, we use our transition rules up to α-conversion when needed.

We now present some specific results of the transition relation in the C_π-calculus. Our first result shows the relation between the set of free channels appearing as objects of output prefixes in the process and the process redexes: this set can be (possibly) enlarged only by opening the scope of a bound channel. Even more, the input actions do not affect the set of free channels appearing as objects of output prefixes in the process.

Lemma 1 *Let $P \xrightarrow{\alpha} P'$.*

1. *If $\alpha = k?l$ then $\mathsf{fo}(P') = \mathsf{fo}(P)$.*
2. *If $\alpha = k!l$ then $l \in \mathsf{fo}(P)$ and $\mathsf{fo}(P') \subseteq \mathsf{fo}(P)$.*
3. *If $\alpha = (\nu l)k!l$ then $l \in \mathsf{bn}(P)$ and $\mathsf{fo}(P') \subseteq \mathsf{fo}(P) \cup \{l\}$.*
4. *If $\alpha = \tau$ then $\mathsf{fo}(P') \subseteq \mathsf{fo}(P)$.*

Proof. The proof is by induction on the derivation $P \xrightarrow{\alpha} P'$. We only discuss the base case of 3., when rule (OPEN) is applied. Then, $P = (\nu l)P_1$ and $(\nu l)P_1 \xrightarrow{(\nu l)k!l} P'$ is derived from $P_1 \xrightarrow{k!l} P'$. By 2. of this Lemma, we get $l \in \mathsf{fo}(P_1)$ and $\mathsf{fo}(P') \subseteq \mathsf{fo}(P_1)$. Since $l \in \mathsf{bn}((\nu l)P_1)$, we conclude $\mathsf{fo}(P') \subseteq \mathsf{fo}((\nu l)P_1) \cup \{l\}$.

As a direct consequence of Lemma 1 we get the next corollary.

Corollary 1

1. *If $P \xrightarrow{k?l} P'$ and $l \notin \mathsf{fo}(P)$ then $l \notin \mathsf{fo}(P')$.*
2. *If $l \notin \mathsf{fo}(P)$ then there is no process P' and channel k such that $P \xrightarrow{k!l} P'$.*

What we can conclude from Corollary 1 combining its two statements is that a C_π process cannot send a channel that it previously has received if the channel was not specified as an object of an output prefix in the first place. To show that this property is preserved also by all possible redexes of the process let us first relate the set of free channels appearing in output prefixes of the set and any its execution trace. The result is a direct consequence of Lemma 1.

Corollary 2 *If* $P \xrightarrow{\alpha_1} P_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} P_m$ *then* $\mathsf{fo}(P_m) \subseteq \mathsf{fo}(P) \cup \mathsf{bn}(\alpha_1) \cup \ldots \cup \mathsf{bn}(\alpha_m)$.

The next theorem states that if the channel received by a process was not previously specified as an object of an output prefix, it will not be sent in any of the process possible evolutions. Hence, for the C_π processes forwarding a channel, in a sense that a process can send a channel he learns through receiving, is not possible. Before the theorem, we present the precise definition of non-forwarding.

Definition 1 (Non-forwarding Property) *A process* P_1 *satisfies the non-forwarding property if whenever*

$$P_1 \xrightarrow{\alpha_1} P_2 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} P_{m+1}.$$

then if $l \notin \mathsf{fn}(P_i)$ *and* $\alpha_i = k?l$, *for some* $i = 1, \ldots, m-1$, *then* $\alpha_j \neq k'!l$, *for all* $j = i+1, \ldots, m$.

Notice that in the last definition we could also add the condition $\alpha_i \neq (\nu l)k'!l$, for all $j = i+1, \ldots, m$. But, since without loss of generality we can assume all bound outputs are fresh, we can omit such condition. The next theorem attests that all C_π processes respect the non-forwarding property, but in a more rigorous way, where the only restriction for the channel is not to be specified as the free object of any output prefix.

Theorem 1 (The C_π Processes Respect the Non-Forwarding Property) *Let*

$$P_1 \xrightarrow{\alpha_1} P_2 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} P_{m+1}.$$

Then, if $l \notin \mathsf{fo}(P_i)$ *and* $\alpha_i = k?l$, *for some* $i = 1, \ldots, m-1$, *then* $\alpha_j \neq k'!l$, *for all* $j = i+1, \ldots, m$.

Proof. Since without loss of generality we can assume all bound outputs are fresh and $l \notin \mathsf{fo}(P_i)$, using Corollary 2 we get $l \notin \mathsf{fo}(P_j)$, for $j = i+1, \ldots, m+1$. Hence, by Corollary 1 2. we get $\alpha_j \neq k'!l$, for $j = i+1, \ldots, m$.

The result of Theorem 1 should come as no surprise, C_π processes are designed to respect the non-forwarding property. However, considering the π-calculus, it appears to be nontrivial to differentiate processes that respect the non-forwarding property. To attack this goal, we will see in the next section that Theorem 1 can be reused in Proposition 2.

3 Behavioral equivalence

Based on the notion of observable actions, introduced in Section 2, we investigate some specific behavioral identities of our model. To this end, we introduce a behavioral equivalence, called strong bisimulation, which, colloquially speaking, relates two processes if one can play a symmetric game over them: each action of one process can be mimicked by the other (and with the order reversed), leading to two processes that are again related. The relation that we are interested in is the largest such relation, called strong bisimilarity.

Definition 2 (Strong bisimilarity) *The largest symmetric binary relation over processes* $\sim$, *satisfying*

$$\text{if } P \sim Q \text{ and } P \xrightarrow{\alpha} P', \text{ where } \mathsf{bn}(\alpha) \cap \mathsf{fn}(Q) = \emptyset, \text{ then } Q \xrightarrow{\alpha} Q' \text{ and } P' \sim Q',$$

for some process Q', *is called strong bisimilarity.*

Notice that, since our transition rules match those of the π-calculus, our strong bisimilarity relation is precisely one of the π-calculus [15], restricted to the C_π processes. One consequence of the non-forwarding property of our calculus is the possibility of the creation of closed domains for channels. A property, resembling the creation of a secure channel, which scope is statically determined, can be formally stated using the definition of strong bisimilarity.

Proposition 1 (Closed Domains for Channels) *For any process P, channel m and prefix π, the following equality holds*

$$(\nu k)(((\nu l)k!l.m?y.[y=l]\pi.0) \mid k?x.P) \sim (\nu k)(((\nu l)k!l.m?y.0) \mid k?x.P)$$

Proof. The proof follows by coinduction on the definition of the strong bisimulation (see Appendix A).

In both processes in Proposition 1 the left thread creates a new channel l and sends it over a (private) channel k to the right thread. The equality states that then the channel l cannot be received afterward in the left thread. This is due to the fact that the right thread cannot forward received channels. We may notice that both processes in the proposition define the final scope for channel l, hence, determining a closed domain for the channel. The interpretation of this proposition can be twofold. On one hand, the right thread after receiving a fresh channel (l) cannot send the received channel, since it respects the non-forwarding property. On the other hand, the left thread sends the channel l (to the right thread) only once and the channel afterward behaves "statically", since then it cannot be exchanged even between any two sub-processes of process $P\{l/x\}$. Further explanations are given in the next section.

The π-calculus processes that do not forward names. The C_π processes satisfy our non-forwarding property (Definition 1): if the received channel is new to the process it will not be sent later on. Generally, the π-calculus processes do not meet the non-forwarding property. However, we may notice that there are some π-calculus processes which are not part of the C_π syntax but still respect this property. For example, consider the π-calculus process

$$k?x.(\nu l)(l!x.0 \mid l?y.0)$$

where any received channel along k is then sent on l, but since l is restricted, the process will not output the received channel. But the condition that the channel along which the forwarding is performed (here l) is restricted is not enough. For example, the π-calculus process $k?x.(\nu l)(k!l.l!x.0 \mid l?y.0)$ does not satisfy the non-forwarding property.

This hints that differentiating π-calculus processes that do not forward received channels, in any of their possible evolutions, may not be a simple task. As one solution to the problem we propose the next result which states that a π-calculus process P, and any of its possible evolutions, do not forward received channels if one can find a C_π process Q, such that P and Q are bisimilar. In the theorem, we refer to the non-forwarding property of Definition 1, extended to consider all π-calculus processes. Naturally, the result of the next theorem refers to sum-free π-calculus processes, since in this paper we are not considering the sum operator in the C_π.

Proposition 2 (The π-calculus Processes That do not Forward Names) *Let P be a π-calculus process. If there is a C_π process Q, such that $P \sim Q$, then P satisfies the non-forwarding property.*

Proof. The proof is derived to Appendix A.

Although the result of Proposition 2 is only of the existential nature, we believe it is a step towards more practical results. One such result might be proving that for a given π-calculus process P one can derive a C_π process Q such that if $P \sim Q$ then P respect the non-forwarding property. There we can also use a relaxed definition of the non-forwarding property, in which processes do not forward names received along some predefined set of channels. We leave such investigations for future work.

4 Examples

In this section, we further investigate some interesting scenarios representable in C_π. Since a process in our calculus can learn new names but cannot gain the capability to send such names, we may distinguish two levels of channel ownership of a process that are invariant to the process evolution:

- *administrator*: the process that creates the channel, it has all capabilities over the channel;
- *user*: the process that learns the channel name through communication (scope extrusion) and can communicate along the channel but cannot send it.

Hence, all administrators are also users but the conversely is not true. Also, notice that any process that receives a channel can become a user for that channel (but not administrator), and, hence, all processes may be considered as potential users for any channel. If we consider modelling our opening example in C_π calculus

$$(\nu session)channel!session.Alice \quad | \quad channel?x.Bob \quad | \quad Carol$$

Alice is the administrator for *session* and *Bob* becomes a user after the reception. In C_π it is not possible for *Bob* afterward to send *session* to a third party (e.g., to *Carol*). If *Bob* wants *Carol* to get the access to channel *session* he can only tell *Alice* and let her decide whether she wants to send *session* to *Carol* or not. Therefore, we can have

- $Bob = channel!carol.0$, where *Bob* sends to *Alice* channel *carol*, and then terminates;
- $Alice = channel?y.y!session.Alice'$, where *Alice* receives the channel from *Bob* and decides to send *session* along the received channel, and
- $Carol = carol?x.Carol'$, where *Carol* can finally receive *session* along channel *carol*.

We remark this simple example relies on the purely concurrent setting, here *session* can be held by three or more parties at the same time. The correlation of the C_π and the linearity of session types would need further investigation.

4.1 Authentication

In the C_π-calculus specifying $(\nu l)P$ means P is the administrator for channel l. Process P can extrude the scope of l by sending, but none of the receiving processes will ever become administrators for the received channel l, since they will never gain the capability for sending l. Since sending capability cannot be transferred to other processes, we may conclude the administrator property over some channel can be used as an authentication over processes. For example, a process Q that previously has received l may check at any point if the other party with whom he is communicating at the moment over l is actually an administrator or only a user for channel l. This can be done by specifying

$$Q = (\nu n)l!n.n?x.[x = l].Q',$$

where first the private session with other party listening on l is established by sending a fresh channel n, and then along n a channel is expected to be received. If the channel received is l, then the other party has proved to be an administrator for l. Notice that Q itself does not have to be an administrator for l in this example.

As another example, consider that the above process P has two threads running in parallel

$$k!l.k?y.[y=l]\pi.P_1 \quad | \quad k?x.[x=l]k!l.P_2$$

where, before activating $\pi.P_1$ and P_2 and their possible interactions, both threads test whether the other one is an administrator for channel l. Namely, after the first synchronization, the left thread matches the received channel with l, i.e., we obtain configuration $k?y.[y=l]\pi.P_1 \mid [l=l]k!l.P_2\{l/x\}$. If the channel received is l, then the right thread concludes that the left thread is an administrator for l and sends the same channel back. Then, the left thread also matches the received channel with l and continues only if the two names match, leading to $[l=l]\pi.P_1\{l/y\} \mid P_2\{l/x\}$. After that, both threads have proved to be administrators for l, meaning that both have proved that they originate from process P.

4.2 Modelling groups and name hiding

Controlling name sharing in the π-calculus has been investigated in past and several process models are proposed to this end. In [1], on the π-calculus syntax, an additional construct is introduced, called group creation, and a typing discipline is developed. The intention of the group construct is to restrict communications: channels specified to be in a group cannot be communicated outside the scope of the corresponding group construct. Hence, the group creation closes the domain for channels specified in the construct. In [5], on the π-calculus syntax, an additional construct hide is introduced. Construct hide has similar properties to channel restriction, but it is more static since it forbids the channel extrusion for the channel specified to be hidden. Again, roughly speaking, hide construct closes the domain for a channel specified in the construct. In what follows, we try to represent similar behaviors without any additional language constructs, but directly in the C_π calculus (and hence, directly in the π-calculus). Formalization of the relationship between the mentioned models and the C_π is left for future work.

As one way to represent closed domain for a channel l in the C_π we may consider process

$$(\nu k)((\nu l)k!l.0 \mid k?x.P)$$

resembling the one from Proposition 1, where the left thread creates the channel l, sends it to the right thread and then terminates. The right thread receives the channel, after which the two actions synchronize and the starting process silently evolves to $(\nu k)(\nu l)(0 \mid P\{l/x\})$. As we have shown in Proposition 1, process (and any its subprocess) $P\{l/x\}$ cannot perform output with object l since channel l was not originally created by process P. For this property we can state that name l will never leak out of the scope of P, hence that all communications along channel l are private to process P. Also, we may observe that channel l in process $(\nu k)(\nu l)(0 \mid P\{l/x\})$ has a static behavior, similar to *CCS* channels [12].

This constellation indeed resembles the group creation of the π-calculus with groups and name hiding. The similarity is that in the π-calculus with groups, a channel declared as a member of a group cannot be acquired as a result of communication by the process outside the scope of the group. The major difference is that in our example the channel behaves as a *CCS*-like channel, i.e., the channel cannot be acquired as a result of communication by any process. This brings us to our next example, that combines channel declaration and authentication, presented in Section 4.1. Consider that for a given channel, we want to statically determine a boundary for the possible channel extrusion, the part of the process we shall call a group. In that case, we may conclude that only members of the group should be able to receive the given channel. As a concrete example consider process

$$(\nu g_l)((\nu l)P \mid Q)$$

where by g_l we denote that the scope of channel g_l determines the group for channel l. Now, to make sure that channel l, whose administrator is process P, is sent only to processes scoped over with g_l, before each sending of channel l, we must make sure that the receiver is an administrator for channel g_l. Hence, instead of construct $k!l$ in P, we would use

$$(\nu n)k!n.n?x.[x = g_l]n!l$$

where first a private session with the process willing to receive l is established through channel n, and then the channel received on n is matched with g_l. Only if the name received on n is g_l, i.e., after the other process has proved that he is a member of the group, channel l is sent. Notice that if the other process can send g_l this means the process originates either from P or from Q.

4.3 Open-ended groups

Groups described in the previous example provide an interesting framework to investigate sharing protected resources in distributed environments but has the limitation that a group, once created, always has a fixed scope, which sometimes might be considered too restrictive. In C_π open-ended groups are directly modeled, since sending a resource in C_π does not transmit the capability for its further dissemination. For example, in $(\nu group)k!group.P$ the administrator of *group*, that is the process that creates the group, can send the name of the group to other processes, while the receiving process only becomes a user of the group and does not gain the capability to invite new members to the group.

5 Encoding Uncontrolled Name Passing

In this section, we show how to model forwarding in C_π, as in the standard π-calculus. We start by presenting the basic idea, which we later formalize by means of an encoding.

Throughout this section, we use the polyadic version of our calculus. This enables us to formalize our ideas in a more crisp way, but, as in the π-calculus, each polyadic communication can be represented by a sequence of monadic ones. Furthermore, we will use poliadicity in a controlled way, so that we do not introduce a non-well sorted communications [13].

In the π-calculus, there is no syntactic restriction on names that can appear as objects of output prefixes, hence a process like $k?x.g!x.P \mid k!l.Q$ may be specified. One way to represent this process in C_π is to:

- create a special process dedicated for (repeatedly) sending channel l, called *handler* of the channel,
- while sending channel l also send a channel dedicated for communicating with the handler, and
- bypass sending of the received channel to the handler process.

Hence, we may try to represent the π-calculus process introduced above as

$$k?(x_1,x_2).x_2!g.P \mid k!(l,m_l).Q \mid m_l?y.y!(l,m_l).0$$

where the process in the middle now sends l together with channel m_l dedicated for communicating with the handler process (the rightmost one), and the leftmost process receives both channels and instead of sending l along g it sends g to the handler along m_l. The handler process receives g and sends l (again, together with m_l) along the received channel, in such way mimicking forwarding. Such representation does not work in the case when the leftmost process is not an administrator for channel g (e.g., assume process $k?x.g!x.P$ is derived from $k?y.k?x.y!x.P$), since then the process cannot send g to the handler process. To this end, we must refine our representation of forwarding to support situations when processes are potentially not administrators for any given channel. Thus, we introduce another type of handler processes which are in charge of forwarding channels that are subjects of output actions. For example, the starting π-calculus process would be represented as

$$k?(x_1,x_2).(\nu e)n_g!e.x_2!e.P \mid k!(l,m_l).Q \mid m_l?y.y?z.z!(l,m_l).0 \mid n_g?y.y!g.0$$

where we added the rightmost thread, which is the handler process of channel g, used to bypass sending of channel g in the leftmost thread. Now the communication in this process goes as follows: first, the two leftmost threads synchronize on channel k (as in previous examples), leading to

$$(\nu e)n_g!e.m_l!e.P \mid Q \mid m_l?y.y?z.z!(l,m_l).0 \mid n_g?y.y!g.0$$

where the name x_2 is instantiated with handler name m_l (eliding from the substitution in process P). Then, instead of sending l on g, the new channel e is created and sent to the handlers of l and g

$$(\nu e)(P \mid Q \mid e?z.z!(l,m_l).0 \mid e!g.0)$$

enabling handler of channel g to send g to the handler of channel l, leading to $(\nu e)(P \mid Q \mid g!(l,m_l).0 \mid 0)$ where sending l (together with the handling name m_l) along channel g is finally activated.

Notice that we need a few generalizations of this approach:

- each name can be sent infinitely many times, hence, each handling process must be repeatedly available for communication;
- each channel can be used either as a subject or as an object of output action, and, hence, for each channel we need both types of handler processes (one as for l and the other as for g in the last example). For the rest of this section we will call the handler of a channel a process that comprehends both types of handlers mentioned above for that channel;
- in the source language, processes synchronize their dual (i.e., input/output) actions directly, while in the target language output process first synchronize with the handler process, and only after that handler process synchronize with the input process. In the example above, process P may proceed even though the channel l has not been received by any process. Hence, we need a mechanism to allow processes in the target language to synchronize their actions directly.

We formalize these ideas by introducing an encoding as a pair $([\![\cdot]\!], \varphi_{[\![\,]\!]})$, where $[\![\cdot]\!]$ is a translation function and $\varphi_{[\![\,]\!]}$ is a renaming policy [6]. The translation maps each π-calculus (source) term P into the C_π (target) term $[\![P]\!]$, and while doing so it uses the renaming policy, that maps each name of the source term a into a tuple of names (a,n_a,m_a), where n_a and m_a are not names of any source term, and for each name a different names n_a and m_a are used. Also, the translation uses some additional names, which we assume to be from a reserved set of names, disjoint from all names of the source language and all names introduced by the renaming policy. All these definitions follow the idea of [6].

The translation function is defined in Table 3. The first rule in the table translates a process scoped with the channel restriction. The source process is encoded as scoped with the original channel name k and the two names associated to k by the renaming policy, i.e. n_k and m_k, and it introduces the handler

$$[\![(\nu k)P]\!] = (\nu k, n_k, m_k)([\![P]\!] \mid !n_k?x.x!k.0 \mid !m_k?(x_1,x_2).x_1?y.(\nu t)y!(k,n_k,m_k,t).x_2!t.0)$$

$$[\![a!b.P]\!] = (\nu e_1,e_2)n_a!e_1.m_b!(e_1,e_2).e_2?y.y!e_1.[\![P]\!]$$

$$[\![a?x.P]\!] = a?(x,n_x,m_x,x').x'?y.[\![P]\!]$$

$$[\![[c_1=d_1]\ldots[c_n=d_n]a?x.P]\!] = [c_1=d_1]\ldots[c_n=d_n]a?(x,n_x,m_x,x').x'?y.[\![P]\!]$$

$$[\![[c_1=d_1]\ldots[c_n=d_n]a!b.P]\!] = (\nu e_1,e_2)[c_1=d_1]\ldots[c_n=d_n]n_a!e_1.m_b!(e_1,e_2).e_2?y.y!e_1.[\![P]\!]$$

$$[\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!] \qquad [\![!P]\!] = ![\![P]\!] \qquad [\![0]\!] = 0$$

Table 3: Encoding of π-calculus processes into C_π processes.

process for the channel in parallel with $[\![P]\!]$. We use $(\nu k,n,m)$ to abbreviate $(\nu k)(\nu n)(\nu m)$. The handler process has two threads in parallel. The left thread is repeatedly available to be invoked on n_k (see the rule of output) and it sends channel k along the received channel. The use of this process will be only to send k to the right thread of any other handler process (as n_g in the example above). The right thread of the handler is repeatedly available to be invoked on m_k and it receives a pair of names (see the rule for output). Along the left received name (x_1) it receives a channel (from the left thread of some handler process) and outputs the channel k together with "addresses" of the handler, n_k and m_k, and, in addition, a new channel t. By sending n_k and m_k, we make possible for the process that receives (see the rule for input) to be able afterward to directly invoke the handler for k. By sending a new channel t to the receiving process and also (in the continuation of the right thread of the handler) to the sending process we establish a private connection between the two processes, which then can directly synchronize and activate their continuations at the same time.

The encoding of the output process creates two fresh channels e_1 and e_2, and sends one end of e_1 to the left thread handler process of name a (the subject of the output) and the other end of e_1, together with and e_2, to the right thread of the handler process for name b (the object of the output). In the continuation, before activating the image of P, along e_2 a channel is received and used for the output (to synchronize directly with the input process). We remark that in this rule names e_1, e_2 and y are taken to be from the reserved set of names, and hence cannot appear as free in $[\![P]\!]$. The same assumption is made for names x' and y in the rule for input, hence there also x' and y are not free in $[\![P]\!]$. In the rule for input four channels are received, the channel together with addresses of his handler and a fresh channel (see the right thread of a handler process). The received fresh channel is only used, as we noted, to synchronize with the sending process (same as the channel received in this synchronization). The rest of the rules shows that the encoding is homomorphism elsewhere.

Notice the encoding does not interfere with our notion of ownership described in Section 4. The role of channel administrator is still present, i.e., handlers are included in that domain. Hence, controlling such domain can still be done, in contrast to the regular π-calculus processes where one cannot statically identify a domain where the sending the channel capability is confined to.

We may also notice that in the rule for output (Table 3) the two channels (the addresses) of the two handler processes are used, one for the object and the other for the subject of the prefix. This reflects the fact that in order to be capable to mimic all the actions of the source term, we need to introduce handler processes for all free names of the input and output prefixes of the source term. Since the handler processes are introduced directly in the rule for restricted channels (Table 3), we give our main result for the correctness of the encoding only for the π-calculus processes that contain only bound names. A π-calculus process that has no free names is called *closed*.

As closed π-calculus processes can only exhibit τ transitions, and those match the reduction semantic [15], for the simplicity we chose to deal with the reduction semantics of the π-calculus. Therefore, our operational correspondence result relates the set of closed π-calculus processes, with the reduction semantics (using the reduction relation $\rightarrow$, as defined in [15]), and C_π-processes with the labeled transition system. Notice that the reduction relation of the π-calculus relies on the structural congruence relation [15], which, by rule $[a=a]\pi.P \equiv \pi.P$, may introduce free names. These newly introduced names are not of our interest, as they do not require handlers. To this end, for a π-calculus process P we define $\mathsf{fnn}(P)$, a subset of $\mathsf{fn}(P)$ that is invariant with respect to the structural congruence relation. Hence, we define $\mathsf{fnn}([a=a]\pi.P) = \mathsf{fnn}(\pi.P)$, and otherwise $\mathsf{fnn}(P)$ coincides with $\mathsf{fn}(P)$. Notice that this means $\mathsf{fnn}([a=b]\pi.P) = \{a,b\} \cup \mathsf{fnn}(\pi.P)$, if $a \neq b$.

We use $\xrightarrow{\tau}^*$ to denote the transitive closure of $\xrightarrow{\tau}$. We are now ready to present our result, showing that if the source term P reduces to Q then the encoding of P reduces in a number of τ steps to the process bisimilar to the encoding of the process Q. Again, we assume that the π-calculus processes are sum-free. First, we present an auxiliary result.

Lemma 2 *If $P \rightarrow Q$ then $[\![P]\!] \mid H \xrightarrow{\tau}^* \sim [\![Q]\!] \mid H$, where*

- *if $\mathsf{fnn}(P) = \{k_1, \ldots, k_n\}$ then*
$$H = \prod_{i \in \{1,\ldots,n\}} H_{k_i},$$
- *if $\mathsf{fnn}(P) = \emptyset$ then $H = 0$.*

Proof. The proof is derived to Appendix B.

The above result can already be seen as a form of non-standard completeness as it uses the top-level handlers. This hints that the completeness result can also be stated for π-calculus processes that are not closed, with the encoding that is not compositional, but weakly compositional, such as the encoding from the join-calculus into the π-calculus [3]. We leave such investigations for future work.

Our main result, given in the next theorem, is a direct consequence of Lemma 2.

Theorem 2 (Operational Correspondence: Completeness) *Let P be a closed (sum-free) π-calculus process. If $P \rightarrow Q$ then $[\![P]\!] \xrightarrow{\tau}^* \sim [\![Q]\!]$.*

We also believe that our encoding satisfies the soundness property [6]. This claim relies on the fact that the encoding manipulates the source names in a controlled way. Each name of a source term is translated into a triple of names, and a renaming policy ensures that for different source names different triples of names are used. Other names introduced by the encoding are bound. Using this fact and the fact that each source prefix is translated into a sequence of prefixes of always the same length, we may notice that different post-processing steps might get interleaved, but post-processing steps of different reduction steps in a source term cannot interfere with each other. We also leave formalization of this claim for future work.

6 Conclusions and related work

The notion of secrecy has been studied intensively in process calculi in the past and the variety of techniques have been proposed. The most related to our work are process models building on the π-calculus, such as [1, 5, 2, 10, 7, 19].

Cardelli et al. [1] introduce a language construct for group creation and a typing discipline, where a group is a type for a channel. The group creation construct blocks communications of channels that are declared as members of the group outside the initial scope of the group, hence preventing the leakage of protected channels. Kouzapas and Philippou [10] extend the model of π-calculus with groups by constructs that allow reasoning about the private data in information systems. The work of Giunti et. al. [5] introduces an operator called hide which binds a name and has a similar behavior as a name restriction, but in contrast to name restriction it blocks a name extrusion, for which the scope of the hide operator forms a kind of a group that the "hidden" name cannot exit. The paper by Vivas and Yoshida [19] introduces an operator called filter that is statically associated to a process and blocks all actions of the process along names that are not contained in the (polarized) filter. We also mention [7, 2] where the types associate the security levels to channels, where, in the latter work downgrading the security level of a channel is admissible and it is achieved by introducing special, so-called, declassified input and output prefix constructs. All the above approaches share the property that, when building on the π-calculus model, additional language construct and/or a typing discipline is introduced in order to represent some specific aspect of secrecy in a dedicated way. We believe that C_π-calculus appears to be more suitable as an underlying model when studying secrecy, and as such that many aspects of secrecy can be represented in a more canonical way. As a first step, we plan to make a precise representation of group creation [1] in the C_π-calculus, following the intuition provided in Section 4.2.

Several fragments of the π-calculus have been used in different ways and for different purposes. The asynchronous π-calculus [8], proposed by Honda and Takoro, constrains the syntax by allowing only an inactive process to be the continuation of the output prefix, in this way modelling asynchronous communications. The Localised π-calculus [11], proposed by Merro and Sangiorgi, disallows the input capability for the received names and does not consider the matching operator. There, the syntactic restriction is that input placeholder cannot appear as a subject of an input, but, in contrast to our work, the forwarding of names is allowed. The Private π-calculus [14], proposed by Sangiorgi, makes the restriction that objects of output prefixes are always considered as bound, making the symmetry with the input prefixes. Although in Private π-calculus the forwarding of names is not possible, it differs significantly from our work in the restriction that one name can be sent only once. All these calculi share our goal to investigate specific notions in a dedicated way, without requiring the introduction of specialized primitives, instead by considering a suitable fragment of the π-calculus.

In this paper, we have presented Confidential π-calculus, a fragment of the π-calculus [15] in which the forwarding of received names is disabled at the syntax level. To the best of our knowledge, this is the first process model based on the π-calculus that represents the controlled name passing by constraining and not extending the original syntax. Some specific properties of our labeled transition system are given and the non-forwarding property is defined. All C_π processes satisfy this property and the method to differentiate the π-calculus process that never forward received names is proposed, relying on the strong bisimilarity relation of C_π processes and π processes. The strong bisimilarity relation is also used to show that the creation of closed domains for channels is directly representable in the C_π. Examples presented in the paper already give some intuition on scenarios directly representable in C_π, such as authentication and group modelling, and a complete formalization of these ideas is left for future work. The encoding presented here shows that our model is as expressive as the π-calculus, and the formal verification of the correctness of the encoding is given in the form of the completeness of operational correspondence. The soundness of the encoding is left for future work.

Acknowledgments. The author would like to thank Hugo Torres Vieira and Jovanka Pantović for supervising the work presented here and a great help, to Daniel Hirschkoff for valuable email discussions, and to anonymous reviewers for useful remarks and suggestions. This work has been partially supported by the Ministry of Education and Science of the Republic of Serbia, project ON174026.

References

[1] Luca Cardelli, Giorgio Ghelli & Andrew D. Gordon (2005): *Secrecy and group creation*. *Inf. Comput.* 196(2), pp. 127–155, doi:10.1016/j.ic.2004.08.003.

[2] Silvia Crafa & Sabina Rossi (2007): *Controlling information release in the pi-calculus*. *Inf. Comput.* 205(8), pp. 1235–1273, doi:10.1016/j.ic.2007.01.001.

[3] Cédric Fournet & Georges Gonthier (1996): *The Reflexive CHAM and the Join-Calculus*. In Hans-Juergen Boehm & Guy L. Steele Jr., editors: *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, ACM Press, pp. 372–385, doi:10.1145/237721.237805.

[4] Simon J. Gay & Malcolm Hole (2005): *Subtyping for session types in the pi calculus*. *Acta Inf.* 42(2-3), pp. 191–225, doi:10.1007/s00236-005-0177-z.

[5] Marco Giunti, Catuscia Palamidessi & Frank D. Valencia (2012): *Hide and New in the Pi-Calculus*. In Bas Luttik & Michel A. Reniers, editors: *Proceedings Combined 19th International Workshop on Expressiveness in Concurrency and 9th Workshop on Structured Operational Semantics, EXPRESS/SOS 2012, Newcastle upon Tyne, UK, September 3, 2012.*, *EPTCS* 89, pp. 65–79, doi:10.4204/EPTCS.89.6.

[6] Daniele Gorla (2010): *Towards a unified approach to encodability and separation results for process calculi*. *Inf. Comput.* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[7] Matthew Hennessy (2005): *The security pi-calculus and non-interference*. *J. Log. Algebr. Program.* 63(1), pp. 3–34, doi:10.1016/j.jlap.2004.01.003.

[8] Kohei Honda & Mario Tokoro (1991): *An Object Calculus for Asynchronous Communication*. In Pierre America, editor: *ECOOP'91 European Conference on Object-Oriented Programming, Geneva, Switzerland, July 15-19, 1991, Proceedings*, *Lecture Notes in Computer Science* 512, Springer, pp. 133–147, doi:10.1007/BFb0057019.

[9] Kohei Honda, Vasco Thudichum Vasconcelos & Makoto Kubo (1998): *Language Primitives and Type Discipline for Structured Communication-Based Programming*. In Chris Hankin, editor: *Programming Languages and Systems - ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 - April 4, 1998, Proceedings*, *Lecture Notes in Computer Science* 1381, Springer, pp. 122–138, doi:10.1007/BFb0053567.

[10] Dimitrios Kouzapas & Anna Philippou (2017): *Privacy by typing in the π-calculus*. *Logical Methods in Computer Science* 13(4), pp. 1–42, doi:10.23638/LMCS-13(4:27)2017.

[11] Massimo Merro & Davide Sangiorgi (2004): *On asynchrony in name-passing calculi*. *Mathematical Structures in Computer Science* 14(5), pp. 715–767, doi:10.1017/S0960129504004323.

[12] Robin Milner (1980): *A Calculus of Communicating Systems*. *Lecture Notes in Computer Science* 92, Springer, doi:10.1007/3-540-10235-3.

[13] Robin Milner (1992): *The Polyadic Pi-calculus (Abstract)*. In Rance Cleaveland, editor: *CONCUR '92, Third International Conference on Concurrency Theory, Stony Brook, NY, USA, August 24-27, 1992, Proceedings*, *Lecture Notes in Computer Science* 630, Springer, p. 1, doi:10.1007/BFb0084778.

[14] Davide Sangiorgi (1996): *pi-Calculus, Internal Mobility, and Agent-Passing Calculi*. *Theor. Comput. Sci.* 167(1&2), pp. 235–274, doi:10.1016/0304-3975(96)00075-8.

[15] Davide Sangiorgi & David Walker (2001): *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press.

[16] Daniel J Solove (2005): *A taxonomy of privacy*. *U. Pa. L. Rev.* 154, p. 477, doi:10.2307/40041279.

[17] Michael Carl Tschantz & Jeannette M. Wing (2009): *Formal Methods for Privacy*. In Ana Cavalcanti & Dennis Dams, editors: *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, *Lecture Notes in Computer Science* 5850, Springer, pp. 1–15, doi:`10.1007/978-3-642-05089-3_1`.

[18] Vasco T. Vasconcelos (2012): *Fundamentals of session types*. *Inf. Comput.* 217, pp. 52–70, doi:`10.1016/j.ic.2012.05.002`.

[19] José-Luis Vivas & Nobuko Yoshida (2002): *Dynamic Channel Screening in the Higher Order pi-Calculus*. *Electr. Notes Theor. Comput. Sci.* 66(3), pp. 170–184, doi:`10.1016/S1571-0661(04)80421-3`.

[20] Alan F Westin (2003): *Social and political dimensions of privacy*. *Journal of social issues* 59(2), pp. 431–453, doi:`10.1111/1540-4560.00072`.

A Proofs from Section 3

Proposition 1 For any process P, channel m and prefix π, next equality holds

$$(\nu k)(((\nu l)k!l.m?y.[y=l]\pi.0) \mid k?x.P) \sim (\nu k)(((\nu l)k!l.m?y.0) \mid k?x.P).$$

Proof. The proof follows by coinduction, by showing that the relation

$$\begin{aligned}\mathscr{R} \;=\; \{ & \big((\nu k)((\nu l)k!l.m?y.[y=l]\pi.0 \mid k?x.P), (\nu k)((\nu l)k!l.m?y.0 \mid k?x.P)\big),\\ & \big((\nu k)(\nu l)(m?y.[y=l]\pi.0 \mid Q), (\nu k)(\nu l)(m?y.0 \mid Q)\big),\\ & \big((\nu l)(m?y.[y=l]\pi.0 \mid Q), (\nu l)(m?y.0 \mid Q)\big),\\ & \big((\nu k)(\nu l)([n=l]\pi.0 \mid Q), (\nu k)(\nu l)(0 \mid Q)\big),\\ & \big((\nu l)([n=l]\pi.0 \mid Q), (\nu l)(0 \mid Q)\big)\\ & \big((\nu k)(\nu l)(\nu n)([n=l]\pi.0 \mid Q), (\nu k)(\nu l)(\nu n)(0 \mid Q)\big),\\ & \big((\nu l)(\nu n)([n=l]\pi.0 \mid Q), (\nu l)(\nu n)(0 \mid Q)\big)\\ & \mid \text{for all } n,m \in \mathscr{C}, \text{ such that } n \neq l, \text{ and all processes } P \text{ and } Q, \text{ such that } l \notin \mathsf{fo}(Q)\,\}\end{aligned}$$

where $n \neq l$, is contained in the strong bisimilarity, i.e., $\mathscr{R} \subseteq \sim$.

We show that each action of one process can be mimicked by the other process in the pair in $\mathscr{R}$, leading to processes that are again in relation $\mathscr{R}$. Let the process in the first pair

$$(\nu k)((\nu l)k!l.m?y.[y=l]\pi.0 \mid k?x.P) \xrightarrow{\alpha} P'.$$

Then, since actions of the starting process can only be actions of its two branches, we conclude that either $\alpha = (\nu l)k!l$ or $\alpha = k?n$ or it is the synchronization of these two actions, in which case $\alpha = \tau$. We reject the first two options, since the subject of the action is bound in the starting process and by rule (RES) it cannot be observed outside of the process. Hence, we conclude $\alpha = \tau$ and $P' = (\nu k)(\nu l)(m?y.[y = l]\pi.0 \mid P\{l/x\})$. Then, by applying (OUT), (OPEN), (IN), (CLOSE-L) and (RES), respectively, we get

$$(\nu k)((\nu l)k!l.m?y.0 \mid k?x.P) \xrightarrow{\tau} (\nu k)(\nu l)(m?y.0 \mid P\{l/x\}),$$

and since $l \notin \mathsf{fn}(P)$ and x cannot appear as an object in the prefixes in P we conclude $l \notin \mathsf{fo}(P\{l/x\})$. Hence, we have $\big((\nu k)(\nu l)(m?y.[y = l]\pi.0 \mid P\{l/x\}), (\nu k)(\nu l)(m?y.0 \mid P\{l/x\})\big) \in \mathscr{R}$. The symmetric case is analogous.

Now let us consider processes in the second pair of $\mathscr{R}$. If

$$(\nu k)(\nu l)(m?y.[y=l]\pi.0 \mid Q) \xrightarrow{\alpha} P',$$

then observable α can originate from both of the branches or from their synchronization.

—*Left branch:* If the observable originate from the left branch, then $\alpha = m?n$, and by (IN), (PAR-L) and (RES) we get

$$(\nu k)(\nu l)(m?y.[y=l]\pi.0 \mid Q) \xrightarrow{m?n} (\nu k)(\nu l)([n=l]\pi.0 \mid Q),$$

where, by the side condition of (RES) we conclude $n \notin \{k,l\}$. In the same way we get

$$(\nu k)(\nu l)(m?y.0 \mid Q) \xrightarrow{m?n} (\nu k)(\nu l)(0 \mid Q),$$

and $\big((\nu k)(\nu l)([n=l]\pi.0 \mid Q), (\nu k)(\nu l)(0 \mid Q)\big) \in \mathscr{R}$ holds.

—*Right branch:* If the action originates from the right branch, i.e., from $Q \xrightarrow{\alpha} Q'$, we distinguish two cases:

(i) if by rules (PAR-R) and (RES) is derived

$$(\nu k)(\nu l)(m?y.[y=l]\pi.0 \mid Q) \xrightarrow{\alpha} (\nu k)(\nu l)(m?y.[y=l]\pi.0 \mid Q'),$$

where we conclude that $k,l \notin \mathsf{n}(\alpha)$, hence $l \notin \mathsf{fo}(Q')$. Then by the same rules we get

$$(\nu k)(\nu l)(m?y.0 \mid Q) \xrightarrow{\alpha} (\nu k)(\nu l)(m?y.0 \mid Q'),$$

and $\big((\nu k)(\nu l)(m?y.[n=l]\pi.0 \mid Q'), (\nu k)(\nu l)(m?y.0 \mid Q')\big) \in \mathscr{R}$ holds.

(ii) if by rules (PAR-R), (RES) and (OPEN) is derived

$$(\nu k)(\nu l)(m?y.[y=l]\pi.0 \mid Q) \xrightarrow{(\nu k)\alpha} (\nu l)(m?y.[y=l]\pi.0 \mid Q'),$$

then $l \notin \mathsf{n}(\alpha)$. Notice that the scope of channel l cannot be extruded this way since $l \notin \mathsf{fo}(Q)$. Hence, process Q cannot perform output action with object l. Then by the same rules we get

$$(\nu k)(\nu l)(m?y.0 \mid Q) \xrightarrow{(\nu k)\alpha} (\nu l)(m?y.0 \mid Q'),$$

and, again, $\big((\nu l)(m?y.[n=l]\pi.0 \mid Q'), (\nu l)(m?y.0 \mid Q')\big) \in \mathscr{R}$ holds.

—*Synchronization of branches:* We again distinguish two cases:

(i) if from

$$m?y.[y=l]\pi.0 \xrightarrow{m?n} [n=l]\pi.0 \qquad \text{and} \qquad Q \xrightarrow{m!n} Q',$$

where we can make the same observation on Q as before to conclude that $l \neq n$, by rules (COMM-R) and (RES) is derived

$$(\nu k)(\nu l)(m?y.[y=l]\pi.0 \mid Q) \xrightarrow{\tau} (\nu k)(\nu l)([n=l]\pi.0 \mid Q').$$

Then, using $m?y.0 \xrightarrow{m?n} 0$, and the same rules as above we get

$$(\nu k)(\nu l)(m?y.0 \mid Q) \xrightarrow{\tau} (\nu k)(\nu l)(0 \mid Q'),$$

and we get $\big((\nu k)(\nu l)([n=l]\pi.0 \mid Q'), (\nu k)(\nu l)(0 \mid Q')\big) \in \mathscr{R}$.

(ii) if from

$$m?y.[y = l]\pi.0 \xrightarrow{m?n} [n = l]\pi.0 \qquad \text{and} \qquad Q \xrightarrow{(\nu n)m!n} Q',$$

where as before we can assume $l \neq n$, by rules (CLOSE-R) and (RES) is derived

$$(\nu k)(\nu l)(m?y.[y = l]\pi.0 \mid Q) \xrightarrow{\tau} (\nu k)(\nu l)(\nu n)([n = l]\pi.0 \mid Q'),$$

then using $m?y.0 \xrightarrow{m?n} 0$, we may observe

$$(\nu k)(\nu l)(m?y.0 \mid P) \xrightarrow{\tau} (\nu k)(\nu l)(\nu n)(0 \mid Q'),$$

and $\big((\nu k)(\nu l)(\nu n)([n = l]\pi.0 \mid Q'), (\nu k)(\nu l)(\nu n)(0 \mid Q')\big) \in \mathscr{R}$.

The symmetric cases and the rest of the pairs from $\mathscr{R}$ are analogous. For the rest of the pairs note that in all left branch of the first components we have $[n = l]\pi.0$, where $n \neq l$, and hence, its observational power is equivalent to the observational power of inactive process 0 appearing as the left branch in the right components.

Proposition 2 Let P be a (sum-free) π-calculus process. If there is a C_π process Q, such that $P \sim Q$, then P satisfies the non-forwarding property.

Proof. Let $P_1 = P$ be a be a (sum-free) π-calculus process and let $P_1 \xrightarrow{\alpha_1} P_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} P_{m+1}$. Let us fix $i \in \{1, \dots, m-1\}$, and assume $l \notin \mathsf{fn}(P_i)$ and $\alpha_i = k?l$. Since without loss of generality we can assume all bound outputs are fresh, we get $\alpha_j \neq (\nu l)k'!l$, for all $j = i+1, \dots, m$, directly. In addition to the first assumption, let us assume there is $j \in \{i+1, \dots, m\}$ such that $\alpha_j = k'!l$. Since $P_1 \sim Q_1$ (where $Q_1 = Q$), we conclude there are C_π processes $Q_2, \dots, Q_{m+1}$ such that

$$Q_1 \xrightarrow{\alpha_1} Q_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_m} Q_{m+1}$$

and $P_n \sim Q_n$, for all $n = 1, \dots, m+1$, where $Q_i \xrightarrow{k?l} Q_{i+1}$ and $Q_j \xrightarrow{k'!l} Q_{j+1}$. We now distinguish two cases.

1. If $l \notin \mathsf{fn}(Q_i)$ then we get a direct contradiction with Theorem 1.
2. If $l \in \mathsf{fn}(Q_i)$, we choose a fresh channel l' and a substitution σ that is defined only on channel l and maps it to l'. Then, from $P_j \xrightarrow{\alpha_j} P_{j+1}$, by consequitive application of Lemma 1.4.8 of [15], we conclude $(P_j)\sigma \xrightarrow{(\alpha_j)\sigma} (P_{j+1})\sigma$, for all $j = i+1, \dots, m$. Since $l \notin \mathsf{fn}(P_i)$ we get $(P_i)\sigma = P_i$. Now from

$$P_1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{i-1}} P_i \xrightarrow{(\alpha_i)\sigma} (P_{i+1})\sigma \xrightarrow{(\alpha_{i+1})\sigma} \dots \xrightarrow{(\alpha_m)\sigma} (P_{m+1})\sigma,$$

and $P_1 \sim Q_1$, we again conclude there are C_π processes $Q_2, \dots, Q_{m+1}$ such that

$$Q_1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{i-1}} Q_i \xrightarrow{(\alpha_i)\sigma} Q_{i+1} \xrightarrow{(\alpha_{i+1})\sigma} \dots \xrightarrow{(\alpha_m)\sigma} Q_{m+1},$$

where $P_j \sim Q_j$, for all $j = 1, \dots, i$ and $(P_j)\sigma \sim Q_j$, for all $j = i+1, \dots, m+1$. Since l' has been chosen to be a fresh channel, we get $l' \notin \mathsf{fn}(Q_i)$, and since $Q_i \xrightarrow{(k)\sigma?l'} Q_{i+1}$ and $Q_j \xrightarrow{(k')\sigma!l'} Q_{j+1}$, we fall into the first case, and hence, we again get contradiction with Theorem 1.

B Proofs from Section 5

Abbreviations. For the sake of readability, use the abbreviation

$$H_k = !n_k?x.x!k.0 \mid !m_k?(x_1,x_2).x_1?y.(\nu t)y!(k,n_k,m_k,t).x_2!t.0,$$

assuming that $\varphi_{[\![\,]\!]}(k) = (k,n_k,m_k)$, and we omit writing trailing 0's whenever possible.

We may notice that the encoding defined in Table 3 does not introduce any free names by itself, except in the rule for output (and matching prefixed output), where names n_a and m_b are introduced. We may also notice also that these introduced names are the ones specified in the renaming policy of names a and b, respectively. Hence, the following result is straightforward.

Lemma 3 (Name invariance) *Let P be a π-calculus process and let substitutions σ and σ' be such that $\varphi_{[\![\,]\!]}((a)\sigma) = (\varphi_{[\![\,]\!]}(a))\sigma'$, for all $a \in \mathscr{N}$. Then $[\![(P)\sigma]\!] = ([\![P]\!])\sigma'$.*

For the operational correspondence, we will need only one case of the name invariance result, and we state it in the next corollary.

Corollary 3 *If $\varphi_{[\![\,]\!]}(k) = (k,n_k,m_k)$ and $\varphi_{[\![\,]\!]}(x) = (x,n_x,m_x)$ then*

$$[\![P]\!]\{k/x\}\{n_k/n_x\}\{m_k/m_x\} = [\![P\{k/x\}]\!].$$

For the operational correspondence we will need definition of strong bisimilarity of polyadic C_π-calculus. Such definition is exactly the same as Definition 2, except that labels of the LTS (where rules in Table 2 are adapted for polyadic calculus following expected lines) are carrying a tuple of channels, i.e.,

$$\alpha ::= \quad k!(l_1,\dots,l_n) \quad \mid \quad k?(l_1,\dots,l_n) \quad \mid \quad (\nu l_1,\dots,l_m)k!(l_1,\dots,l_n) \quad \mid \quad \tau$$

For such strong bisimilarity relation $\sim$ we may show to obey some standard properties stated in the next proposition.

Proposition 3 *1. $\sim$ is an equivalence relation and a non-input congruence;*

2. $[a=a]\pi.P \sim \pi.P$;

3. $P_1 \mid (P_2 \mid P_3) \sim (P_1 \mid P_2) \mid P_3$;

4. $P_1 \mid P_2 \sim P_2 \mid P_1$;

5. $P \mid 0 \sim P$;

6. $(\nu k)(\nu l)P \sim (\nu l)(\nu k)P$;

7. $(\nu k)0 \sim 0$;

8. $P_1 \mid (\nu k)P_2 \sim (\nu k)(P_1 \mid P_2)$, if $k \notin \mathsf{fn}(P_1)$;

9. $!P \sim P \mid !P$;

10. $(\nu k)!k?(x_1,\dots,x_n).P \sim 0$;

11. $(\nu k,n_k,m_k)H_k \sim 0$.

We take the structural congruence relation $\equiv$ as it is defined for the π-calculus processes in [15].

Lemma 4 *If $P \equiv Q$ then $[\![P]\!] \sim [\![Q]\!]$.*

Proof. The proof is by case analysis on the structural congruence rule applied.

1. $[a=a]\pi.P \equiv \pi.P$.

   We distinguish two cases for prefix π.

   (a) If $\pi = [b_1 = c_1]\dots[b_n = c_n]d?x$, then by definition of the encoding and Proposition 3 we get

   $$\begin{aligned}\llbracket [a=a]\pi.P \rrbracket &= [a=a][b_1=c_1]\dots[b_n=c_n]d?(x,n_x,m_x,x').x'?y.\llbracket P \rrbracket\\ &\sim [b_1=c_1]\dots[b_n=c_n]d?(x,n_x,m_x,x').x'?y.\llbracket P \rrbracket\\ &= \llbracket [b_1=c_1]\dots[b_n=c_n]d?x.P \rrbracket\\ &= \llbracket \pi.P \rrbracket.\end{aligned}$$

   (b) If $\pi = [b_1 = c_1]\dots[b_n = c_n]d!g$, then, again, by definition of the encoding and Proposition 3 we get

   $$\begin{aligned}\llbracket [a=a]\pi.P \rrbracket &= (\nu e_1,e_2)[a=a][b_1=c_1]\dots[b_n=c_n]n_d!e_1.m_g!(e_1,e_2).e_2?x.x!e_1.\llbracket P \rrbracket\\ &\sim (\nu e_1,e_2)[b_1=c_1]\dots[b_n=c_n]n_d!e_1.m_g!(e_1,e_2).e_2?x.x!e_1.\llbracket P \rrbracket\\ &= \llbracket [b_1=c_1]\dots[b_n=c_n]d!e.P \rrbracket\\ &= \llbracket \pi.P \rrbracket.\end{aligned}$$

2. $(\nu k)(\nu l)P \equiv (\nu l)(\nu k)P$. By the definition of the encoding and Proposition 3 we get

   $$\begin{aligned}\llbracket (\nu k)(\nu l)P \rrbracket &= (\nu k,n_k,m_k)((\nu l,n_l,m_l)(\llbracket P \rrbracket \mid H_l) \mid H_k)\\ &\sim (\nu l,n_l,m_l)((\nu k,n_k,m_k)(\llbracket P \rrbracket \mid H_k) \mid H_l))\\ &= \llbracket (\nu l)(\nu k)P \rrbracket.\end{aligned}$$

3. $(\nu k)0 \equiv 0$. By the definition of the encoding and Proposition 3 we get

   $$\begin{aligned}\llbracket (\nu k)0 \rrbracket &= (\nu k,n_k,m_k)(0 \mid H_k)\\ &\sim (\nu k,n_k,m_k)H_k\\ &\sim 0 = \llbracket 0 \rrbracket.\end{aligned}$$

4. $P \mid (\nu a)Q \equiv (\nu a)(P \mid Q)$, if $a \notin \mathsf{fn}(P)$. By the definition of the encoding and Proposition 3 we get

   $$\begin{aligned}\llbracket P \mid (\nu k)Q \rrbracket &= \llbracket P \rrbracket \mid (\nu k,n_k,m_k)(\llbracket Q \rrbracket \mid H_k)\\ &\sim (\nu k,n_k,m_k)(\llbracket P \rrbracket \mid \llbracket Q \rrbracket \mid H_k)\\ &= (\nu k,n_k,m_k)(\llbracket P \mid Q \rrbracket \mid H_k)\\ &= \llbracket (\nu l)(P \mid Q) \rrbracket.\end{aligned}$$

5. The rest of the cases are analogous.

Lemma 5 *Let P and Q be π-calculus processes.*

1. *If $P \equiv Q$ then $\mathsf{fnn}(P) = \mathsf{fnn}(Q)$.*
2. *If $a \notin \mathsf{fnn}(P)$ then there exist a π-calculus process P' such that $P \equiv P'$ and $a \notin \mathsf{fn}(P')$.*
3. *if $a \notin \mathsf{fnn}(P)$ and $P \to Q$ then $a \notin \mathsf{fnn}(Q)$.*

Proof. 1. The only structural congruence rule affecting free names is $[a=a]\pi.P \equiv \pi.P$, and by the definition $\mathsf{fnn}([a=a]\pi.P) = \mathsf{fnn}(\pi.P)$.

2. Assume $a \notin \mathsf{fnn}(P)$. If $a \notin \mathsf{fn}(P)$ then the proof is finished. Now assume $a \in \mathsf{fn}(P)$. Then a can appear only in P in a sub-process of the form $[a = a]\pi.Q$. In this case we may show, by induction on the structure of P, that using the structural congruence rule $[a = a]\pi.Q \equiv \pi.Q$, we can get rid of all such matchings that mention name a.

3. Follows by an easy induction on $\rightarrow$ derivation.

Lemma 2 If $P \rightarrow Q$ then $[\![P]\!] \mid H \xrightarrow{\tau}^* \sim [\![Q]\!] \mid H$, where

- if $\mathsf{fnn}(P) = \{k_1, \ldots, k_n\}$, then
$$H = \prod_{i \in \{1,\ldots,n\}} H_{k_i},$$
- if $\mathsf{fnn}(P) = \emptyset$ then $H = 0$.

Proof. The proof is by induction on $\rightarrow$ derivation.

1. *Base case*: $k!l.P \mid k?x.Q \rightarrow P \mid Q\{l/x\}$. Since k and l are free in the starting process, we can encode it as
$$R = [\![k!l.P \mid k?x.Q]\!] \mid H_k \mid H_l \mid H,$$
where if $\mathsf{fnn}(k!l.P \mid k?x.Q) = \{k, l, k_1, \ldots, k_n\}$ then $H = \prod_{i \in \{1,\ldots,n\}} H_{k_i}$. If $\mathsf{fnn}(k!l.P \mid k?x.Q) = \{k, l\}$ then $H = 0$. Then,
$$\begin{array}{rl} R & = [\![k!l.P]\!] \mid [\![k?x.Q]\!] \mid H_k \mid H_l \mid H \\ & = (\nu e_1, e_2) n_k!e_1.m_l!(e_1,e_2).e_2?y.y!e_1.[\![P]\!] \mid k?(x, n_x, m_x, x').x'?y.[\![Q]\!] \\ & \quad \mid !n_k?y.y!k \mid !m_k?(x_1,x_2).x_1?y.(\nu t)y!(k, n_k, m_k, t).x_2!t \\ & \quad \mid !n_l?y.y!l \mid !m_l?(x_1,x_2).x_1?y.(\nu t)y!(l, n_l, m_l, t).x_2!t \mid H \\ & \xrightarrow{\tau}\xrightarrow{\tau} (\nu e_1, e_2)(e_2?y.y!e_1.[\![P]\!] \mid k?(x, n_x, m_x, x').x'?y.[\![Q]\!] \\ & \quad \mid e_1!k \mid H_k \\ & \quad \mid !n_l?y.y!l \mid e_1?y.(\nu t')y!(l, n_l, m_l, t').e_2!t') \mid !m_l?(x_1,x_2).x_1?y.(\nu t)y!(l, n_l, m_l, t).x_2!t \mid H, \end{array}$$
where the output process synchronize with the left thread of the handler of name k and with the right thread of the handler of name l. At this point, the two handlers can synchronize and the last process evolves to
$$\begin{array}{l} \xrightarrow{\tau}\xrightarrow{\tau}\xrightarrow{\tau} (\nu e_2, e_1, t')(t'!e_1.[\![P]\!] \mid t'?y.[\![Q]\!]\{l/x\}\{n_l/n_x\}\{m_l/m_x\} \\ \quad \mid 0 \mid H_k \\ \quad \mid !n_l?y.y!l \mid 0) \mid !m_l?(x_1,x_2).x_1?y.(\nu t)y!(l, n_l, m_l, t).x_2!t \mid H, \end{array}$$
where, after the synchronization of the two handlers, name l (together with n_l, m_l and t') is finally received in the input process, after which channel t' is also received in the left-hand side process, making the encoding of processes P an Q only unlocked in the synchronization:
$$\begin{array}{l} \xrightarrow{\tau} (\nu e_2, e_1, t')([\![P]\!] \mid [\![Q]\!]\{l/x\}\{n_l/n_x\}\{m_l/m_x\} \\ \quad \mid 0 \mid H_k \\ \quad \mid !n_l?y.y!l \mid 0) \mid !m_l?(x_1,x_2).x_1?y.(\nu t)y!(l, n_l, m_l, t).x_2!t \mid H. \end{array} \tag{1}$$
By Corollary 3 we get
$$[\![Q]\!]\{l/x\}\{n_l/n_x\}\{m_l/m_x\} = [\![Q\{l/x\}]\!].$$

Hence, we conclude the last derived process in equation (1) is equal to

$$\begin{aligned}&(\nu e_2, e_1, t')(\llbracket P\rrbracket \mid \llbracket Q\{l/x\}\rrbracket\\&\mid 0 \mid H_k\\&\mid !n_l?y.y!l \mid 0) \mid !m_l?(x_1,x_2).x_1?y.(\nu t)y!(l,n_l,m_l,t).x_2!t \mid H.\end{aligned}$$

Since $e_2, e_1, t' \notin \mathsf{fn}(\llbracket P\rrbracket \mid \llbracket Q\{l/x\}\rrbracket \mid 0 \mid H_k \mid !n_l?y.y!l \mid 0)$, by Proposition 3 we have that the last derived process is strongly bisimilar to

$$\begin{aligned}&\llbracket P\rrbracket \mid \llbracket Q\{l/x\}\rrbracket \mid (\nu e_2)(\nu e_1)(\nu t)0 \mid H_k \mid H_l \mid H\\ \sim\,&\llbracket P\rrbracket \mid \llbracket Q\{l/x\}\rrbracket \mid H_k \mid H_l \mid H\\ =\,&\llbracket P \mid Q\{l/x\}\rrbracket \mid H_k \mid H_l \mid H.\end{aligned}$$

2. $P \mid R \to Q \mid R$ is derived from $P \to Q$. By induction hypothesis

$$\llbracket P\rrbracket \mid H_1 \xrightarrow{\tau}{}^{*} S,$$

where $S \sim \llbracket Q\rrbracket \mid H_1$ and if $\mathsf{fnn}(P) = \{k_1,\dots,k_n\}$ then

$$H_1 = \prod_{i\in\{1,\dots,n\}} H_{k_i},$$

and if $\mathsf{fnn}(P) = \emptyset$ then $H_1 = 0$. Now, if $\mathsf{fnn}(R) \setminus \mathsf{fnn}(P) = \{l_1,\dots,l_m\}$, let us take

$$H_2 = \prod_{j\in\{1,\dots,m\}} H_{l_j}.$$

If $\mathsf{fnn}(R) \setminus \mathsf{fnn}(P) = \emptyset$ let us take $H_2 = 0$. Then, by (PAR-L) we can derive

$$\llbracket P\rrbracket \mid H_1 \mid \llbracket R\rrbracket \mid H_2 \xrightarrow{\tau}{}^{*} S \mid \llbracket R\rrbracket \mid H_2.$$

By Lemma 3 we get $\llbracket P\rrbracket \mid H_1 \mid \llbracket R\rrbracket \mid H_2 \sim \llbracket P\rrbracket \mid \llbracket R\rrbracket \mid H_1 \mid H_2$ then

$$\llbracket P \mid R\rrbracket \mid H_1 \mid H_2 = \llbracket P\rrbracket \mid \llbracket R\rrbracket \mid H_1 \mid H_2 \xrightarrow{\tau}{}^{*} S',$$

where $S' \sim S \mid \llbracket R\rrbracket \mid H_2$, by the definition of strong bisimilarity. We can now conclude

$$\begin{aligned}S' &\sim S \mid \llbracket R\rrbracket \mid H_2\\ &\sim \llbracket Q\rrbracket \mid H_1 \mid \llbracket R\rrbracket \mid H_2\\ &\sim \llbracket Q\rrbracket \mid \llbracket R\rrbracket \mid H_1 \mid H_2\\ &= \llbracket Q \mid R\rrbracket \mid H_1 \mid H_2.\end{aligned}$$

3. $(\nu k)P \to (\nu k)Q$ is derived from $P \to Q$. Again, by induction hypothesis

$$\llbracket P\rrbracket \mid H_1 \xrightarrow{\tau}{}^{*} S,$$

where $S \sim \llbracket Q\rrbracket \mid H_1$ and if $\mathsf{fnn}(P) = \{k_1,\dots,k_n\}$ then

$$H_1 = \prod_{i\in\{1,\dots,n\}} H_{k_i},$$

while if $\mathsf{fnn}(P) = \emptyset$ then $H_1 = 0$. Since,

$$\llbracket(\nu k)P\rrbracket \mid H = (\nu k, n_k, m_k)(\llbracket P\rrbracket \mid H_k) \mid H,$$

we distinguish two cases:

(a) if $k \in \mathsf{fnn}(P)$ then $H_k \mid H = H_1$. Since $k, n_k, m_k \notin \mathsf{fn}(H)$, by Proposition 3 we get

$$\begin{array}{rcl} (\nu k, n_k, m_k)([\![P]\!] \mid H_k) \mid H & \sim & (\nu k, n_k, m_k)([\![P]\!] \mid H_k \mid H) \\ & \xrightarrow{\tau}^* & (\nu k, n_k, m_k)S, \end{array}$$

where $\xrightarrow{\tau}^*$ transition(s) follows by the induction hypothesis and rule (RES). By Proposition 3

$$\begin{array}{rcl} (\nu k, n_k, m_k)S & \sim & (\nu k, n_k, m_k)([\![Q]\!] \mid H_1) \\ & = & (\nu k, n_k, m_k)([\![Q]\!] \mid H_k \mid H) \\ & \sim & (\nu k, n_k, m_k)([\![Q]\!] \mid H_k) \mid H \\ & = & [\![(\nu k)Q]\!] \mid H, \end{array}$$

and by definition and transitivity of strong bisimilarity we get

$$[\![(\nu k)P]\!] \mid H \xrightarrow{\tau}^* \sim [\![(\nu k)Q]\!] \mid H.$$

(b) if $k \notin \mathsf{fnn}(P)$, then $H = H_1$. By Lemma 5 there exist P' such that $P \equiv P'$ and $k \notin \mathsf{fn}(P')$. Since $\equiv$ is a congruence and by Lemma 4 we get $[\![P]\!] \sim [\![P']\!]$ and $[\![(\nu k)P]\!] \sim [\![(\nu k)P']\!]$. Then, by definition of the encoding and Proposition 3 we have

$$\begin{array}{rcl} [\![(\nu k)P]\!] \mid H_1 & \sim & [\![(\nu k)P']\!] \mid H_1 \\ & = & (\nu k, n_k, m_k)([\![P']\!] \mid H_k) \mid H_1 \\ & \sim & [\![P']\!] \mid (\nu k, n_k, m_k)H_k \mid H_1 \\ & \sim & [\![P']\!] \mid H_1 \sim [\![P]\!] \mid H_1. \end{array}$$

Since $P \to Q$ and $k \notin \mathsf{fnn}(P)$, by Lemma 5 we get $k \notin \mathsf{fnn}(Q)$. By the same lemma we conclude there exist Q' such that $Q \equiv Q'$ and $k \notin \mathsf{fn}(Q')$. Hence, again

$$\begin{array}{rcl} [\![Q]\!] \mid H_1 & \sim & [\![Q']\!] \mid H_1 \\ & \sim & [\![Q']\!] \mid (\nu k, n_k, m_k)H_k \mid H_1 \\ & \sim & (\nu k, n_k, m_k)([\![Q']\!] \mid H_k) \mid H_1 \\ & = & [\![(\nu k)Q']\!] \mid H_1 \sim [\![(\nu k)Q]\!] \mid H_1. \end{array}$$

By definition and transitivity of strong bisimilarity we get

$$[\![(\nu k)P]\!] \mid H_1 \xrightarrow{\tau}^* \sim [\![(\nu k)Q]\!] \mid H_1.$$

4. $P' \to Q'$ is derived from $P \to Q$, where $P \equiv P'$ and $Q \equiv Q'$. By induction hypothesis

$$[\![P]\!] \mid H_1 \xrightarrow{\tau}^* S,$$

where $S \sim [\![Q]\!] \mid H_1$, and if $\mathsf{fnn}(P) = \{k_1, \ldots, k_n\}$ then

$$H_1 = \prod_{i \in \{1,\ldots,n\}} H_{k_i},$$

while if $\mathsf{fnn}(P) = \emptyset$ then $H_1 = 0$. By Lemma 4 and Lemma 5, $P \equiv P'$ implies $[\![P]\!] \sim [\![P']\!]$ and $\mathsf{fnn}(P) = \mathsf{fnn}(P')$, and $Q \equiv Q'$ implies $[\![Q]\!] \sim [\![Q']\!]$. Then, by Proposition 3 we get $[\![P]\!] \mid H_1 \sim [\![P']\!] \mid H_1$, hence, by definition of strong bisimilarity

$$[\![P']\!] \mid H_1 \xrightarrow{\tau}^* S',$$

where $S' \sim S \sim [\![Q]\!] \mid H_1 \sim [\![Q']\!] \mid H_1$, which completes the proof.

As a direct consequence of Lemma 2, we get the operational correspondence result for the encoding of closed π-calculus processes.

Theorem 2 Let P be a closed (sum-free) π-calculus process. If $P \to Q$ then $[\![P]\!] \xrightarrow{\tau}^* \sim [\![Q]\!]$.