Reflection, Encodability and Separation*

Stian Lybech[1]

Dept. of Computer Science, Reykjavík University, Iceland
stian21@ru.is

1 Introduction

Process calculi are formalisms for modelling and reasoning about concurrent and distributed computations; a prominent example being the π-calculus of Milner et al. [7, 10]. It models computation as communication of channel names through named channels, which thus marks it as a purely first-order model. Later variants such as the Higher-Order π-calculus (HOπ) [9, 8] adds constructs for higher-order process mobility; but an early result by Sangiorgi [9, 8] showed that HOπ can be encoded in the first-order π-calculus, which took away much of the interest in the higher-order paradigm, since higher-order behaviour may thus be viewed as just an extension on top of an already computationally complete, first-order language.

However, higher-order behaviour may alternatively be viewed as a limited form of *reflection*; i.e. the ability of a program to turn code into data, modify or compute with it, and reinstantiate it as running code. This capability is inherent in the Reflective Higher-Order (RHO or ρ) calculus of Meredith and Radestock [5], and process mobility here appears as a special case where data (code) is just transmitted without modification. Surprisingly also, there is no encoding of the ρ-calculus into the π-calculus satisfying the validity criteria of Gorla [4], as we shall see below, which shows that higher-order behaviour cannot always be treated as a simple extension of the first-order paradigm.

2 The Reflective Higher-Order calculus

Definition 1 (ρ-calculus syntax)**.** *The ρ-calculus syntax is given by the grammar:*

$$\begin{aligned} P &::= \mathbf{0} \mid P_1 \mid P_2 \mid x\langle\!|P|\!\rangle \mid x(y).P \mid \urcorner x \ulcorner \\ x, y &::= \ulcorner P \urcorner \end{aligned}$$

Notably, names and processes are built from the *same* syntax, so if P is a process, then $\ulcorner P \urcorner$ is a *name*, pronounced *quote* P. The nil, input and parallel construct are as in the π-calculus. The *lift* construct $x\langle\!|P|\!\rangle$ quotes the process P, thereby creating the name $\ulcorner P \urcorner$, and outputs it on x; thus name generation is handled explicitly in the calculus, rather than implicitly by a π-calculus style ν-operator. Lastly, the *drop* construct $\urcorner x \ulcorner$ removes the quotes of the name to run the process within; this construct is thus similar to a process variable X in e.g. HOπ.

Definition 2 (ρ-calculus semantics)**.** *The semantics is given by the standard rules for parallel composition and structural congruence (as in e.g. the π-calculus) plus the following rule for communication:*

$$[\rho\text{-COM}]\frac{x_1 \equiv_N x_2}{x_1(y).P \mid x_2\langle\!|Q|\!\rangle \to P\{\ulcorner Q \urcorner / y\}}$$

*Much of this work was done in collaboration with Hans Hüttel, Bjarke B. Bojesen and Alex R. Bendixen at Aalborg University.

The *name equivalence* relation $\equiv_{\mathcal{N}}$ extends structural congruence to names, since names are now structured terms. Lastly, the drop operation is performed by syntactic substitution (again similar to process variables in HOπ):

$$\ulcorner x_1 \urcorner \{\ulcorner P \urcorner / x_2\} = P \quad \text{if } x_1 \equiv_{\mathcal{N}} x_2$$

The lack of a ν-operator means that all names are globally visible, and this in turn affects how barbed bisimulation may be defined for this calculus to facilitate comparison with other calculi that do have this operator. Thus, Meredith and Radestock [5] define an *$\mathcal{N}$-restricted observation predicate* $\downarrow^{\mathcal{N}}$, parametrised with a set $\mathcal{N}$ of names, such that $P \downarrow^{\mathcal{N}} x$ only holds if P is observable at x *and* $x \in \mathcal{N}$. Their definition of (weak) barbed bisimulation is then standard, except that it too must be parametrised accordingly, and the resulting notion of behavioural equivalence is then called *weak, $\mathcal{N}$-restricted barbed bisimilarity*, written $\approx^{\mathcal{N}}_{\downarrow}$.

3 Encodability and separation

We shall now consider some encodability and separation results that characterise the relationship between the ρ-calculus and other, more well-known calculi. The first concerns an encoding $[\![\ ]\!] : A\pi \to \rho$, originally due to Meredith and Radestock [5]:

Theorem 1 (Meredith & Radestock)**.** $P_1 \approx_\pi P_2 \iff [\![P_1]\!] \approx^{\phi(\mathrm{fn}(P_1))}_{\downarrow} [\![P_2]\!]$
where $\approx_\pi$ is weak barbed bisimilarity in the asynchronous π-calculus, and $\phi : \mathcal{N}_\pi \to \mathcal{N}_\rho$ is a translation of π-calculus names into ρ-calculus names using any suitable name generation discipline (see e.g. [1] for an example).

The encoding works by firstly mapping each name in P_1, P_2 into ρ-calculus names, using a suitable name generation discipline (the function ϕ). $(\boldsymbol{\nu} x)\, P$ is translated as an input $p(x).[\![P]\!]$, representing a request for a fresh name, and the translation builds a distributed name generator alongside the translated processes to serve such requests, by dynamically generating names from a distinct name space that are ensured never to clash with the ϕ-mapping. By restricting the observation predicate to the ϕ-translated set of *free names* in P_1, it is thus ensured that none of these dynamically generated names can be observed.[1]

However, the converse of this theorem does not hold: the π-calculus cannot encode the ρ-calculus without introducing divergence. The proof of this relies on a separation result originally due to Carbone and Maffeis [3], showing that their ${}^e\pi$-extension cannot be encoded into the monadic π-calculus. ${}^e\pi$ is an extension of the monadic π-calculus that allows *vectors* of names to appear in subject position of input and output. Let the *match degree* $\mathrm{MD}(\mathcal{L})$ of a language $\mathcal{L}$ denote the least upper bound on the number of names that must be matched to yield a reduction in $\mathcal{L}$, corresponding to the length of the vectors in subject position. Then $\mathrm{MD}(\pi) = 1$ and $\mathrm{MD}({}^e\pi) = \infty$. The theorem was rederived by Gorla as follows: for two languages $\mathcal{L}_1, \mathcal{L}_2$, if $\mathrm{MD}(\mathcal{L}_1) > \mathrm{MD}(\mathcal{L}_2)$, then there is no encoding of $\mathcal{L}_1$ into $\mathcal{L}_2$ satisfying the validity criteria proposed in [4]. It follows immediately that ${}^e\pi$ cannot be encoded in the π-calculus since $\mathrm{MD}({}^e\pi) > \mathrm{MD}(\pi)$.

But the ρ-calculus *can* encode ${}^e\pi$: by using a suitable convention of name generation, we can encode ${}^e\pi$-like name composition $x_1 \cdot x_2 \cdot \ldots \cdot x_n$ in the ρ-calculus by composing the processes

[1] Note that the original translation by Meredith and Radestock in [5] contains a subtle error, allowing two π-processes P_1 and P_2 to be created, such that $P_1 \not\approx_\pi P_2$ but where nevertheless $[\![P_1]\!] \approx^{\varphi(\mathrm{fn}(P_1))}_{\downarrow} [\![P_2]\!]$. We show this in [1] along with an amended translation.

within the names. Assume that X_i is the process within the x_i'th name; then we can define

$$\ulcorner X_1 \urcorner \cdot \ulcorner X_2 \urcorner \cdot \ldots \cdot \ulcorner X_n \urcorner \triangleq \ulcorner X_1 \mid X_2 \mid \ldots \mid X_n \urcorner$$

which can be extended to vectors of arbitrary length. This composition can be performed at runtime through the ρ-calculus' lift operator, and we can thus encode ${}^e\pi$ input and output operations with n-ary subjects as follows:

$$\begin{aligned} [\![x_1 \cdot \ldots \cdot x_n(y).P]\!]_a &\triangleq a(\!|\ulcorner x_1 \urcorner \mid \ldots \mid \ulcorner x_n \urcorner|\!) \mid a(v).v(y).[\![P]\!]_{a+} \\ [\![\overline{x_1 \cdot \ldots \cdot x_n}\langle z\rangle]\!]_a &\triangleq a(\!|\ulcorner x_1 \urcorner \mid \ldots \mid \ulcorner x_n \urcorner|\!) \mid a(v).v(\!|\ulcorner z \urcorner|\!) \end{aligned}$$

where we assume all the names x_i are implemented as quoted processes $\ulcorner X_i \urcorner$; and where the parameter a is an internal name that is chosen fresh for each translation, and $a+$ is derived from a by a suitable convention of name incrementation, that is ensured to never cause a name clash.[2] The remaining operators are as in the encoding of the π-calculus. Thus, for ${}^e\pi$-processes P_1, P_2 we have the following result:

Theorem 2. $P_1 \approx P_2 \iff [\![P_1]\!] \approx_{\downarrow}^{\phi(\mathrm{fn}(P_1))} [\![P_2]\!]$

The proof is similar to that for the π-calculus translation; see [2] and [5] for details. The key point is that the a names are generated by the translation, and are therefore not in the set of free names of P_1. Thus, they are unobservable by the $\downarrow^{\phi(\mathrm{fn}(P_1))}$ predicate.

Corollary 1. *There is no encoding* $[\![\ ]\!] : \rho \to \pi$ *satisfying the validity criteria of [4].*

This is proved by contradiction: if such an encoding existed, it could be composed with the encoding of ${}^e\pi$ in the ρ-calculus to yield an encoding of ${}^e\pi$ into the π-calculus, in contradiction of the aforementioned theorem by Gorla [4].

Another result relates the ρ-calculus to other higher-order calculi such as CHOCS [11] and HOπ. Both of these can encode the lazy λ-calculus such that λ-variables x are translated as process variables; this is a natural extension of the name invariance criterion of [4] for higher-order calculi. But even though the ρ-calculus can encode the π-calculus, and thus, through Milner's encoding [6], the λ-calculus, there is no encoding satisfying the aforementioned criterion:

Theorem 3. *There is no valid encoding* $[\![\ ]\!] : \lambda \to \rho$ *satisfying that* $[\![x]\!] = \ulcorner \phi(x) \urcorner$

Details of the proof are given in [1]; the crucial detail is that there is no means in the ρ-calculus to alter a process once it has been quoted: it can only be dropped. This means we cannot control the order of evaluation of more than two terms, e.g. $[\![e_1 e_2 e_3]\!]$, which thus allows us to derive a contradiction; either of operational correspondence or divergence reflection. From this, we immediately obtain a similar corollary of separation w.r.t. HOπ:

Corollary 2. *There is no valid encoding* $[\![\ ]\!] : \mathcal{H} \to \rho$ *satisfying that* $[\![X]\!] = \ulcorner \phi(X) \urcorner$.

Interestingly, this separation result does not seem to derive from any difference in the higher-order nature of these calculi, but rather from the absence of a scoped, dynamic name binding construct $[x/y]$ (as in CHOCS), or a mechanism for abstraction/application (as in HOπ). If either of these constructs were added to the ρ-calculus, then it would indeed become possible to create an encoding of the λ-calculus, satisfying the aforementioned criteria.

[2] Note that this encoding disregards the ordering of names within the name vector, due to the commutativity of parallel composition. However, this is acceptable since the separation result for ${}^e\pi$ by [4] only relies on the match degree, i.e. the *number* of names in the vector, and not their ordering.

References

[1] Alexander R. Bendixen, Bjarke B. Bojesen, and Stian Lybech. Encodability and typability of reflective higher-order languages. Technical report, Department of Computer Science, Aalborg University, January 2021.

[2] Alexander R. Bendixen, Bjarke B. Bojesen, and Stian Lybech. Typing reflection in higher-order psi-calculi. Technical report, Department of Computer Science, Aalborg University, June 2021.

[3] Marco Carbone and Sergio Maffeis. On the expressive power of polyadic synchronisation in pi-calculus. *Nordic Journal of Computing*, 10(2):70–98, 2003.

[4] Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, 2010.

[5] L.G. Meredith and Matthias Radestock. A reflective higher-order calculus. *Electronic Notes in Theoretical Computer Science*, 141(5):49 – 67, 2005. Proceedings of the Workshop on the Foundations of Interactive Computation (FInCo 2005).

[6] Robin Milner. Functions as processes. *Mathematical structures in computer science*, 2(2):119–141, 1992.

[7] Robin Milner. The polyadic π-calculus: a tutorial. In *Logic and Algebra of Specification*, pages 203–246. Springer Berlin Heidelberg, 1993.

[8] Davide Sangiorgi. *Expressing mobility in process algebras: first-order and higher-order paradigms*. PhD thesis, University of Edinburgh, 1993.

[9] Davide Sangiorgi. From π-calculus to higher-order π-calculus — and back. In M. C. Gaudel and J. P. Jouannaud, editors, *TAPSOFT'93: Theory and Practice of Software Development*, pages 151–166. Springer Berlin Heidelberg, 1993.

[10] Davide Sangiorgi and David Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

[11] Bent Thomsen. A calculus of higher order communicating systems. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL' 89*, POPL'89, pages 143–154, New York, NY, USA, 1989. ACM Press.