


A COMPOSITIONAL ENCODING OF π -CALCULUS INTO C_π -CALCULUS

Ivan Prokić¹ 

<https://doi.org/10.24867/META.2024.20>

Original scientific paper

Abstract. Recently, a fragment of the π -calculus, named C_π -calculus, has been introduced as a model for confidential name passing in concurrent systems. The key feature of the C_π -calculus is disabling forwarding in name passing. It has been shown that even being a fragment of the π -calculus, it is possible to recover the full π -expressivity in this fragment. This was done via an encoding of π -processes into the C_π -processes and showing that such an encoding satisfies some desired properties. However, one property this encoding does not satisfy is compositionality. In this paper, we propose a new encoding that satisfies this property. We evaluate the encoding with an example and leave the proof of operational correspondence for future work.

AMS Mathematics Subject Classification (2020): 68Q85

Key words and phrases: process calculi, π -calculus, expressiveness, encodings

1. Introduction

The π -calculus is a well-known and well-developed model for expressing mobile and concurrent systems [1]. In the process algebra community, it is frequently considered the algebra for concurrent computation in the same way that λ -calculus is viewed as the model for sequential computation. In the past decades, great work has been conducted in proposing variants of the π -calculus, mostly by extending its syntax and semantics with many new constructs that allow for a direct expression of several distinguished features of concurrent systems. On the other hand, a few works spotted that to get some features of concurrent systems, instead of extending the π -calculus, one can consider only its fragment. The most important work in this direction is the modeling of asynchronous communications [2], but also the modeling of internal mobility [3], and locality [4].

Recently, another fragment of the π -calculus, called C_π -calculus, appeared [5] that allows for the modeling of confidential name passing in concurrent systems. This model makes a simple syntactic restriction that disallows one of the key features of the π -calculus - the forwarding, in such a way making

¹Department of Fundamental Sciences, Faculty of Technical Sciences, University of Novi Sad, e-mail: prokic@ftn.uns.ac.rs

π	::=		<i>Prefixes</i>
		$a!k$	output
		$a?x$	input
		$[a = b]\pi$	matching
P	::=		<i>Process terms</i>
		0	termination
		$\pi.P$	prefix
		$P \mid P$	parallel composition
		$(\nu k)P$	name restriction
		$!P$	replication

Table 1: Syntax of C_π

it possible to model confidential communications directly. However, the same paper showed that the full expressivity of the π -calculus is not compromised in this fragment, since it is possible to encode it in the C_π . The encoding presented in [5] is shown correct through operational correspondence. However, this encoding comes with one limitation - it relies on the specific “handler” processes that make the encoding non-compositional. In this paper, we give another encoding that is compositional by the definition. Here, we only define the encoding and provide an example. Proof of the operational correspondence for this encoding will be our future work.

2. The syntax and semantics

The syntax of the C_π -calculus is given in Table 1. It consists of three prefixes: output, input, and matching. The prefixes are then used to build prefixed processes, which can also be terminated, a parallel composition, name restriction, and replication. All of the process constructs are explained in more detail later in the operational semantics. The syntax of processes relies on the notion of free and bound names. All names in processes are said to be free except for names in name restriction construct (k) and placeholder names in input (x) - these are called bound. The syntax of the π -calculus is the same as in Table 1, with only one distinction. In C_π we have a restriction that the name to be sent in output prefix (k) cannot be bound as a placeholder in the input prefix latter - thus preventing forwarding. For instance, $a?x.a!x.0$ is not a C_π process, but it is a valid π process.

The semantics of the C_π -calculus is given in terms of reduction relation in Table 3, which relies on the structural congruence relation given in Table 2. The first three rules of the structural congruence relation define the set of processes with respect to the parallel operator being a commutative monoid. The following three rules deal with name restriction, they allow for garbage collection, scope extrusion, and name swapping. The last two rules allow for removing matching if the matched names coincide, and the final allows for replication. The first reduction rule defines how two processes working in par-

$$\begin{aligned}
 P \mid 0 &\equiv P & P \mid Q &\equiv Q \mid P & (P \mid Q) \mid R &\equiv P \mid (Q \mid R) & (\nu k)0 &\equiv 0 \\
 P \mid (\nu k)Q &\equiv (\nu k)(P \mid Q) \text{ if } k \notin \text{fn}(P) & (\nu k)(\nu l)P &\equiv (\nu l)(\nu k)P \\
 [a = a]\pi.P &\equiv \pi.P & !P &\equiv P \mid !P
 \end{aligned}$$

Table 2: Structural congruence.

$$\begin{array}{c}
 \text{(R-COMM)} \\
 \frac{}{k!l.P \mid k?x.Q \rightarrow P \mid Q\{l/x\}} \\
 \\
 \text{(R-PAR)} \\
 \frac{P \rightarrow Q}{P \mid R \rightarrow Q \mid R} \\
 \\
 \text{(R-RES)} \\
 \frac{P \rightarrow Q}{(\nu k)P \rightarrow (\nu k)Q} \\
 \\
 \text{(R-STRU)} \\
 \frac{P \equiv P' \rightarrow Q' \equiv Q}{P \rightarrow Q}
 \end{array}$$

Table 3: Reduction relation.

allel can establish communication. The left process is output prefixed, it is ready to send l over name k . The right process is input prefixed, it is ready to receive a name over the same name k . Then, the synchronization of the output and the input happens and we obtain the processes where the prefixes have been consumed and the name l , that has been received in the inputting process is substituted for the placeholder x . The following three rules close the reduction relation under parallel composition, name restriction, and structural congruence relation. We remark that the semantics for the π -calculus is the same.

3. The encoding

Our new encoding from π -calculus terms into the C_π -calculus terms is given in Table 4. The encoding is presented as a function $\llbracket \cdot \rrbracket$. The encoding function is parameterized with a set of pairs of names ρ , whose purpose is to collect the names x bound in inputs (i.e. the placeholders) and pair them with m_x that are used as placeholders for name handlers for the received names (as explained later). Also, our target terms for our encoding are the polyadic C_π -processes. This is only for the simpler definition of the encoding, the polyadic C_π can be directly encoded into the one presented in the previous section in the standard way.

Our encoding distinguishes between the two types of output prefixes: ones that send the name directly (as in $a!k$) and those that forward the received name (as in $a?x.a!x$) The first send prefix is encoded in the scope of three fresh names m_k, e_1 , and e_2 . Name m_k is the name of the *handler process*, and the other two names are used only for blocking and synchronization of the sending and receiving processes. The matching construct does not change, but in the continuation, we have sending on name a fresh names e_1 and e_2 - first is going

$$\begin{aligned}
\llbracket [\tilde{c} = \tilde{d}]a!k.P \rrbracket_\rho &= (\nu m_k, e_1, e_2)([\tilde{c} = \tilde{d}]a!(e_1, e_2).m_k!(e_1).e_2!().\llbracket P \rrbracket_\rho \\
&\quad | !m_k?(x).x!(k, m_k).0) \\
\llbracket [\tilde{c} = \tilde{d}]a!x.P \rrbracket_{\rho, (x, m_x)} &= (\nu e_1, e_2)[\tilde{c} = \tilde{d}]a!(e_1, e_2).m_x!(e_1).e_2!().\llbracket P \rrbracket_{\rho, (x, m_x)} \\
\llbracket [\tilde{c} = \tilde{d}]a?x.P \rrbracket_\rho &= [\tilde{c} = \tilde{d}]a?(y, z).y?(x, m_x).z?().\llbracket P \rrbracket_{\rho, (x, m_x)} \\
\llbracket (\nu k)P \rrbracket_\rho &= (\nu k)\llbracket P \rrbracket_\rho \\
\llbracket P_1 \mid P_2 \rrbracket_\rho &= \llbracket P_1 \rrbracket_\rho \mid \llbracket P_2 \rrbracket_\rho \\
\llbracket !P \rrbracket_\rho &= !\llbracket P \rrbracket_\rho \\
\llbracket 0 \rrbracket_\rho &= 0
\end{aligned}$$

Table 4: Encoding of π processes into C_π processes.

to be used in the receiving process to communicate with the handler process, and the second to synchronize unlocking the encoding of the continuation of sending and receiving processes. Then, we have sending e_1 on m_k - that is to the handler process, and sending on e_2 - to synchronize unlocking directly with the receiving process. Finally, the encoding of the continuation process P follows. In parallel we have the handler process: it is ready to receive on m_k repeatedly and reply along the received name with k and m_k .

Encoding of the other sending prefix (the one that represents forwarding) goes as follows. First in ρ we find pair (x, m_x) . Then the rest is the same as for the other send prefix, except that now we do not have the handler process (since x is only a placeholder), and that instead of sending to the name handler (in the previous case m_k) here we have only its placeholder m_x .

The encoding of the receiving process goes as follows. The matching construct remains the same. Receiving happened along a with two names. The first (see e_1 in the encoding of output) is then used to receive from the handler, and, finally, the second name (see e_2 in the encoding of output) is used to synchronize with the sending process and to unlock the continuation. The encoding is homomorphism for the name restriction, parallel composition, replication, and inactive process.

Directly from the definition we have the compositionally result.

Theorem 3.1. *For all π -calculus processes P_1 and P_2 it holds that*

$$\llbracket P_1 \mid P_2 \rrbracket_\rho = \llbracket P_1 \rrbracket_\rho \mid \llbracket P_2 \rrbracket_\rho$$

As noted earlier, the proof of the operational correspondence is left for future work. Here we give an example of how our encoding works.

Example 3.2. Consider the π -calculus process

$$P = a!k.0 \mid a?x_1.b!x_1.0 \mid b?x_2.0$$

where in parallel we have the left process sending k along a , the middle process receiving the name and forwarding it to the right process along name b . Hence, we have the following reductions

$$P \rightarrow b!k.0 \mid b?x_2.0 \rightarrow 0$$

Following the encoding introduced in this paper we have

$$\begin{aligned} \llbracket P \rrbracket_\emptyset = & (\nu m_k, e_1, e_2)(a!(e_1, e_2).m_k!(e_1).e_2!().0 \mid !m_k?(x).x!(k, m_k).0) \\ & \mid a?(y, z).y?(x_1, m_{x_1}).z?().(\nu e'_1, e'_2)b!(e'_1, e'_2).m_{x_1}!(e'_1).e'_2!().0) \\ & \mid b?(y, z).y?(x_2, m_{x_2}).z?().0 \end{aligned}$$

Now we can see the encoding at work by showing the reductions step by step. The first synchronization is on name a , and the resulting process is

$$\begin{aligned} & (\nu m_k, e_1, e_2)(m_k!(e_1).e_2!().0 \mid !m_k?(x).x!(k, m_k).0) \\ & \mid e_1?(x_1, m_{x_1}).e_2?().(\nu e'_1, e'_2)b!(e'_1, e'_2).m_{x_1}!(e'_1).e'_2!().0) \\ & \mid b?(y, z).y?(x_2, m_{x_2}).z?().0 \end{aligned}$$

Here, the scope of names m_k , e_1 , and e_2 is now extruded to the receiving process, which is now ready to receive along e_1 . But, before that, the sending process has to activate the handler process on m_k , which leads to

$$\begin{aligned} & (\nu m_k, e_1, e_2)(e_2!().0 \mid e_1!(k, m_k).0 \mid !m_k?(x).x!(k, m_k).0) \\ & \mid e_1?(x_1, m_{x_1}).e_2?().(\nu e'_1, e'_2)b!(e'_1, e'_2).m_{x_1}!(e'_1).e'_2!().0) \\ & \mid b?(y, z).y?(x_2, m_{x_2}).z?().0 \end{aligned}$$

The receiving process (in the middle line) can receive the name k , along with the name of the handler m_k , reducing to

$$\begin{aligned} & (\nu m_k, e_1, e_2)(e_2!().0 \mid !m_k?(x).x!(k, m_k).0) \\ & \mid e_2?().(\nu e'_1, e'_2)b!(e'_1, e'_2).m_k!(e'_1).e'_2!().0) \\ & \mid b?(y, z).y?(x_2, m_{x_2}).z?().0 \end{aligned}$$

The unlocking along e_2 happens, and we end up with

$$\begin{aligned} & (\nu m_k, e_1, e_2)(!m_k?(x).x!(k, m_k).0) \\ & \mid (\nu e'_1, e'_2)b!(e'_1, e'_2).m_k!(e'_1).e'_2!().0) \\ & \mid b?(y, z).y?(x_2, m_{x_2}).z?().0 \end{aligned}$$

Now the communication along name b takes place

$$\begin{aligned} & (\nu m_k, e_1, e_2)(!m_k?(x).x!(k, m_k).0) \\ & \mid (\nu e'_1, e'_2)(m_k!(e'_1).e'_2!().0) \\ & \mid e'_1?(x_2, m_{x_2}).e'_1?().0) \end{aligned}$$

Because of our parameterized encoding, name m_k is now in place in the process in the middle line, hence it can connect with the handler process, which in turn sends k and m_k to the process in the last line. Hence, in two reduction steps, we end up with

$$\begin{aligned} & (\nu m_k, e_1, e_2)(!m_k?(x).x!(k, m_k).0) \\ & \mid (\nu e'_1, e'_2)(e'_2!().0) \\ & \mid e'_1?().0) \end{aligned}$$

Now the unlocking happens and we finally obtain (using the structural congruence) only the restricted handler process

$$(\nu m_k)!m_k?(x).x!(k, m_k).0$$

Since the handler process starts with sending along a restricted name it is behaviorally equivalent to the terminated process 0.

When compared with the encoding given in [5], our encoding has the following differences:

- it does not rely on the global handler processes and does not use outer and inner encodings, instead
- it introduces handler processes for each sending prefix (that is not forwarding).

For these reasons, we believe encoding from this paper is more natural - confirmed by the compositionality result obtained here. However, considering the proof of operational correspondence, we predict certain complications, since in the approach presented here the handler processes can appear under replication. In [5] this was not the case, since there the handler processes were introduced at the top level.

Acknowledgement

The author acknowledges the financial support of the Department of Fundamentals Sciences, Faculty of Technical Sciences, University of Novi Sad, in the frame of Project “Improving the teaching process in the English language in fundamental disciplines”.

References

- [1] D. Sangiorgi and D. Walker, *The Pi-Calculus - a theory of mobile processes*, Cambridge University Press, 2001.
- [2] K. Honda and M. Tokoro, “An Object Calculus for Asynchronous Communication”, *ECOOP’91 European Conference on Object-Oriented Programming*, Geneva, Switzerland, July 15-19, 1991.
- [3] D. Sangiorgi, “pi-Calculus, Internal Mobility, and Agent-Passing Calculi”, *Theoretical Computer Science*, vol. 167, pp. 235–274, 1996.
- [4] M. Merro and D. Sangiorgi, “On asynchrony in name-passing calculi”, *Mathematical Structures in Computer Science*, vol. 14, pp. 715–767, 2004.
- [5] I. Prokić and H. T. Vieira, “The C_π -calculus: A model for confidential name passing”, *Journal of Logical and Algebraic Methods in Programming*, vol. 119, 2021.