Communication Errors in the π-Calculus are Undecidable

Vasco T. Vasconcelos
Department of Informatics
Faculty of Sciences, University of Lisbon

António Ravara *
Department of Mathematics
Lisbon Institute of Technology

Keywords: concurrency, programming calculi, program correctness

Abstract

We present an undecidability proof of the notion of communication errors in the polyadic π-calculus. The demonstration follows a general pattern of undecidability proofs – reducing a well-known undecidable problem to the problem in question. We make use of an encoding of the λ-calculus into the π-calculus to show that the decidability of communication errors would solve the problem of deciding whether a lambda term has a normal form.

Introduction. The detection of communication errors in process calculi is crucial to ensure the safety of concurrent programs, i.e., the absence of run-time errors. The usual approach is to develop a type system, which is sound w.r.t. the notion of error, but, in general, not complete. The notions of communication errors are usually undecidable, what makes the type approach relevant. For the polyadic π-calculus [7] there is a strong believe that it is also the case.

Hereby we show that the notion of communication error in the polyadic π-calculus is undecidable. The proof follows a general pattern of undecidability proofs [4]: we reduce the problem of deciding whether a lambda term has a normal form [5] to the problem of deciding whether a process is an error. More precisely, we define a computable function $\ulcorner\cdot\urcorner$ from λ-terms into π-terms, and show that the decidability of '$\ulcorner M\urcorner \in \textsc{Err}$' implies the decidability of '$M\downarrow$', for which we conclude immediately that '$\ulcorner M\urcorner \in \textsc{Err}$' is undecidable.

This result, although not surprising, is up to the authors' knowledge, original. It shows an important application of the encodings of the λ-calculus into the π-calculus – the transference of results. It also shows that only by indirect means one can statically detect possible run-time errors in concurrent programs, for example, by using type systems.

The asynchronous (polyadic) π-calculus. We briefly present the asynchronous (polyadic) π-calculus [3, 6]. Consider a countable set of *names* a, b, v, x, and let $\tilde{v}$ stand for a sequence of names and $\tilde{x}$ for a sequence of pairwise distinct names.

Definition 1. The set of *processes* is given by the following grammar.

$$P \quad ::= \quad \overline{a}[\tilde{v}] \quad | \quad a(\tilde{x}).P \quad | \quad P \mid Q \quad | \quad \nu x\, P \quad | \quad !\, a(\tilde{x}).P \quad | \quad \mathbf{0}$$

An input prefixed process $a(\tilde{x}).P$ receives a sequence of values $\tilde{v}$ along a and becomes $P[\tilde{v}/\tilde{x}]$ (that is only defined when the sequences $\tilde{x}$ and $\tilde{v}$ have the same length). An output prefixed

*Corresponding author email: amar@math.ist.utl.pt

process $\overline{a}[\tilde{v}]$ sends the sequence of values $\tilde{v}$ along a. $\mathbf{0}$ is the terminated process, $P \mid Q$ is the parallel composition of processes, and $\nu a\, P$ restricts the scope of the name a to the process P. Process $!\, a(\tilde{x}).P$ is a persistent input prefixed process.

Definition 2.

1. An occurrence of a name x in a process P is *bound* if it is in a part of P with the form $a(\tilde{w}x\tilde{y}).P$ or $\nu x\, P$; otherwise the occurrence of x is *free*. We define accordingly the sets $fn(P)$ and $bn(P)$ of the *free names* and the *bound names* in a process P.
2. *Alpha-conversion* is denoted by $\equiv_\alpha$.
3. The process $P[\tilde{v}/\tilde{x}]$ denotes the *simultaneous substitution* in P of the free occurrences of each name in $\tilde{x}$ by the respective names in $\tilde{v}$; it is defined only when $\tilde{x}$ and $\tilde{v}$ are of the same length.

We abbreviate to $\nu\tilde{x}\, P$ a process $\nu x_1 \cdots \nu x_n\, P$. Consider that the operator 'ν' binds tighter than the operator '|'.

The operational semantics is defined via a structural relation defining static rules, and a labelled transition relation defining dynamic rules.

Definition 3. The *structural congruence* relation is inductively defined by the following rules.

$$P \equiv Q \text{ if } P \equiv_\alpha Q$$

$$P \mid \mathbf{0} \;\equiv\; P \qquad P \mid Q \;\equiv\; Q \mid P \qquad (P \mid Q) \mid R \equiv\; P \mid (Q \mid R)$$

$$\nu x\, \mathbf{0} \equiv \mathbf{0} \qquad \nu xy\, P \equiv \nu yx\, P \qquad \nu x\, P \mid Q \equiv \nu x\, (P \mid Q) \text{ if } x \notin fn(Q)$$

In the sequel we consider processes modulo structural congruence.

Definition 4. The set of *action labels* is given by the following grammar.

$$\alpha \quad ::= \quad \tau \quad \mid \quad a(\tilde{v}) \quad \mid \quad \nu\tilde{x}\, \overline{a}[\tilde{v}] \quad \text{where } \{\tilde{x}\} \subseteq \{\tilde{v}\} \setminus \{a\}$$

The *silent action* τ denotes internal communication in the process; the *input action* $a(\tilde{v})$ represents the reception in the name a of a sequence of names $\tilde{v}$; the *output action* $\nu\tilde{x}\, \overline{a}[\tilde{v}]$ represents the emission to the name a of a sequence of names $\tilde{v}$, some of them bound (those in $\tilde{x}$; the name a is free to allow the message to be received). In the last two action labels, the name a is called the *subject* of the label, and the names $\tilde{v}$ are the *objects* of the label.

Definition 5. The set $fn(\alpha)$ of the *free names* of the action label α is defined as follows.

$$fn(\tau) = \emptyset, \qquad fn(a(\tilde{v})) = \{a\} \cup \{\tilde{v}\}, \qquad fn(\nu\tilde{x}\, \overline{a}[\tilde{v}]) = \{a\} \cup (\{\tilde{v}\} \setminus \{\tilde{x}\})$$

The set $bn(\alpha)$ of the *bound names* of the action label α is defined accordingly.

We define the operational semantics via an *asynchronous transition relation* – the smallest relation generated by the rules in figure 1. Notice that the parallel composition and the scope restriction operators preserve the transition relation.

Let $\Longrightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, and let $\overset{\alpha}{\Longrightarrow}$ denote $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$. We define below the set of processes with a communication error.

Definition 6 (Error-processes [11]). The set Err of π-processes with a communication error is the following set:

$$\{P \;\mid\; P \implies \nu\tilde{u}\, (\overline{a}[v_1 \cdots v_n] \mid a(x_1 \cdots x_m).Q \mid R), \text{ and } n \neq m\}.$$

(In) $a(\tilde{x}).P \xrightarrow{a(\tilde{v})} P[\tilde{v}/\tilde{x}]$ (Out) $\overline{a}[\tilde{v}] \xrightarrow{\overline{a}[\tilde{v}]} \mathbf{0}$

(RIn) $!\,a(\tilde{x}).P \xrightarrow{a(\tilde{v})} !\,a(\tilde{x}).P \mid P[\tilde{v}/\tilde{x}]$ (Com) $a(\tilde{x}).P \mid \overline{a}[\tilde{v}] \xrightarrow{\tau} P[\tilde{v}/\tilde{x}]$

(RCom) $!\,a(\tilde{x}).P \mid \overline{a}[\tilde{v}] \xrightarrow{\tau} !\,a(\tilde{x}).P \mid P[\tilde{v}/\tilde{x}]$ (Par) $\dfrac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R}\ (bn(\alpha) \cap fn(R) = \emptyset)$

(Res) $\dfrac{P \xrightarrow{\alpha} Q}{\nu x\, P \xrightarrow{\alpha} \nu x\, Q}\ (x \notin fn(\alpha) \cup bn(\alpha))$ (Open) $\dfrac{P \xrightarrow{\nu \tilde{x}\, \overline{a}[\tilde{v}]} Q}{\nu x\, P \xrightarrow{\nu x \tilde{x}\, \overline{a}[\tilde{v}]} Q}\ (a \notin \{x\tilde{x}\})$

Figure 1: *Asynchronous transition relation.*

We finally define an equivalence relation over processes that takes into account the number of τ-actions performed.

Definition 7 (Expansion [2, 9]). A relation $\mathcal{R}$ over processes is an expansion if $P\mathcal{R}Q$ implies:

1. If $P \xrightarrow{\alpha} P'$, then $\exists Q'\ Q \xrightarrow{\alpha} Q'$ and $P'\mathcal{R}Q'$;
2. If $Q \xrightarrow{\tau} Q'$, then either $P\mathcal{R}Q'$, or $\exists P'\ P \xrightarrow{\tau} P'$ and $P'\mathcal{R}Q'$;
3. If $Q \xrightarrow{\alpha} Q'$, where α is an input or an output action, then $\exists P'\ P \xrightarrow{\alpha} P'$ and $P'\mathcal{R}Q'$.

We say that Q expands P, denoted $P \lesssim Q$, if $P\mathcal{R}Q$ for some expansion $\mathcal{R}$. The expansion relation is a preorder and a congruence.

The undecidability proof. We make use of an encoding of the lazy λ-calculus [1] into the π-calculus to show that the decidability of the notion of communication errors in the π-calculus would solve the problem of deciding whether a lambda term has a normal form.

We present an encoding of the lazy λ-calculus into the π-calculus [8, 10]. It is a version by Sangiorgi (for the asynchronous π-calculus) of the original one proposed by Milner.

Definition 8 (Encoding of the lazy λ-calculus [10]).

$$\begin{aligned} [\![\lambda x.M]\!]_p &\stackrel{\text{def}}{=} p(xq).[\![M]\!]_q \\ [\![x]\!]_p &\stackrel{\text{def}}{=} \overline{x}[p] \\ [\![MN]\!]_p &\stackrel{\text{def}}{=} \nu uv\,([\![M]\!]_u \mid \overline{u}[vp] \mid !\,v(q).[\![N]\!]_q) \end{aligned}$$

Henceforth, assume that M is a closed term, unless otherwise stated.

Before proving the result we present some crucial lemmas. The first is rather intuitive, stating when name substitution in processes preserves error freedom. The second is adapted from results in the literature [10], regarding properties of the encoding, namely preservation of termination and divergency. The third is original and not trivial: it guarantees the absence of communication errors in encoded terms. Up to the authors knowledge this is usually done indirectly, via a type system. We present a direct proof.

Lemma 1. If $P \notin \text{Err}$ and $v \notin fn(P)$ then $P[v/x] \notin \text{Err}$.

Proof. If x does not occur free in P then the result holds trivially. Otherwise, since $v \notin fn(P)$ and all redexes in P are good, there are no bad redexes in $P[v/x]$. Thus, the result holds. □

Notice that without the proviso '$v \notin fn(P)$' the lemma is false.

LEMMA 2.

1. If $[\![M]\!]_p \Longrightarrow \xrightarrow{\alpha}$, then $\alpha = p(xq)$ and $M\downarrow$.
2. If $[\![M]\!]_p \stackrel{\alpha}{\Longrightarrow}$ where α is an output action with subject x, then $\alpha = \overline{x}[v]$ or $\alpha = \overline{x}(v)$, for some name v (notice that x must be free in M).
3. If $M\uparrow$, then $[\![M]\!]_p \stackrel{\alpha}{\not\Rightarrow}$, for all α.
4. If $M \Longrightarrow \lambda x.N$, then $[\![M]\!]_p \stackrel{p(xq)}{\Longrightarrow} \gtrsim [\![N]\!]_q$.

Proof. The first clause follows from the second clause in proposition 5.5 [10] (recall that M is closed). The second clause follows from the conjunction of clauses 2 to 4 in proposition 5.4 [10]. The third clause follows from the contrapositive of the conjunction of clauses 2 to 4 in the proposition 5.5 [10] (the conjunction of the clauses implies that if $\exists\alpha\ [\![M]\!]_p \stackrel{\alpha}{\Longrightarrow}$ then $M\downarrow$). The fourth clause is the second clause in the proposition 5.4 [10]. □

LEMMA 3. $[\![M]\!]_p \notin$ ERR.

Proof. By structural induction on M. Observe that the encoding of a variable and the encoding of an abstraction are trivially not erroneous.

For the application, the encoding is, by definition, $[\![NL]\!]_p \stackrel{\text{def}}{=} \nu uv\,([\![N]\!]_u \mid \overline{u}[vp] \mid !\,v(q).[\![L]\!]_q)$. Since by induction hypothesis $[\![N]\!]_u$ and $[\![L]\!]_q$ are not erroneous, by definition no Q such that $[\![N]\!]_u \Longrightarrow Q$ is erroneous. Therefore, only the interaction between the process $[\![N]\!]_u$ and the process $R \stackrel{\text{def}}{=} \overline{u}[vp] \mid !\,v(q).[\![L]\!]_q$ can generate errors. There are two cases to consider:

1. Case $N \Longrightarrow \lambda x.N'$. By Lemma 2.4 we have that

$$[\![NL]\!]_p \Longrightarrow \nu v\,(P[vp/xq] \mid !\,v(q).[\![L]\!]_q) \qquad (\text{where } P[vp/xq] \gtrsim [\![N']\!]_p).$$

   As $[\![N]\!]_u$ is not erroneous by induction hypothesis, neither is P. By Lemma 1 also $P[vp/xq]$ is not erroneous. So, it is only the interaction between $P[vp/xq]$ and $!\,v(q).[\![L]\!]_q$ that can go wrong. Again, we have two cases to consider, depending on whether x is free in N' (and taking into account that $P[vp/xq] \gtrsim [\![N']\!]_p$):

   (a) if x does not occur free in N' then, by lemma 2.1 and lemma 2.2, there is no output action in $P[vp/xq]$, and thus, there is no interaction;
   (b) if x does occur free in N' then, by lemma 2.2, the interaction does not go wrong.

2. Case $N\uparrow$. Then Lemma 2.3 ensures that $[\![N]\!]_u \stackrel{\alpha}{\not\Rightarrow}$; thus there is no interaction with the process R, and nothing goes wrong.

We conclude that the encoding does not generate errors. □

We are now in a position to prove the result of this paper. Let Λ denote the set of lambda terms and Π denote the set of π-processes.

THEOREM. The problem '$P \in$ ERR' is undecidable.

Proof. Suppose that '$P \in$ ERR' is decidable. We show that '$M\downarrow$' is decidable.

Let $\ulcorner\cdot\urcorner : \Lambda \longrightarrow \Pi$ be such that $\ulcorner M\urcorner \stackrel{\text{def}}{=} \nu p\,([\![M]\!]_p \mid \overline{p}[])$.

We have to prove two assertions:

1. the function $\ulcorner\cdot\urcorner$ is computable;
2. if '$\ulcorner M\urcorner \in$ ERR' then '$M\downarrow$'.

The proof of the first assertion follows directly from definition 8. To prove the second assertion notice first that we know from Lemma 3 that $[\![M]\!]_p \notin \textsc{Err}$; then, $\ulcorner M \urcorner$ is an error only when the interaction of $[\![M]\!]_p$ and $\overline{p}[]$ causes the error. Hence, by Lemma 2.1, '$M\downarrow$'.

We attain an absurd, since we know that '$M\downarrow$' is undecidable (corollary 5.6.2, [5]). Therefore, we conclude that '$P \in \textsc{Err}$' is undecidable. □

Acknowledgements This work is partially supported by FCT, by PRAXIS XXI Projects PCEX/P/MAT/46/96 ACL, and 2/2.1/TIT/1658/95 LogComp, as well as ESPRIT IV Working Groups 22704 ASPIRE and 23531 FIREworks.

Special thanks to Carlos Caleiro, for very inspiring discussions on decidability proofs.

References

[1] Samson Abramsky. The lazy lambda calculus. *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1989.

[2] S. Arun-Kumar and Mathew Hennessy. An efficiency preorder of processes. *Acta Informatica* 29(8), pages 737–760, 1992.

[3] Gérard Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche RR–1702, INRIA Sophia-Antipolis, 1992.

[4] Nigel Cutland. *Computability: An Introduction to Recursive Function Theory.* Cambridge University Press, 1980.

[5] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus*, page 63. Cambridge University Press, 1986.

[6] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *5th European Conference on Object-Oriented Programming*, volume LNCS 512, pages 141–162. Springer-Verlag, 1991.

[7] Robin Milner. The polyadic π-calculus: a tutorial. Technical Report ECS-LFCS 91–180, University of Edinburgh, 1991. Appeared in *Proceedings of the International Summer School on Logic and Algebra of Specification*, Marktoberdorf, August 1991. Reprinted in *Logic and Algebra of Specification*, ed. F. L. Bauer, W. Brauer, and H. Schwichtenberg, Springer-Verlag, 1993.

[8] Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science* 2(2), pages 119–141, 1992. Available as Rapport de Recherche RR–1154, INRIA, Sophia Antipolis, 1990.

[9] Davide Sangiorgi and Robin Milner. The problem of "Weak bisimulation up to". In *3rd International Conference on Concurrency Theory*, volume LNCS 630, pages 32–46. Springer-Verlag, 1992.

[10] Davide Sangiorgi. Lazy functions and mobile processes. Rapport de Recherche RR–2515, INRIA, Sophia Antipolis, 1995.

[11] Vasco T. Vasconcelos and Kohei Honda Principal typing schemes in a polyadic π-calculus. In *4th International Conference on Concurrency Theory*, volume LNCS 715, pages 524–538. Springer-Verlag, 1993.