# Communication Errors in the $\pi$-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 $\pi$-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 $\lambda$-calculus into the $\pi$-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 with respect to the notion of error, but, in general, not complete. The notions of communication errors are usually undecidable, and this makes the type approach relevant. For the polyadic $\pi$-calculus [8] this is also the case.

Herein we show that the notion of communication errors in the polyadic $\pi$-calculus is undecidable. The proof follows a general pattern of undecidability results [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 $\lambda$-terms into $\pi$-terms, and show that the decidability of '$\ulcorner M\urcorner \in \textsc{Err}$' implies the decidability of '$M\!\downarrow$', from which we conclude immediately that '$\ulcorner M\urcorner \in \textsc{Err}$' is undecidable. An alternative proof—reducing the halting problem of Turing machines to our problem—would involve first the encoding of Turing machines in the $\pi$-calculus.

This result, although not surprising, is up to the authors' knowledge, original. It shows an important application of the encodings of the $\lambda$-calculus into the $\pi$-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 $\pi$-calculus.** We briefly present the asynchronous polyadic $\pi$-calculus [3, 6, 8]. Assume a countable set of *names* $a, b, p, q, u, v, x$, and let $\tilde{v}$ stand for a sequence of names, and $\tilde{x}$ for a sequence of pairwise distinct names.

Definition 1 (Processes). 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 the process $P$ where names in $\tilde{v}$ replace names in $\tilde{x}$ in $P$. An output process $\overline{a}[\tilde{v}]$ sends the sequence of names $\tilde{v}$ along $a$. $\mathbf{0}$ is the terminated process, $P \mid Q$ is the parallel composition of processes, and $\nu x\, P$ restricts the scope of the name $x$ to the process $P$. Process $!\, a(\tilde{x}).P$ is a persistent input prefixed process. We abbreviate to $\nu x_1 \cdots x_n\, P$ a process $\nu x_1 \cdots \nu x_n\, P$, and consider that the operator '$\nu$' binds tighter than the operator '$|$'.

$$
\begin{array}{llll}
(\textsc{In}) & a(\tilde{x}).P \xrightarrow{a[\tilde{v}]} P[\tilde{v}/\tilde{x}] & (\textsc{Out}) & \overline{a}[\tilde{v}] \xrightarrow{\overline{a}[\tilde{v}]} \mathbf{0} \\[2ex]
(\textsc{RIn}) & !\,a(\tilde{x}).P \xrightarrow{a[\tilde{v}]} !\,a(\tilde{x}).P \mid P[\tilde{v}/\tilde{x}] & (\textsc{Com}) & a(\tilde{x}).P \mid \overline{a}[\tilde{v}] \xrightarrow{\tau} P[\tilde{v}/\tilde{x}] \\[2ex]
(\textsc{RCom}) & !\,a(\tilde{x}).P \mid \overline{a}[\tilde{v}] \xrightarrow{\tau} !\,a(\tilde{x}).P \mid P[\tilde{v}/\tilde{x}] & (\textsc{Par}) & \dfrac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R}\ (\mathrm{bn}(\alpha) \cap \mathrm{fn}(R) = \emptyset) \\[3ex]
(\textsc{Res}) & \dfrac{P \xrightarrow{\alpha} Q}{\nu x\, P \xrightarrow{\alpha} \nu x\, Q}\ (x \notin \mathrm{fn}(\alpha) \cup \mathrm{bn}(\alpha)) & (\textsc{Open}) & \dfrac{P \xrightarrow{\nu\tilde{x}\,\overline{a}[\tilde{v}]} Q}{\nu x\, P \xrightarrow{\nu x\tilde{x}\,\overline{a}[\tilde{v}]} Q}\ (a \notin \{x\tilde{x}\})
\end{array}
$$

$$
(\textsc{Struct}) \quad \frac{P \equiv P' \quad P' \xrightarrow{\alpha} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha} Q}
$$

Figure 1: *Asynchronous transition relation.*

Definition 2 (Free names, substitution).

1. An occurrence of a name $x$ in a process $P$ is *bound* if it is in a part of $P$ with the form $a(\tilde{w}x\tilde{y}).Q$ or $\nu x\, Q$; otherwise the occurrence of $x$ is *free.*
2. We define accordingly the set $\mathrm{fn}(P)$ of the *free names* in a process $P$; *alpha-conversion*, denoted by $\equiv_\alpha$, is defined as for the $\lambda$-calculus.
3. The process $P[\tilde{v}/\tilde{x}]$ is the result of *simultaneously substituting* the names in $\tilde{v}$ for every free occurrence of the corresponding name in $\tilde{x}$ in $P$; it is defined only when $\tilde{x}$ and $\tilde{v}$ are sequences of the same length.

Definition 3 (Operational semantics).

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

$$
\begin{array}{lll}
P \equiv Q \text{ if } P \equiv_\alpha Q & & \\
P \mid \mathbf{0} \equiv P & P \mid Q \equiv Q \mid P & (P \mid Q) \mid R \equiv P \mid (Q \mid R) \\
\nu x\, \mathbf{0} \equiv \mathbf{0} & \nu xy\, P \equiv \nu yx\, P & \nu x\, P \mid Q \equiv \nu x\, (P \mid Q) \text{ if } x \notin \mathrm{fn}(Q)
\end{array}
$$

2. The set of *action labels* is given by the following grammar, where $\{\tilde{x}\} \subseteq \{\tilde{v}\} \setminus \{a\}$.

$$\alpha \quad ::= \quad \tau \quad \mid \quad a[\tilde{v}] \quad \mid \quad \nu\tilde{x}\,\overline{a}[\tilde{v}]$$

3. An occurrence of a name $x$ in an action label $\alpha$ is *bound* if the label is of the form $\nu\tilde{w}x\tilde{y}\,\overline{a}[\tilde{v}]$; otherwise the occurrence of $x$ is *free.* The sets $\mathrm{fn}(\alpha)$ and $\mathrm{bn}(\alpha)$ of the *free names* and the *bound names* in an action label $\alpha$ are defined accordingly.
4. The operational semantics is defined via an *asynchronous transition relation*—the smallest relation generated by the rules in Figure 1.

The *silent action* $\tau$ denotes internal communication within the process; the *input action* $a[\tilde{v}]$ represents the reception on 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 action label, and the names $\tilde{v}$ its *objects.* Let $\Longrightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, and let $\overset{\alpha}{\Longrightarrow}$ denote $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$.

Definition 4 (Error-processes [11]). The set Err of $\pi$-processes with a communication error is the following:

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

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

Definition 5 (Expansion [2, 10]). A relation $\mathcal{R}$ over processes is an expansion if $P\mathcal{R}Q$ implies:

1. If $P \xrightarrow{\alpha} P'$, then there is a $Q'$ such that $Q \overset{\alpha}{\Longrightarrow} Q'$ and $P'\mathcal{R}Q'$;
2. If $Q \xrightarrow{\tau} Q'$, then either $P\mathcal{R}Q'$, or there is a $P'$ such that $P \xrightarrow{\tau} P'$ and $P'\mathcal{R}Q'$;
3. If $Q \xrightarrow{\alpha} Q'$, where $\alpha$ is an input or an output action, then there is a $P'$ such that $P \xrightarrow{\alpha} P'$ and $P'\mathcal{R}Q'$.

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

**The undecidability proof.** We make use of Sangiorgi's version of Milner's original encoding of the lazy $\lambda$-calculus [1] into the $\pi$-calculus [7, 9].

Definition 6 (Encoding of the lazy $\lambda$-calculus [9]).

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

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

Before proving the result we present three crucial lemmas. The first is rather intuitive, stating when name substitution preserves error freedom. The second is adapted from results in the literature [9], and regards the preservation of termination and the divergence of encoded terms. 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 $\{\tilde{v}\} \cap \text{fn}(P) = \emptyset$, then $P[\tilde{v}/\tilde{x}] \notin \text{Err}$.

*Proof.* For each name $x_i$ in $\tilde{x}$, if $x_i$ does not occur free in $P$ then $P[v_i/x_i] = P$. Otherwise, observe that $P \notin \text{Err}$, that $v_i \notin \text{fn}(P)$, and that substitution replaces each name by exactly one name. Thus, by structural induction one easily verifies that $P[\tilde{v}/\tilde{x}] \notin \text{Err}$. $\square$

Notice that without the proviso '$\{\tilde{v}\} \cap \text{fn}(P) = \emptyset$' the result does not hold; just take the process $\overline{a}[] \mid b(x).\mathbf{0}$ and the substitution $[a/b]$.

Lemma 2. Let $M$ be closed in all the items below but the second.

1. If $[\![M]\!]_p \Longrightarrow\xrightarrow{\alpha}$, then $\alpha = p[xq]$ and $M\downarrow$.
2. If $[\![M]\!]_p \overset{\alpha}{\Longrightarrow}$ and $\alpha$ is an output action with subject $x$, then its object is some name $v$ (notice that $x$ must be free in $M$).
3. If $M\uparrow$, then for no $\alpha$, $[\![M]\!]_p \overset{\alpha}{\Longrightarrow}$.
4. If $M \Longrightarrow \lambda x.N$, then $[\![M]\!]_p \overset{p[xq]}{\Longrightarrow}\gtrsim [\![N]\!]_q$.

*Proof.* For the first clause, that $\alpha$ is an input action follows from the contrapositive of the conjunction of clauses 3 and 4 in Proposition 5.5 [9] (the conjunction of the clauses implies that if $\alpha$ is an output then $M$ is open), and from the fact that $M$ is closed; that $M$ converges follows from the second clause of the same proposition. The second clause follows from the conjunction of clauses 3 to 4 in Proposition 5.5 [9]. The third clause follows from the contrapositive of the conjunction of clauses 2 to 4 in the Proposition 5.5 [9] (the conjunction of the clauses implies that if there is an $\alpha$ such that $[\![M]\!]_p \overset{\alpha}{\Longrightarrow}$ then $M\downarrow$). The fourth clause is the second clause in the Proposition 5.4 [9]. $\square$

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 $P$ such that $[\![N]\!]_u \Longrightarrow P$ is erroneous. Therefore, only the interaction between the process $[\![N]\!]_u$ and the process $R \stackrel{\text{def}}{=} \overline{u}[vp] \mid\, !\,v(q).[\![L]\!]_q$ may generate errors. Since $M$ is closed there are two cases to consider:

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

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

   By Lemma 1, $P[vp/xq]$ 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, 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 $[\![N]\!]_u$ has no action; 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.

Theorem 4. The predicate '$P \in$ Err' is undecidable.

*Proof.* Suppose that '$P \in$ Err' is decidable. We show that '$M\!\downarrow$' is decidable.

Let $\ulcorner\cdot\urcorner$ be a function from $\lambda$-terms into $\pi$-terms 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. '$\ulcorner M\urcorner \in$ Err' if and only if '$M\!\downarrow$'.

The proof of the first assertion follows directly from definition 6. To prove the 'if' direction of the second assertion notice first that we know from Lemma 3 that $[\![M]\!]_p \notin$ 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, '$M\!\downarrow$'. For the 'only-if' direction, observe that since $M$ is closed, $M \Longrightarrow \lambda x.N$. Then, from Lemma 2.4 we know that $[\![M]\!]_p \Longrightarrow\xrightarrow{\alpha}$ and an error occurs in $\ulcorner M\urcorner$.

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

**Acknowledgements** This work is partially supported by FCT, by PRAXIS XXI Projects PRAXIS/P/MAT/10002/1998 ProbLog, 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, and to a referee for suggesting a fix to a previous version of the proof of the result.

## References

[1] Samson Abramsky. The lazy lambda calculus. In *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):737–760, 1992.

[3] Gérard Boudol. Asynchrony and the $\pi$-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.R̃oger Hindley and Jonathan P. Seldin. *Introduction to Combinators and $\lambda$-Calculus.* 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. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[8] Robin Milner. The polyadic $\pi$-calculus: a tutorial. In *Logic and Algebra of Specification, Proceedings of International NATO Summer School (Marktoberdorf, Germany, 1991)*, 1993.

[9] Davide Sangiorgi. Lazy functions and mobile processes. Rapport de Recherche RR–2515, INRIA, Sophia Antipolis, 1995. To appear in “Festschrift volume in honor of Robin Milner’s 60th birthday”, MIT Press.

[10] 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.

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