On the Expressiveness of Linearity vs Persistence in the Asychronous Pi-Calculus

Catuscia Palamidessi
INRIA

Vijay Saraswat
IBM

Frank Valencia
CNRS

Bjorn Victor
Uppsala University

March 25, 2006

Abstract

We present an expressiveness study of linearity and persistence of processes. We choose the π-calculus, one of the main representatives of process calculi, as a framework to conduct our study. We consider four fragments of the π-calculus. Each one singles out a natural source of linearity/persistence also present in other frameworks such as Concurrent Constraint Programming (CCP), Linear CCP, and several calculi for security. The study is presented by providing (or proving the non-existence of) encodings among the fragments, a processes-as-formulae interpretation and a reduction from Minsky machines.

The fragments are: (1) The *(polyadic) asynchronous* π-calculus π, (2) *persistent-input* π defined as π with all inputs replicated, (3) *persistent-output* π defined as π with all outputs replicated, and (4) *persistent* π defined as π with all inputs and outputs replicated.

We provide compositional fully-abstract encodings, homomorphic w.r.t the parallel operator, from (1) into (2) and (3) capturing the behaviour of source processes. In contrast, we show that it is impossible to provide such encodings from (1) into (4). Nevertheless we prove that (4) is Turing-powerful. We further show that barbed congruence is undecidable for the zero-adic version of (2), the monadic version of (3) and the bi-adic version of (4). In contrast, we also show that it is decidable for the zero-adic versions of (3) and (4).

Furthermore, we provide a compositional interpretation of the π processes in (4) as First-Order Logic (FOL) formulae. The interpretation translates restriction and input binders into existential and universal quantifiers respectively. We illustrate how the interpretation simulates name extrusion (mobility) in FOL. We use the interpretation to characterize the standard π-calculus notion of barbed observability (reachability) as FOL entailment. We apply this characterization and classic FOL results by Bernays, Schönfinkel and Gödel to identify decidable classes (w.r.t. barbed reachability) of infinite-state processes exhibiting meaningful mobile behaviour.

1 Introduction

Several process calculi such as CCS, CSP, the π-calculus [15] and Linear CCP [8, 22] have an obvious source of *linearity*: Messages (or *senders*) are consumed upon being received. For example, in the π-calculus, the system

$$\overline{x}\langle z\rangle \;\mid\; x(y).P \;\mid\; x(y).Q \tag{1}$$

represents a message with a datum z, tagged with x, that can be *consumed* by either $x(y).P$ or $x(y).Q$. The system can evolve into either (a) $P\{z/y\} \mid x(y).Q$ or (b) $x(y).P \mid Q\{z/y\}$.

Nevertheless, there are other process calculi which follow a different pattern: Messages cannot be consumed; they have *persistent* nature rather than a linear one. One of the most prominent representatives of such calculi is Concurrent Constraint Programming (CCP) [21]. In this framework all messages, more precisely items of information, are accumulated in a global store. The messages in the store can be read but, unlike in Linear CCP, they cannot be consumed, i.e., the store is *monotonic*.

Several other frameworks using a monotonic store can be found in the context of calculi for analyzing and describing security protocols. For instance, Crazzolara and Winskel's SPL [7], the Spi Calculus variants by Fiore and Abadi [9] and by Amadio et al [1], and the calculus of Boreale and Buscemi [4] are all operationally defined in terms of configurations containing items of information (messages) which cannot be consumed during evolution. The idea is that the monotonic store models an attacker's ability to see and remember every message that has been in transit.

A legitimate question is whether such monotonicity, or persistence, restricts the systems that we can specify, model or reason about in the framework. For instance, whether CCP can specify the kind of systems that can be described in Linear CCP. Analogously, in the context of the above-mentioned calculi for security, e.g. in SPL, one may wonder if not allowing the attacker to remove messages from the network may rule out the specification of a possible attack to a given protocol.

In [2,8] it is claimed that due the above-mentioned monotonicity, CCP is less expressive than Linear CCP. The claim is based on the discrimination introduced by certain kind of divergence that can arguably be ignored. As a matter of fact it is ignored by the standard notion of weak bisimulation.

There is another source of linearity in (1): *Receivers* can also be consumed. For example, in the case in which $x(y).P$ evolves into $P\{z/x\}$. Persistent receivers arise, e.g. in the notion of *omega receptiveness* [19] where the input of a name is always available—but always with the same continuation. In the π-calculus persistent receivers are used, for instance, to model functions, objects, higher-order communications, or procedure definitions. Notice that the situation in this case is somehow dual to the persistent outputs case and begs the same kind of question: If we require inputs to be persistent, do we restrict the kind of systems we can specify?

Now, in the above situations we have that either messages or receivers are persistent. One can further consider the complementary case in which both messages and receivers are persistent. In the context of CCP, such a restriction would correspond to CCP with universally-quantified persistent ask operations. In the context of calculi for security, persistent receivers can be used to specify protocols where principals are willing to run an unbounded number of times (and persistent messages to model the fact that every message can be remembered by the spy). In fact, the approach of specifying protocols in a persistent setting, with an unbounded number of sessions, has been explored in [3] by using a classic logic Horn clause representation of protocols (rather than a linear logic one).

In this paper, we present our expressiveness study of linearity and persistence in a well-established framework, namely the *asynchronous π-calculus*. This way our study (and its applications) benefits from standard and well-investigated reasoning techniques and notions of equivalence. Furthermore the linear/persistent features of the above calculi are naturally captured in this framework. Linear messages are represented as asynchronous *outputs*, and linear receivers as *input* processes. Persistent messages (and receivers) can simply be specified using the *replication* operator of the calculus which creates an unbounded number of copies of a given process.

We consider four sub-languages of the asynchronous polyadic π-calculus, each capturing one of the above sources of linearity/persistence. Namely, the polyadic asynchronous π-calculus (π), the *persistent-input* π (PIπ) defined as π but inputs must be replicated, *persistent-output* defined dually, i.e. outputs rather than inputs must be replicated (POπ), and finally *persistent* π defined as π but with all inputs and outputs replicated (Pπ). We conduct our study by providing (or proving the non-existence of) encodings among the fragments, a processes-as-formulae interpretation and a reduction from Minsky machines.

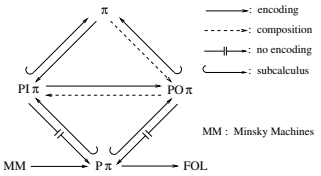

Figure 1: The hierarchy of linearity vs persistence.

Contributions. We provide encodings, homomorphic w.r.t. parallel composition, from π into PIπ and POπ capturing the behaviour of the source processes. These encodings are, respectively, fully abstract w.r.t. weak barbed congruence and weak barbed congruence restricted to encoded contexts. In contrast, we show that it is impossible to provide such encodings from π into Pπ. Intuitively this means that we need one source of linearity, i.e. either on inputs (PIπ) or outputs (POπ) to capture the behaviour of arbitrary π processes via full-abstraction. Despite the impossibility result for Pπ we also prove that Pπ is in

	Pπ	POπ	PIπ
0	yes	yes	no
1	?	no	no
2	no	no	no

Table 1: Decidability of barbed congruence for the n-adic ($n = 0, 1, 2$) persistent calculi

fact Turing-powerful by encoding Minsky machines. Figure 1 illustrates these expressiveness results (a dashed arrow means that the encoding is obtained via composition).

Furthermore, we consider sub-languages of the above π-calculi with restricted arity (i.e., the maximum number of names that can be sent in a single communication) and classify them according to the decidability of barbed congruence and their arity. Interestingly, we state that barbed congruence is undecidable for the zero-adic version of PIπ, the monadic version of POπ and the bi-adic version of Pπ. We also show that barbed congruence is decidable for the zero-adic versions of Pπ and PIπ. We leave open the corresponding decidability question for the monadic version of Pπ. Table 1 summarizes these decidability results.

We also show that Pπ admits a processes-as-formulae compositional interpretation, building on the translation of π to linear logic in [13, 22] and the logical characterization of CCP languages [11, 12]. Specifically, we characterize the standard π-calculus notion of barbed observability (for Pπ) as entailment in First-Order Logic (FOL). Indeed, Pπ can be seen as a CCP language over the Gentzen constraint system (without function symbols), with persistent universal asks [21]. Furthermore, we exploit classic FOL results by Bernays, Schönfinkel and Gödel to identify classes of *infinite-state* processes with meaningful mobile behaviour for which barbed reachability is decidable.

Our expressiveness results bear witness to the generality of the monotonic store assumption in CCP and calculi for security. Moreover, the processes-as-formulae interpretation of POπ has interesting applications. In particular, the decidability results for barbed reachability for POπ may be beneficial for analyzing protocols in which principals (represented as replicated input processes) are willing to run unboundedly many times.

For the sake of readability most proofs and some discussions have been moved to the Appendix.

2 The Calculi

Here we define the calculi we study. We first recall the (polyadic) *asynchronous π-calculus* here referred to as π. The other calculi are defined as syntactic restrictions of π.

$$\text{COM:}\quad \overline{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \longrightarrow P\{\vec{z}/\vec{y}\} \quad \text{if } |\vec{z}| = |\vec{y}|$$

$$\text{PAR:}\ \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \text{RES:}\ \frac{P \longrightarrow P'}{(\nu x)P \longrightarrow (\nu x)P'}$$

$$\text{STRUCT:}\ \frac{P \equiv P' \longrightarrow Q' \equiv Q}{P \longrightarrow Q}$$

Table 2: Reduction Rules.

2.1 Asynchronous Pi Calculus: π

We presuppose a countable set of *names*, ranged over by $x, y, \ldots$, and for each name x, a *co-name* $\overline{x}$. We use $l, l', \ldots$ to range over names and co-names. We use $\vec{x}$ to denote a finite sequence of names $x_1 x_2 \ldots x_n$ of size $|\vec{x}| = n$. The π *processes* are given by the following syntax:

$$P, Q, \ldots := 0 \mid \overline{x}\langle\vec{z}\rangle \mid x(\vec{y}).P \mid (\nu x)P \mid P \mid Q \mid !P$$

requiring that no name may occur more than once in $\vec{y}$.

Intuitively, an *output* $\overline{x}\langle\vec{z}\rangle$ represents a particle tagged with a name x indicating that can be received by an *input process* $x(\vec{y}).P$ which behaves, upon receiving $\vec{z}$, as $P\{\vec{z}/\vec{y}\}$. Furthermore, $x(\vec{y}).P$ binds the names $\vec{y}$ in P. The other binder is the *restriction* $(\nu x)P$ which declares a name x private to P. The *parallel composition* $P \mid Q$ means P and Q running in parallel. The *replication*; $!P$ means $P \mid P \mid \ldots$, i.e., $!P$ represents a *persistent resource*.

We use the standard notations $bn(Q)$ for the *bound names* in Q, and $fn(Q)$ for the *free names* in Q, and write $(\nu x_1 \ldots x_n)P$ to denote $(\nu x_1) \ldots (\nu x_n)P$. We let σ range over non-capturing substitutions of names on processes.

The *reduction* $\longrightarrow$ is the least binary relation on processes satisfying the rules in Table 2. We use $\longrightarrow^*$ to denote the reflexive, transitive closure of $\longrightarrow$. The reductions are quotiented by the *structural congruence* relation $\equiv$.

Definition 2.1. *Let $\equiv$ be the smallest congruence over processes satisfying α-equivalence, the commutative monoid laws for composition with 0 as identity, the replication law $!P \equiv P \mid !P$, the restriction laws $(\nu x)0 \equiv 0$, $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$ and the extrusion law: $(\nu x)(P \mid Q) \equiv P \mid (\nu x)Q$ if $x \notin fn(P)$.*

We conclude the description of π by recalling some process equivalences we shall use throughout the paper. First we recall a basic notion of observation in the π-calculus.

Definition 2.2 (Barbs)**.** *Define $P \downarrow_{\overline{x}}$ iff $\exists \vec{z}, \vec{y}, R : P \equiv (\nu\vec{z})(\overline{x}\langle\vec{y}\rangle \mid R)$ and x is not in $\vec{z}$. Similarly, $P \downarrow_x$ iff $\exists \vec{z}, \vec{y}, Q, R : P \equiv (\nu\vec{z})(x(\vec{y}).Q \mid R)$ and x is not $\vec{z}$. Furthermore, $P \Downarrow_l$ iff $\exists Q : P \longrightarrow^* Q \downarrow_l$.*

Intuitively, given $l = x$ ($l = \overline{x}$) we say that l, a *barb*, can be *observed* at P, written $P \downarrow_l$, iff P can perform an input (output) on channel x.

It should be noticed that in the context of the *asynchronous* π-calculus, as pointed out in [20], in defining the process equivalences it is appropriate to restrict the observables to be the output barbs; i.e., barbs of the form $\overline{x}$. In fact for the congruences below if were to consider inputs barbs as well, the same relation would result.

We begin with a basic π process equivalence, sometimes referred to as *barbed correspondence* [17], which equates processes iff they exhibit the same barbs. In what follows we prefer to refer to this equivalence as *output equivalence* since we only consider output barbs. Recall that a process *context* C is an expression with a hole $[\cdot]$ such that placing a process in the hole produces a well-formed process term.

Definition 2.3 (Output Equivalence, Output Congruence)**.** *We say that P and Q are* output equivalent*, written $P \stackrel{\circ}{\simeq} Q$ iff for every x, $P \Downarrow_{\overline{x}} \Leftrightarrow \ Q \Downarrow_{\overline{x}}$. We say that P and Q are* output congruent*, written $P \simeq Q$, iff for every process context C, $C[P] \stackrel{\circ}{\simeq} C[Q]$.*

We now recall the notion of (weak) barbed bisimilarity.

Definition 2.4 (Barbed Bisimilarity, Barbed Congruence)**.** *A* (weak) barbed bisimulation *is a symmetric relation $\mathcal{R}$ satisfying the following: $(P, Q) \in \mathcal{R}$ implies that:*

1. *$P \longrightarrow P'$ then $\exists Q' : Q \longrightarrow^* Q' \wedge (P', Q') \in \mathcal{R}$.*
2. *$P \downarrow_{\overline{x}}$ then $Q \Downarrow_{\overline{x}}$.*

We say that P and Q are (weak) barbed bisimilar*, written $P \dot{\approx} Q$, iff $(P, Q) \in \mathcal{R}$ for some bisimulation $\mathcal{R}$. Furthermore,* (weak) barbed congruence $\approx$ *is defined as: $P \approx Q$ iff for every process context $C[\cdot]$, $C[P] \dot{\approx} C[Q]$.*

2.2 (Semi) Persistent Subcalculi of π

The Persistent-Output Calculus: POπ. The *persistent-output* calculus POπ arises as from π by requiring all outputs to be replicated. In fact for POπ processes, $\longrightarrow$ can equivalently be defined as in Table 2 with COM replaced with the rule below. The new rule reflects the *linear-input* and *persistent-output* nature of POπ.

$$!\overline{x}\langle\vec{z}\rangle \ \mid \ x(\vec{y}).P \ \longrightarrow \ !\overline{x}\langle\vec{z}\rangle \ \mid \ P\{\vec{z}/\vec{y}\} \quad \text{if } |\vec{z}| = |\vec{y}|$$

The Persistent-Input Calculus: PIπ. The PIπ calculus results from π by requiring all input processes to be replicated. The relation $\longrightarrow$ for PIπ can be equivalently defined as in Table 2 with COM replaced with the rule

$$\overline{x}\langle\vec{z}\rangle \ \mid \ !x(\vec{y}).P \ \longrightarrow \ P\{\vec{z}/\vec{y}\} \ \mid \ !x(\vec{y}).P \quad \text{if } |\vec{z}| = |\vec{y}|$$

The Persistent Calculus: Pπ. Finally, we have the *persistent* calculus Pπ where both output and input processes are required to be replicated. The relation $\longrightarrow$ for Pπ can be equivalently defined as in Table 2 with COM replaced with the rule

$$!\overline{x}\langle\vec{z}\rangle \mid !x(\vec{y}).P \longrightarrow P\{\vec{z}/\vec{y}\} \mid !\overline{x}\langle\vec{z}\rangle \mid !x(\vec{y}).P \text{ if } |\vec{z}| = |\vec{y}|$$

The next proposition reflects the duality of PIπ and POπ.

Proposition 2.5 (Monotonicity)**.** *1. If P is a* POπ *process, $P \longrightarrow Q$ and $P \downarrow_{\overline{x}}$ then $Q \downarrow_{\overline{x}}$.*

2. If P is a PIπ *process $P \longrightarrow Q$ and $P \downarrow_x$ then $Q \downarrow_x$.*

3. If P is a Pπ *process, $P \longrightarrow Q$ and $P \downarrow_l$ then $Q \downarrow_l$.*

2.3 Calculi Conventions and their Equivalences

We will work with fragments of the various calculi in terms of arity.

Definition 2.6 (Arity)**.** *Define the* arity *of R, $\mathcal{A}(R)$, as $\mathcal{A}(\overline{x}\langle\vec{y}\rangle) = |\vec{y}|$, $\mathcal{A}(x(\vec{y}).P) = max(|\vec{y}|, \mathcal{A}(P))$, $\mathcal{A}(P \mid Q) = max(\mathcal{A}(P), \mathcal{A}(Q))$, $\mathcal{A}((\nu x)P) = \mathcal{A}(!P) = \mathcal{A}(P)$.*

Given $\Sigma \in \{\pi, \text{PO}\pi, \text{PI}\pi, \text{P}\pi\}$, the k-adic (version of) Σ, Σ^k is defined as Σ except that its processes have arity less or equal to k. Define $\Sigma^\omega = \Sigma$.

Convention 2.7. *Henceforth* **Calc** *denote the set of calculi defined so far, i.e.,* $\{\pi^k, \text{P}\pi^k, \text{PI}\pi^k, \text{PO}\pi^k \mid k \in \mathbb{N} \cup \{\omega\}\}$.

Definition 2.8 (Parameterized Equivalences)**.** *Let $\Sigma \in$* **Calc**. *We write $P \mathrel{\mathring{\simeq}}^\Sigma Q$ ($P \mathrel{\dot{\approx}}^\Sigma Q$) iff P and Q are Σ processes and $P \mathrel{\mathring{\simeq}} Q$ ($P \mathrel{\dot{\approx}} Q$). Similarly, we write $P \simeq^\Sigma Q$ ($P \approx^\Sigma Q$) iff P and Q are Σ processes and for every Σ context C, $C[P] \mathrel{\mathring{\simeq}}^\Sigma C[Q]$ ($C[P] \mathrel{\dot{\approx}}^\Sigma C[Q]$).*

When no confusion arises we omit the indices from process relations.

Remark 2.9. Reduction, modulo $\equiv$, is invariant wrt the processes of a given calculus. We write $P \longrightarrow_\Sigma Q$ to mean that $P \longrightarrow Q$ and P and Q are Σ processes.

3 Encodings and their properties

In the following sections we provide, or under reasonable conditions demonstrate the impossibility of the existence of, encodings $[\![\cdot]\!]$ from the terms of a given language into the terms of another.

The following condition is particularly appropriate in the context of distributed systems. It describes encodings preserving the parallel topology of the source system.

Definition 3.1. *An encoding $[\![\cdot]\!]$ is a* homomorphism w.r.t. parallel composition *iff $[\![P \mid Q]\!] = [\![P]\!] \mid [\![Q]\!]$. Homomorphism w.r.t the other operators is defined analogously.*

The following notions describe some criteria used in the literature for the correctness of encodings (see e.g., [17]).

Definition 3.2 (Correctness Criteria)**.** *Let* $\Sigma, \Sigma' \in$ **Calc**, *and* $\bowtie \in \{\dot{\approx}, \approx, \overset{\circ}{\simeq}, \simeq\}$. *Let* $[\![\cdot]\!] : \Sigma \to \Sigma'$ *be an* encoding *(i.e., a map) of* Σ *terms into* Σ' *terms. The encoding is* sound *wrt* $\bowtie$ *iff* $[\![P]\!] \bowtie^{\Sigma'} [\![Q]\!]$ *implies* $P \bowtie^{\Sigma} Q$. *The encoding is* complete *wrt* $\bowtie$ *iff* $P \bowtie^{\Sigma} Q$ *implies* $[\![P]\!] \bowtie^{\Sigma'} [\![Q]\!]$. *The encoding is* fully abstract *wrt* $\bowtie$ *iff it is both sound and complete wrt* $\bowtie$. *Finally,* $[\![\cdot]\!]$ *is* ideal *wrt* $\bowtie$ *iff* $[\![P]\!] \bowtie P$.

Intuitively, given a chosen equivalence, full abstraction says that the encoding reflects (soundness) and preserves (completeness) equivalence of source terms. Full abstraction is a useful criterion for the correctness of an encoding wrt a given equivalence when ideal encodings may not exist. Notice that the criterion of being ideal is stronger than being fully abstract.

4 On the Expressiveness of Pπ

In this section we study the expressiveness of the persistent calculus Pπ. We first prove that it is *impossible* to provide a *sound* encoding, homomorphic wrt parallel composition, from π into Pπ. This holds for all the equivalences under consideration in this paper—see Definition 3.2.

Despite the above impossibility result, we prove that Pπ is Turing powerful. We also show Pπ processes can compositionally be encoded as FOL formulae. We illustrate how *mobility* in Pπ can be naturally simulated in FOL and state the characterization of barbed reachability as FOL entailment. We use the characterization and classic FOL theorems to prove decidability results for meaningful classes of infinite-state mobile Pπ processes.

4.1 Impossibility of encoding π into Pπ

Key to our impossibility result is establishing the property $P \mid P \approx P$ for arbitrary Pπ processes P. Consider $P = (\nu z)!\overline{x}\langle z\rangle$. P may be viewed as a generator of a single private name broadcast on x while $P \mid P$ may be viewed as a generator of two different privates names broadcast on x. Therefore, it may not be immediate than $P \mid P$ should be barbed congruent to P in Pπ. In fact, the property would not hold if we had the mismatch operator $[x \neq y]Q$ whose intended meaning is that Q will be executed iff x and y are different names [20], as the following example illustrates:

Example 4.1. Take $R = !x(y).!x(y').[y \neq y']!\overline{t}$ and $Q = (\nu z)!\overline{x}\langle z\rangle$. One can verify that $(R \mid Q) \not\Downarrow_{\overline{t}}$ but $(R \mid Q \mid Q) \Downarrow_{\overline{t}}$.

The following monotonicity property from [20], which also does not hold in the presence of mismatch, will be very useful for our results:

Proposition 4.2. *For any name substitution* σ, $P \longrightarrow Q$ *implies* $P\sigma \longrightarrow Q\sigma$.

Now, the first interesting result for Pπ is that output congruence and barbed congruence (restricted to Pπ contexts) coincide. The result is a corollary of the following lemma whose proof basically rests on showing that Pπ is confluent.

Lemma 4.3. $\mathrel{\dot{\simeq}}^{\mathrm{P}\pi} = \mathrel{\dot{\approx}}^{\mathrm{P}\pi}$.

Corollary 4.4. $\simeq^{\mathrm{P}\pi} = \approx^{\mathrm{P}\pi}$.

We now proceed to prove the Duplication lemma below. First we need the following Context Lemma.

Lemma 4.5 (Context Lemma)**.** *$P \approx^{\mathrm{P}\pi} Q$ if for every* Pπ *process T and name substitution σ,*

$$T \mid P\sigma \mathrel{\dot{\approx}}^{\mathrm{P}\pi} T \mid Q\sigma$$

Proof. Similar to Lemma 2.1.19 in [20]. □

Lemma 4.6 (Duplication Lemma)**.** *For every* Pπ *processes $P \mid P \approx^{\mathrm{P}\pi} P$.*

Proof. From Lemma 4.3 we can freely replace $\approx$ with $\simeq$. The proof proceeds by induction on the size of P. The proof of $P = (\nu x)R$ is particularly interesting and it uses Proposition 4.2 and Lemmas 4.3 and 4.5. The other cases are easier—see Appendix for details. □

The following proposition can be proven from Duplication Lemma above, Lemma 4.3, and analysis in the reduction of $!P$ in arbitrary contexts with the help of the Context Lemma above.

Proposition 4.7. *For every* Pπ *process P, we have $!P \approx^{\mathrm{P}\pi} P$.*

We now have all what we need to prove our impossibility result:

Theorem 4.8 (Impossibility of Sound Encodings)**.** *Let $\bowtie \in \{\mathrel{\dot{\approx}}, \approx, \mathrel{\dot{\simeq}}, \simeq\}$. There is no encoding $[\![\cdot]\!] : \pi \to \mathrm{P}\pi$, homomorphic wrt parallel composition, such that for all $P, Q \in \pi$, $[\![P]\!] \bowtie^{\mathrm{P}\pi} [\![Q]\!]$ implies $P \bowtie^{\pi} Q$.*

Proof. Notice $\mathrel{\dot{\simeq}}^{\pi}$ contains all the other process equivalences of the form $\bowtie^{\pi}$ while $\approx^{\mathrm{P}\pi}$ is contained in every process equivalence of the from $\bowtie^{\mathrm{P}\pi}$. Then, it suffices to show that there are P, Q such that $[\![P]\!] \approx^{\mathrm{P}\pi} [\![Q]\!]$ but $P \not\mathrel{\dot{\simeq}}^{\pi} Q$ with $[\![\cdot]\!]$ being homomorphic wrt parallel composition. Take $P = R \mid R$ and $Q = R$ where $R = \overline{x} \mid x.x.\overline{t}$. Clearly, $P \not\mathrel{\dot{\simeq}}^{\pi} Q$ since $P \Downarrow_{\overline{t}}$ but $Q \not\Downarrow_{\overline{t}}$. From Lemma 4.6 and homomorphism wrt parallel composition we obtain $[\![P]\!] = [\![R]\!] \mid [\![R]\!] \approx^{\mathrm{P}\pi} [\![R]\!] = [\![Q]\!]$ as wanted. □

4.2 FOL Characterization of Pπ

In this section we give a characterization of Pπ in first-order logic by providing a compositional translation of Pπ processes into logical formulae, following the translation of π into linear logic [22, Table 2], and the well-known embedding

of intuitionistic logic in linear logic through the "of course" modality "!". In particular we shall identify barbed reachability in Pπ as logical consequence.

We assume the reader is familiar with basic notations and concepts of first-order logic. We presuppose a first-order language $\mathcal{L}$ whose non-logical symbols include predicates of the form $\mathtt{out}^k$ where $k \geq 0$ denotes the arity of the predicate. Given two FOL formulae F and G over $\mathcal{L}$, we write $F \models G$ iff the implication formula $(F \Rightarrow G)$ is *logically valid*. If $F \models G$ we say that G is a *logical consequence* of F.

The following proposition simplifies the kind of Pπ processes we need to consider in the translation.

Definition 4.9. *A* Pπ *process is said to be* minimal *iff it can be generated by the following syntax:*

$$P, Q, \ldots := 0 \mid !\overline{x}\langle\vec{y}\rangle \mid !x(\vec{y}).P \mid (\nu x)P \mid P \mid Q.$$

Hence minimal processes are those Pπ processes where replication only appears immediately before input and output processes. For example, $!(\nu x)P$ is not minimal.

Definition 4.10. *Let* $m(\cdot) : \mathrm{P}\pi \to \mathrm{P}\pi$ *be the map into minimal processes given as* $m(!0) = m(0) = 0$, $m(!\overline{x}\langle\vec{z}\rangle) = !\overline{x}\langle\vec{z}\rangle$, $m(!x(\vec{y}).Q) = !x(\vec{y}).m(Q)$, $m(!(P \mid Q)) = m(P \mid Q) = m(P) \mid m(Q)$ *and* $m(!(\nu x)P) = m((\nu x)P) = (\nu x)m(P)$.

Proposition 4.11. *For every* $P \in \mathrm{P}\pi$, $m(P) \approx^{\mathrm{P}\pi} P$.

Proof. From Proposition 4.7. □

Therefore, we can freely restrict ourselves to minimal processes. The following encoding compositionally translates minimal processes into formulae.

Definition 4.12. *Let* $[\![\cdot]\!] : \mathrm{P}\pi \rightharpoonup$ FOL *be the partial map from minimal* Pπ *terms into* FOL *formulae given by:*

$$\begin{aligned}
[\![0]\!] &= \mathtt{true} \\
[\![!\overline{x}\langle\vec{z}\rangle]\!] &= \mathtt{out}^k(x, \vec{z}) \quad \textit{with } k = |\vec{z}| + 1 \\
[\![!x(\vec{y}).P]\!] &= \forall\vec{y}(\mathtt{out}^k(x, \vec{y}) \Rightarrow [\![P]\!]) \textit{ with } k = |\vec{y}| + 1 \\
[\![P \mid Q]\!] &= [\![P]\!] \wedge [\![Q]\!] \\
[\![(\nu x)P]\!] &= \exists x [\![P]\!]
\end{aligned}$$

Intuitively, the above encoding $[\![\cdot]\!]$ is meant to capture in logic terms how computation proceeds in Pπ. In particular it has the following property: P will perform an output iff that output is a logical consequence of $[\![P]\!]$. Notice that existential quantification corresponds to restriction, which can simulate *name extrusion* as illustrated below. Also notice that in the translation the two binders of Pπ, input and restriction, are translated into universal and existential quantifiers (resp), hence reflecting an elegant duality.

Example 4.13 (Name Extrusion in FOL)**.** The process $P = (\nu z)(!\overline{x}\langle z\rangle \mid !z(u).\overline{u})$ *creates a name* z, broadcasts it to the outside on x, and waits on it for a message u from the outside. So, $R = Q \mid P$, with $Q = x(y).\overline{y}\langle t\rangle$, can perform the output $\overline{t}$, i.e., $R \Downarrow_{\overline{t}}$. Consider the FOL translation $[\![R]\!]$ in Definition 4.12:

$$\forall y(\mathtt{out}(x,y) \Rightarrow \mathtt{out}(y,t))$$
$$\wedge\, \exists z(\mathtt{out}(x,z) \wedge \forall u\, \mathtt{out}(z,u) \Rightarrow \mathtt{out}(u)).$$

which is logically equivalent to:

$$\begin{array}{l} \exists z \forall y(\mathtt{out}(x,y) \Rightarrow \mathtt{out}(y,t)) \\ \wedge(\mathtt{out}(x,z) \wedge \forall u\, \mathtt{out}(z,u) \Rightarrow \mathtt{out}(u)). \end{array} \tag{2}$$

Since $\mathtt{out}(z,t)$ is a logical consequence of $\forall y(\mathtt{out}(x,y) \Rightarrow \mathtt{out}(y,t)) \wedge \mathtt{out}(x,z)$, from (2) we obtain $\exists z\, \mathtt{out}(z,t) \wedge \forall u\, \mathtt{out}(z,u) \Rightarrow \mathtt{out}(u)$ from which we obtain $\mathtt{out}(t)$ as logical consequence.

Roughly speaking, the logical step to (2) corresponds to using the Structural Equivalence to move a restriction to outermost position (Definition 2.1). The other steps involve Modus Ponens (plus some deduction rules for quantifiers) which corresponds to applying rule COM. □

The following theorem states the characterization of barbed observability in terms of logical consequence. It is related to a similar characterization in [22, Theorem 2.6]. Recall that $\mathcal{A}(P)$ denotes the arity of P (see Definition 2.6) and that from Proposition 4.11, up-to barbed congruence for Pπ, we can confine our attention to minimal processes.

Theorem 4.14 (FOL Characterization of Barbs)**.** *Let* $[\![\cdot]\!] : \mathrm{P}\pi \rightharpoonup \mathrm{FOL}$ *be the map in Definition 4.12. Let* P *be a minimal* Pπ *process. Then*

$$P \Downarrow_{\overline{x}} \quad \textit{if and only if} \quad [\![P]\!] \models \exists \vec{z}\, \mathtt{out}^{k+1}(x,\vec{z})$$

for some $k \leq \mathcal{A}(P)$.

Proof. The proof of the "if" direction, the most difficult case, uses a normal form representation of the target formulae. Such normal forms simplifies the analysis of how the formulae on the right-hand of $\models$ could have been deduced from $[\![P]\!]$. See Appendix. □

Remark 4.15. Intuitively, the correspondence in Theorem 4.14 holds because we do not have operators than can make use of the fact that two names are different. It would not hold if we had mismatch with the natural translation $[\![[x \neq y].P]\!] = (x \neq y) \Rightarrow [\![P]\!]$. E.g., $(\nu x, y)[x \neq y].\bar{t}$ can perform $\bar{t}$ while from $\exists x, y(x \neq y) \Rightarrow \mathtt{out}(t)$ we cannot conclude $\mathtt{out}(t)$. For more on this issue see Remark A.1 in the appendix.

Applications of the FOL Characterization. The following results are meant to illustrate applications of the above FOL characterization by using classic results from FOL to prove decidability results for barb reachability and barbed congruence.

Decidable Pπ Classes. The following lemma identifies several classes of Pπ processes whose barb-reachability problem is decidable. These classes include classes of *infinite-state* processes with name mobility (extrusion). The reachability question is relevant for safety properties stating that a given undesired output will never be performed.

Lemma 4.16 (Decidability of Barb Reachability)**.** *Let* $[\![\cdot]\!] : \mathrm{P}\pi \rightharpoonup \mathrm{FOL}$ *be the map in Definition 4.12. Given* $P \in \mathrm{P}\pi$ *and a name* z*, if* P *belongs to one of the following classes*

1. $\{R \mid [\![R]\!] \Leftrightarrow \forall\vec{x}\exists\vec{y}F\}$ *(Bernays-Schönfinkel's class).*
2. $\{R \mid [\![R]\!] \Leftrightarrow \forall\vec{x}\exists uw\forall\vec{y}F\}$ *(Gödel's class).*
3. $\{R \mid R \equiv R'$ *for some* R' *s.t.* $|fn(R')\cup bn(R')| \leq 2\}$ *(Two-Variables Class).*
4. $\mathrm{P}\pi^0$ *(Persistent CCS-like Class).*

where F *is a quantifier-free formula, then the question of whether* $P \Downarrow_{\overline{z}}$ *is decidable.*

Proof. (Outline) The proof proceeds by reducing the question, with the help of Proposition 4.11, Theorem 4.14, to the validity of a formula which is in the class of either Bernays-Schönfinkel, Gödel, two-variables, or Monadic FOL formulae without function symbols. All these classes of formulae are decidable [5]. □

Decidable Classes with Mobility. Let us illustrate briefly the name extrusion capabilities of the "mobile" classes in the above lemma. It is important to recall that input and restriction binders are translated into universal and existential quantifiers, respectively. This means that alternation of quantifiers corresponds to alternation of inputs and restrictions.

Consequently, Bernays-Schönfinkel's class allows *providers of new names*, i.e, processes that upon request, say on a channel x, can output private names in a given return channel. For example, $Q = !x(y).(\nu z)!\overline{y}\langle z\rangle$.

It is worth pointing out that this class is closed under parallel composition. So Q composed with a process R in the class, remains in the class: e.g. R could be a process $!\overline{x}\langle r\rangle \mid !r(z).Q'$ requesting from Q a fresh name—notice that R is in the class if Q' is also in the class.

Nevertheless, in general the Bernays-Schönfinkel's class does not allow processes with inputs on private names as the $P = (\nu z)(!\overline{x}\langle z\rangle \mid !z(u).!\overline{u})$ in our name-extrusion example (Example 4.13). However, the Gödel class allows such processes only if the number of such inputs on private names is less than three.

The third class allows processes which can be rewritten (re-using bound names wherever possible) with only two names. For example P and Q above belong to the class since $P \equiv (\nu z)(!\overline{x}\langle z\rangle \mid !z(x).!\overline{x})$ and $Q \equiv !x(u).(\nu x)!\overline{u}\langle x\rangle$.

Decidability Result for Barbed Congruence. It is easy to adapt the results [6] to prove that (weak) barbed congruence is *undecidable* for the zero-adic version of the π, in our notation π^0. In contrast, here we prove that (weak)

barbed congruence is decidable for the zero-adic version of Pπ, Pπ^0. The proof, which can be found in the Appendix, uses the FOL reasoning.

The following theorem states the decidability of all the equivalence under consideration for Pπ^0.

Theorem 4.17 (Decidable Equivalences of Pπ^0)**.** *Let* $\Sigma = \mathrm{P}\pi^0$*. Given* P, Q *in* Σ *and* $\bowtie \in \{\mathring{\simeq}, \simeq, \dot{\approx}, \approx\}$*, the question whether* $P \bowtie^{\Sigma} Q$ *is decidable.*

Proof. The construction for the decidability of the second question involves the use of FOL reasoning to characterize a finite set of contexts sufficient for verifying $\simeq^{\Sigma}$—see Appendix. □

4.3 Turing Expressiveness of Pπ

In Section 4.1 we proved that there is no sound encoding, homomorphic wrt parallel, from π into Pπ. In this section we show that despite such an impossibility result Pπ is Turing-powerful. We do this by encoding two-counter machines, also called Minsky machines, which are known to be Turing-powerful [16].

Minsky Machines. A *two-counter Minsky machine* is an imperative program consisting of a sequence of labelled instructions $I_1; \ldots; I_k$ which modify the values of two non-negative counters c_0 and c_1. The instructions, using counters c_n for $n \in \{0, 1\}$, are of three kinds: $L_i : \texttt{halt}$, $L_i : c_n := c_n + 1;\ \texttt{goto}\ L_j$, and $L_i : \texttt{if}\ c_n = 0\ \texttt{then goto}\ L_j^1\ \texttt{else}\ c_n := c_n - 1;\ \texttt{goto}\ L_j^2$. The Minsky machine starts at L_s and halts if control reaches the location of a `halt` instruction. A Minsky machine $M(v_0, v_1)$ computes the value n if it halts with $c_0 = n$.

Encoding Minsky Machines. Our encoding of a given Minsky machine M with start location L_s and initial counter values v_0, v_1 into Pπ, $[\![M_s(v_0, v_1)]\!]$, is given below, with the encoding of non-negative numbers in counter c, $(\!|n|\!)_c$. The counter values are encoded in a standard fashion (similar to the persistent lists in [14]), and each location L_i corresponds to a fresh name l_i over which the current counter values are passed. Where ever $\overline{l_j}\langle c, c_{n+1}\rangle$ appears, order the objects correctly based on n (using addition modulo 2).

$$
\begin{array}{lll}
[\![M_s(v_0,v_1)]\!] & = (\nu c_0, c_1)\big((\!|v_0|\!)_{c_0} \mid (\!|v_1|\!)_{c_1} \mid\ !\,\overline{l_s}\langle c_0, c_1\rangle \\
& \qquad \mid \textstyle\prod_{1\le i\le k} [\![I_i]\!]\big) & \text{where } M = I_1; \ldots; I_k \\
[\![L_i : \texttt{halt}]\!] & = !\, l_i(c_0, c_1)\,.\,!\,\overline{halt}\langle c_0\rangle & \\
[\![L_i : c_n := c_n + 1;\ \texttt{goto}\ L_j]\!] & & \\
& = !\, l_i(c_0, c_1)\,.\,(\nu c)(S(c, c_n) \mid !\,\overline{l_j}\langle c, c_{n+1}\rangle) & \\
[\![L_i : \texttt{if}\ c_n = 0\ \texttt{then goto}\ L_j^1\ \texttt{else}\ c_n := c_n - 1;\ \texttt{goto}\ L_j^2]\!] & & \\
& = !\, l_i(c_0, c_1)\,.\,(\nu s, z)\big(!\,\overline{c_n}\langle s, z\rangle \mid & \\
& \qquad !\, z\,.\,!\,\overline{l_j^1}\langle c_0, c_1\rangle \mid !\, s(c)\,.\,!\,\overline{l_j^2}\langle c, c_{n+1}\rangle\big) & \\
(\!|0|\!)_c & = Z(c) & \\
(\!|n|\!)_c & = (\nu p)((\!|n-1|\!)_p \mid S(c, p)) & \text{for } n > 0 \\
Z(c) & = !\, c(s, z)\,.\,!\,\overline{z} & \text{(zero)} \\
S(c, p) & = !\, c(s, z)\,.\,!\,\overline{s}\langle p\rangle & \text{(successor)}
\end{array}
$$

In the encoding, because of the persistent nature of Pπ, all states which have been triggered can always be "re-executed". The encoding of (persistent)

counter values uses private channels for signalling successor and zero values, and incremented values are created at private locations. Thus the operations on the counters in one state have no effect on the values encoded in another state – the encoding is free of side-effects.

For example, consider the encoding the if-then-else instruction. The counter values at the previous enabled location are received over l_i; the counter c_n is asked for its value and will respond on *one* of the fresh names s and z. If it responds on z, location l_j^1 is triggered with the current counter values; if it responds on s, indicating it is a successor value, then location l_j^2 is triggered with the predecessor of c_n (which is received over s) and the other counter value.

Theorem 4.18. *A Minsky machine $M_s(v_0, v_1)$ computes the value n if and only if $[\![M_s(v_0, v_1)]\!] \mid !\, halt(c)\,.\, Dec_n(c) \Downarrow \overline{yes}$*
where
$Dec_0(c) \;=\; (\nu s, z)(!\,\overline{c}\langle s, z\rangle \mid !\, z\,.\,!\,\overline{yes})$
$Dec_i(c) \;=\; (\nu s, z)(!\,\overline{c}\langle s, z\rangle \mid !\, s(p)\,.\, Dec_{i-1}(p))$ *for $i > 0$*

Applications of the Minsky Encoding. In the previous sections we proved that all equivalences and barb reachability problems are decidable for $\mathrm{P}\pi^0$. Here we state, on the contrary, that all these problems are undecidable for $\mathrm{P}\pi$.

As a direct consequence of the encoding of two-counter machines, we get undecidability of barbed reachability:

Lemma 4.19 (Undecidability of $\mathrm{P}\pi$ Barb Reachability)**.** *Given P in $\mathrm{P}\pi$ and a name x, the question whether $P \Downarrow_{\overline{x}}$ is undecidable.*

Proof. From Theorem 4.18 and the undecidability of the Halting problem for Minsky Machines. □

From the above lemma and a series of reductions (see Appendix) we get the following results.

Theorem 4.20 (Undecidable Equivalences of $\mathrm{P}\pi$)**.** *Let $\Sigma = \mathrm{P}\pi$. Given P, Q in Σ and $\bowtie \in \{\mathring{\simeq}, \simeq, \dot{\approx}, \approx\}$, the question whether $P \bowtie^{\Sigma} Q$ is undecidable.*

Remark 4.21. In fact, the above undecidability results (Lemma 4.19 and Theorem 4.20) apply already to $\mathrm{P}\pi^2$—they are all obtained from reductions of the Halting Problem for Minsky Machines which were encoded using only the $\mathrm{P}\pi^2$ fragment of $\mathrm{P}\pi$. We leave open the corresponding decidability questions for $\mathrm{P}\pi^1$—recall that the all the corresponding questions are decidable for $\mathrm{P}\pi^0$ (Theorem 4.17).

5 Expressiveness of Semi-Persistent Calculi

Here we study the expressiveness of the semi-persistent calculi $\mathrm{PI}\pi$ and $\mathrm{PO}\pi$ by means of encodings from π. Let us first give some intuition about what we wish to capture in our encodings.

The Problem: Encoding Linearity. Consider the π system:

$$S = \overline{x}\langle u\rangle \;\mid\; \overline{x}\langle w\rangle \;\mid\; x(y).\overline{y}\langle m\rangle \;\mid\; x(y).\overline{y}\langle n\rangle \tag{3}$$

An encoding from π into a semi-persistent calculus will be a homomorphism that on S takes the form

$$[\![S]\!] = [\![\overline{x}\langle u\rangle]\!] \;\mid\; [\![\overline{x}\langle w\rangle]\!] \;\mid\; [\![x(y).\overline{y}\langle m\rangle]\!] \;\mid\; [\![x(y).\overline{y}\langle n\rangle]\!]$$

Intuitively, to capture the linear communication nature of π, the encoding would evolve into a process that behaves either as (a) $[\![\overline{u}\langle m\rangle]\!] \mid [\![\overline{w}\langle n\rangle]\!]$ or as (b) $[\![\overline{w}\langle m\rangle]\!] \mid [\![\overline{u}\langle n\rangle]\!]$. Notice that in each case an output and input (prefix) are consumed.

The obvious problem is that in the semi-persistent calculi either input or outputs are persistent. Let us first discuss the encoding of π into POπ.

From π into POπ. A convenient approach is to view (the *encoding* of) input processes as agents competing for the data of (the *encoding* of) an output which must become unavailable upon being received by the successful agent.

A naive solution is to have the above-mentioned competing agents send a private channel r on which the output data would be received; e.g., $[\![x(\vec{y}).P]\!] = (\nu r)(!\overline{x}\langle r\rangle \;\mid\; r(\vec{y}).[\![P]\!])$. The encoded outputs must wait for the private channel on which they send their data; e.g., $[\![\overline{x}\langle\vec{z}\rangle]\!] = x(r).!\overline{r}\langle\vec{z}\rangle$. Now, a problem is then that, e.g., the two encoded outputs in (3) may get the private channel of only one of the encoded inputs, thus making it impossible for the other encoded input to get u or w. So one of the encoded outputs will be consumed and the other will be unable to react with other encoded inputs.

The above observation suggests that encoded outputs should also send a secret channel s on which they get a encoded input's secret channel. We could then try

$$\begin{array}{rcl} [\![\overline{x}\langle\vec{z}\rangle]\!] & = & (\nu s)(!\overline{x}\langle s\rangle \;\mid\; s(r).!\overline{r}\langle\vec{z}\rangle) \\ [\![x(\vec{y}).P]\!] & = & x(s).(\nu r)(!\overline{s}\langle r\rangle \;\mid\; r(\vec{y}).[\![P]\!]) \end{array} \tag{4}$$

with $[\![0]\!] = 0$ and $[\![\cdot]\!]$ being homomorphic w.r.t all other operators. This solves the above problem, but it creates the dual one. E.g., one of the two encoded inputs in (3) will successfully get the data but the other will be unable to react with another encoded output. It would then seem that we need a more involved protocol to solve the problem.

From π to PIπ and from PIπ to POπ. The above-mentioned problem of the encoding of the input being unable to react with other encoded outputs would disappear if such an input was replicated; i.e., if a copy becomes unable to react, we can always try another one. Now recall inputs are always replicated in PIπ. So, an encoding of π into POπ may arise from an encoding of π into PIπ composed with the encoding in (4). Let us then give the latter encoding first.

5.1 Encoding π into PIπ: Forwarders

To make a replicated-input behave as a resource that provides a service only once, one may suggest: $[\![x(\vec{y}).P]\!] = (\nu l)(\bar{l} \mid !x(\vec{y}).!l.[\![P]\!])$ and $[\![\overline{x}\langle\vec{z}\rangle]\!] = \overline{x}\langle\vec{z}\rangle$. The idea is that the encoded input has a private "lock" l which is activated only once. So, even if the input is replicated, its continuation can be executed only once. Now, the problem is that the prefix $!x(\vec{y})$ may act as a "sink" consuming several outputs. Nevertheless, a suitable combination of the above "lock" idea with a forwarding mechanism leads us to the following encoding:

Definition 5.1. *The encoding $[\![\cdot]\!] : \pi \to \mathrm{PI}\pi$ is a homomorphism for parallel composition, restriction and replication, otherwise is defined as $[\![0]\!] = 0, [\![\overline{x}\langle\vec{z}\rangle]\!] = \overline{x}\langle\vec{z}\rangle$ and*

$$[\![x(\vec{y}).P]\!] = (\nu\, t\, f)(\bar{t} \mid !x(\vec{y}).(\nu\, l\,)(\bar{l} \mid !t.!l.([\![P]\!] \mid !\overline{f}) \mid !f.!l.\overline{x}\langle\vec{y}\rangle))$$

where $t, f, l \notin fn(P) \cup \{x, \vec{y}\}$.

Intuitively, an encoded input behaves thus: It creates two locks "true" t and "false" f, and then always waits for messages on x. The first time it receives a message $\overline{x}\langle z\rangle$, it activates $\bar{t}$. It then creates a second lock l—we will comment on the need of this lock below. This way only the $!t.!l$-branch is activated and the message is accepted by executing $[\![P]\!]\{\vec{z}/\vec{y}\}$ and activating $\overline{f}$. For every subsequent message $\overline{x}\langle u\rangle$ the input gets, only the $!f.!l$-branch is opened, and hence $\overline{x}\langle\vec{u}\rangle$ is forwarded. Notice that if we did not use the lock l, then a "dangling" f-branch $!f.\overline{x}\langle\vec{z}\rangle$, resulting after having received the first message $\overline{x}\langle\vec{z}\rangle$, could be opened by an $\overline{f}$. This would cause $\overline{x}\langle z\rangle$ to be forwarded but this message must be consumed.

Properties of $[\![\cdot]\!] : \pi \to \mathrm{PI}\pi$. The above encoding is in fact ideal w.r.t barbed congruence. The proof is standard and it uses a fundamental property of asynchronous π: Forwarders are barbed congruent to the null process [10]; i.e. $!x(\vec{y}).\overline{x}\langle\vec{y}\rangle \approx 0$.

Proposition 5.2. *Let $[\![\cdot]\!] : \pi \to \mathrm{PI}\pi$ be the encoding in Definition 5.1. For every P, $[\![P]\!] \approx P$ holds.*

From the above proposition, we get full abstraction w.r.t barbed congruence.

Theorem 5.3. *(Full Abstraction) Let $[\![\cdot]\!] : \pi \to \mathrm{PI}\pi$ as in Definition 5.1. For every P, Q: $P \approx Q$ iff $[\![P]\!] \approx^{\mathrm{PI}\pi} [\![Q]\!]$.*

Application. Using the above encoding we can prove that barbed congruence for the zero-adic version of PIπ, PIπ^0, is undecidable. This is to be contrasted with the decidability of Pπ^0 shown in the previous section. The results follows from the full-abstraction theorem above and the undecidability of barbed congruence for π^0 which can be proven as that of weak bisimilarity for π^0 in [6].

Theorem 5.4. *Let $\Sigma = \mathrm{PI}\pi^0$. Given $P, Q \in \Sigma$, the question whether $P \approx^{\Sigma} Q$ is undecidable.*

5.2 Encoding π into POπ via composition

We can now use the above encoding of π into PIπ to get an encoding of π into POπ by composing it with the following encoding from PIπ into POπ.

Definition 5.5. *The encoding $f = [\![\cdot]\!] : \mathrm{PI}\pi \to \mathrm{PO}\pi$ is a homomorphism for parallel composition, restriction, and replication, otherwise is defined as $[\![0]\!] = 0$, and*

$$\begin{aligned} [\![\overline{x}\langle\vec{z}\rangle]\!] &= (\nu s)(!\overline{x}\langle s\rangle \mid s(r).!\overline{r}\langle\vec{z}\rangle) \\ [\![!x(\vec{y}).P]\!] &= !x(s).(\nu r)(!\overline{s}\langle r\rangle \mid r(\vec{y}).[\![P]\!]) \end{aligned}$$

where $s, r \notin fn(P) \cup \{x, z\}$. Let g be $[\![\cdot]\!] : \pi \to \mathrm{PI}\pi$ in Definition 5.1. The encoding $[\![\cdot]\!] : \pi \to \mathrm{PO}\pi$ is the composite function $f \circ g$.

Properties of $[\![\cdot]\!] : \pi \to \mathrm{PO}\pi$. Let us state the main properties of $[\![\cdot]\!] : \pi \to \mathrm{PO}\pi$ given in Definition 5.5. Because of this encoding maps a linear output into a replicated one with the same barb, the encoding does not enjoy the property of being ideal wrt barbed congruence. Notice that replicated inputs were not a problem since the standard barb congruence for π does not observe inputs barbs. However, the following proposition states that the encoding is fully-abstract w.r.t. (weak) barbed bisimilarity.

Proposition 5.6. *For $[\![\cdot]\!] : \pi \to \mathrm{PO}\pi$ in Definition 5.5 we have: $P \dot{\approx} Q$ if and only if $[\![P]\!] \dot{\approx}^{\mathrm{PO}\pi} [\![Q]\!]$*

Nevertheless, due to the compositionally of $[\![\cdot]\!]$, we can give a stronger correspondence result which takes into account weak barbed congruence. Assume that $[\![\cdot]\!] : \pi \to \mathrm{PO}\pi$ in Definition 5.5 is extended to process contexts: We decree that $[\![[\cdot]]\!] = [\cdot]$. Define $[\![P]\!] \approx^{\mathrm{PO}\pi}_{[\cdot]} [\![Q]\!]$ iff for every π context C, $[\![C]\!][[\![P]\!]] \dot{\approx}^{\mathrm{PO}\pi} [\![C]\!][[\![Q]\!]]$.

Theorem 5.7 (Full-Abstraction wrt Encoded Contexts)**.** *Let $[\![\cdot]\!] : \pi \to \mathrm{PO}\pi$ as in Definition 5.5. $P \approx Q$ iff $[\![P]\!] \approx^{\mathrm{PO}\pi}_{[\cdot]} [\![Q]\!]$*

Remark 5.8. Our encoding simulates an atomic communication with a protocols of finer communications, thus one can envisage a malicious context which does not behave according to the protocol. This is the same kind of problem of the standard encoding of polyadic π into the monadic π-calculus [14]. See Remark A.6 in the appendix for a discussion and a counter-example to full-abstraction wrt barbed congruence.

Applications. Obviously, the undecidability of barbed congruence for POπ follows from its subcalculus Pπ. However, we left open the (un)-decidability of Pπ^1. As an application of the above encoding we state the undecidability of barbed congruence for POπ^1.

Theorem 5.9. *Let $\Sigma = \mathrm{PO}\pi^1$. Given $P, Q \in \Sigma$, the question of whether $P \approx^{\Sigma} Q$ is undecidable.*

So all in all we have shown undecidability for barbed congruence for $\mathrm{PI}\pi^0$, $\mathrm{PO}\pi^1$, and $\mathrm{P}\pi^2$. However, in contrast to $\mathrm{PI}\pi^0$ and like $\mathrm{P}\pi^0$, barbed congruence for $\mathrm{PO}\pi^0$ is decidable as shown below.

From $\mathrm{PO}\pi^0$ into $\mathrm{P}\pi^0$. We conclude this section by proving the decidability of barbed congruence for $\mathrm{P}\pi^0$ by encoding it into its subcalculus $\mathrm{P}\pi^0$. This encoding also gives us a FOL characterization for $\mathrm{PO}\pi^0$.

Definition 5.10. *The encoding $[\![\cdot]\!] : \mathrm{PO}\pi^0 \to \mathrm{P}\pi^0$ is a homomorphism for parallel composition, restriction, and replication, otherwise is defined as $[\![0]\!] = 0$, $[\![!\overline{x}]\!] = !\overline{x}$ and $[\![x.P]\!] = !x.[\![P]\!]$.*

Notice that linear inputs are interpreted as replicated ones. The following result states that in the context of $\mathrm{PO}\pi^0$ this interpretation is correct wrt barbed congruence.

Proposition 5.11. *Let $[\![\cdot]\!] : \mathrm{PO}\pi^0 \to \mathrm{P}\pi^0$ be the encoding in Definition 5.10. We have $[\![P]\!] \approx^{\mathrm{PO}\pi^0} P$.*

From the above proposition and the decidability $\approx$ for $\mathrm{P}\pi^0$ (Theorem 4.17) we obtain the following:

Corollary 5.12. *Let $\Sigma = \mathrm{PO}\pi^0$. Given $P, Q \in \Sigma$, the question of whether $P \approx^{\Sigma} Q$ is decidable.*

Furthermore using the above encoding and lemma and the processes-as-formulae FOL interpretation of $\mathrm{P}\pi$ (Definition 4.12) we conclude that $\mathrm{PO}\pi^0$ can be interpreted likewise. E.g., $x.P$ and $(\nu x)P$ are compositionally interpreted as the formulae $\mathtt{out}(x) \Rightarrow [\![P]\!]$ and $\exists x [\![P]\!]$, respectively, where $[\![P]\!]$ is the interpretation of P.

Corollary 5.13. *Let $f : \mathrm{P}\pi \to \mathrm{FOL}$ as in Definition 4.12, $g : \mathrm{PO}\pi^0 \to \mathrm{P}\pi^0$ in Definition 5.10 and $m : \mathrm{P}\pi \to \mathrm{P}\pi$ in Definition 4.10. Let $P \in \mathrm{PO}\pi^0$ and $[\![P]\!] = (f \circ m \circ g)(P)$:*

$$P \Downarrow_{\overline{x}} \quad \textit{if and only if} \quad [\![P]\!] \models \mathtt{out}(x).$$

References

[1] R. Amadio, D. Lugiez, and V. Vanackere. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290, 2003.

[2] E. Best, F. de Boer, and C. Palamidessi. Partial order and sos semantics for linear constraint programs. In *Proc. of Coordination'97*, volume 1282 of *LNCS*, 1997.

[3] B. Blanchet. From linear to classical logic by abstract interpretation. *Information Processing Letters*, 95(5), 2005.

[4] M. Boreale and M. Buscemi. A framework for the analysis of security protocols. In *Proc. CONCUR'02*, volume 2421 of *LNCS*, 2002.

[5] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[6] N. Busi, M. Gabbrielli, and G. Zavattaro. Comparing recursion, replication and iteration in process calculi. In *Proc. ICALP'04*, volume 3142 of *LNCS*, 2004.

[7] F. Crazzolara and G. Winskel. Events in security protocols. In *Proc. CCS 2001*. ACM Press, 2001.

[8] F. Fages, P. Ruet, and S. Soliman. Linear concurrent constraint programming: operational and phase semantics. *Information and Computation*, 2001.

[9] M. Fiore and M. Abadi. Computing symbolic models for verifying cryptographic protocols. In *Proc. CSFW-14*. IEEE, 2001.

[10] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.

[11] P. Lincoln and V. Saraswat. Proofs as concurrent processes: A logical interpretation of concurrent constraint programming. Technical report, Xerox PARC, 1991.

[12] N. P. Mendler, P. Panangaden, P. J. Scott, and R. A. G. Seely. A logical view of concurrent constraint programming. *Nordic J. of Computing*, 2(2):181–220, 1995.

[13] D. Miller. The pi-calculus as a theory in linear-logic. In *Proc. of Workshop on Extensions to Logic Programming*, volume 660 of *LNCS*, 1992.

[14] R. Milner. *Communicating and Mobile Systems: the π-calculus*. Cambridge University Press, 1999.

[15] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

[16] M. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[17] U. Nestmann. What is a 'good' encoding of guarded choice? *Information and Computation*, 156:287–319, 2000.

[18] P. Quaglia and D. Walker. On encoding pπ in mπ. In *Proc. FSTTCS'98*, volume 1530 of *LNCS*, 1998.

[19] D. Sangiorgi. The name discipline of uniform receptiveness. *Theoretical Computer Science*, 221(1–2):457–493, 1999.

[20] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[21] V. Saraswat. *Concurrent Constraint Programming*. The MIT Press, Cambridge, MA, 1993.

[22] V. Saraswat and P. Lincoln. Higher-order linear concurrent constraint programming. Technical report, Xerox PARC, 1992.

[23] N. Yoshida. Graph types for monadic mobile processes. In *Proc. FSTTCS'96*, volume 1180 of *LNCS*, 1996.

A Appendix

Proofs for section 4.1: Impossibility of encoding $\mathrm{P}\pi$ into π

Lemma. *4.3.* $\overset{\circ}{\simeq}^{\mathrm{P}\pi} = \dot{\approx}^{\mathrm{P}\pi}$.

Proof. (Outline) Let $\Sigma = \mathrm{P}\pi$. Obviously, $\approx^{\Sigma} \subseteq \simeq^{\Sigma}$. The other direction is obtained by showing that $\overset{\circ}{\simeq}^{\Sigma}$ is a barbed bisimulation (Definition 2.4).

Take an arbitrary $(P, Q) \in \mathcal{R}$. The second condition of Definition 2.4 follows from $P \overset{\circ}{\simeq} Q$. As for the first condition, suppose $P \longrightarrow_{\mathrm{P}\pi} P'$. We need to find Q' such that $Q \longrightarrow^{*}_{\Sigma} Q'$ and $(P', Q') \in \mathcal{R}$. We shall show that $P' \overset{\circ}{\simeq} P$, so it suffices to take $Q' = Q$.

From Definition 2.2 $P' \Downarrow_{\overline{x}}$ implies $P \Downarrow_{\overline{x}}$. For the other direction, verify that if $P \longrightarrow_{\Sigma} P'$ then there are M and M' such that $P \equiv M$ and $P' \equiv M'$,

$$M = (\nu \vec{z})(\prod_{i \in I} !\overline{x_i}\langle \vec{z_i} \rangle \mid \prod_{j \in J} !x_j(\vec{y_j}).P_j \mid \prod_{k \in K} !P_k)$$

$$M' = (\nu \vec{z}\vec{u})(\prod_{i \in I'} !\overline{x_i}\langle \vec{z_i} \rangle \mid \prod_{j \in J'} !x_j(\vec{y_j}).P_j \mid \prod_{k \in K'} !P_k)$$

with $I \subseteq I', J \subseteq J'$, $K \subseteq K'$, and $fn(M) = fn(M')$. Clearly, $M \Downarrow_{\overline{x}}$ implies $M' \Downarrow_{\overline{x}}$. Since structural congruence is included in $\overset{\circ}{\simeq}$, we conclude that $P \Downarrow_{\overline{x}}$ implies $P' \Downarrow_{\overline{x}}$ as wanted. □

Lemma. *4.6. For every* $\mathrm{P}\pi$ *processes* $P \mid P \approx^{\mathrm{P}\pi} P$.

Proof (Outline). From Lemma 4.3 we can freely replace $\approx$ with $\simeq$. The proof proceeds by induction on the size of P. The most interesting case is $P = (\nu x)R$. The basic case $P = !\overline{x}\langle \vec{z} \rangle$ follows from the fact that for every Q, $!Q \mid !Q \simeq !Q$. The case $P = Q \mid R$ follows by appeal to induction and from the fact that $\simeq^{\mathrm{P}\pi}$ is a congruence. The remaining cases are similar or simpler.

Let P be of the form $(\nu x)R$. From Lemmas 4.5 and 4.3 it suffices to show that

$$(T \mid P\sigma \mid P\sigma) \Downarrow_{\overline{x}} \text{ iff } (T \mid P\sigma) \Downarrow_{\overline{x}}$$

for arbitrary given T and σ.

The "if" direction is trivial. For the "only if" direction pick two fresh names n_1 and n_2. Notice that σ must not involve x, so from the freshness of n_1 and n_2, conclude that

$$(T \mid (R\{n_1/x\}\sigma \mid R\{n_2/x\}\sigma)) \Downarrow_{\overline{x}}.$$

Now pick a fresh name n and a substitution $\sigma_n = \{n/n_1, n/n_2\}$. Using Proposition 4.2 and the freshness of n, n_1, n_2, we must have:

$$(T \mid (R\{n_1/x\}\sigma \mid R\{n_2/x\}\sigma)\sigma_n) \Downarrow_{\overline{x}}.$$

Observe that $T \mid (R\{n_1/x\}\sigma \mid R\{n_2/x\}\sigma)\sigma_n = T \mid (R\{n/x\}\sigma \mid R\{n/x\})\sigma$ since n, n_1 and n_2 are fresh. Since the size of $R\{n/x\}$ is less than the size of P,

by appeal to induction we have $R\{n/x\} \mid R\{n/x\} \simeq^{\mathrm{P}\pi} R\{n/x\}$. Since $\simeq^{\mathrm{P}\pi}$ is a congruence conclude that

$$(T \mid R\{n/x\}\sigma) \Downarrow_{\overline{x}} .$$

But n is fresh, hence

$$(T \mid (\nu x)R\sigma) \Downarrow_{\overline{x}}$$

as wanted. □

Proofs for section 4.2: FOL Characterization of $\mathrm{P}\pi$

Theorem. *4.14. Let $[\![\cdot]\!] : \mathrm{P}\pi \rightharpoonup \mathrm{FOL}$ be the map in Definition 4.12. Let P be a minimal $\mathrm{P}\pi$ process. Then*

$$P \Downarrow_{\overline{x}} \quad \textit{if and only if} \quad [\![P]\!] \models \exists \vec{z}\, \mathtt{out}^{k+1}(x, \vec{z})$$

for some $k \leq \mathcal{A}(P)$.

Proof (Outline). Here we outline the main steps of the proof.

($\Leftarrow$) For this direction one first shows using induction on length of the derivation that every reduction corresponds to a logic deduction: More precisely,

$$\text{if } P \longrightarrow_{\mathrm{P}\pi} Q \text{ then } [\![P]\!] \models [\![Q]\!]. \tag{5}$$

From Definition 2.2 if $P \Downarrow_{\overline{x}}$ then there are $\vec{z}, \vec{y}$ and R, R' s.t., $P \longrightarrow^*_{\mathrm{P}\pi} R = (\nu \vec{z})(\overline{x}\langle \vec{y} \rangle \mid R')$ and x is not in $\vec{z}$. Clearly, $[\![R]\!] \models \exists \vec{y} : \mathtt{out}^{k+1}(x, \vec{y})$ for $k = |\vec{y}|$. The desired result follows easily from Equation 5.

($\Rightarrow$) The proof of this direction makes use of normal forms. A process in $\mathrm{P}\pi$ is a *normal form* if it takes the form

$$(\nu \vec{u})(\prod_{i \in I} !\overline{x_i}\langle \vec{z_i} \rangle \mid \prod_{j \in J} !x_j(\vec{y_j}).P_j)$$

with each P_j being itself in normal form. It can be verified that every *minimal process* is structurally equivalent to a normal form.

For the sake of clarity let us confine our attention to the case in which the formulae $[\![P]\!]$ are monadic—we omit the arity index "1" in $\mathtt{out}^1$. This case exhibits the essential structure of the proof. (The general case can be obtained essentially the same way but using a more involved definition of the measure function $M(\cdot)$ below needed for the induction: $M(F)$ is the smallest number of applications of Modus Ponens among the proofs of $F \models \exists z\, \mathtt{out}(x, z)$ in the Herbrand Proof System).

Given a formula F, define $M(F)$ as the number of implications in F. The proof proceeds by induction on $M([\![P]\!])$ assuming that P is a normal form $(\nu \vec{u})(\prod_{i \in I} !\overline{x_i} \mid \prod_{j \in J} !x_j.P_j)$. So, suppose that $[\![P]\!] = F \models \mathtt{out}(x)$ where

$$F = \exists \vec{u}(\bigwedge_{i \in I} \mathtt{out}(x_i) \wedge \bigwedge_{j \in J} \mathtt{out}(x_j) \Rightarrow [\![P_j]\!]).$$

We verify that (1) $\exists \vec{u}(\bigwedge_{i\in I} \mathtt{out}(x_i)) \models \mathtt{out}(x)$ or else (2) there exists $k \in I \cap J$ such that for $G =$

$$\exists \vec{u}(\bigwedge_{i\in I} \mathtt{out}(x_i) \wedge [\![P_k]\!] \wedge \bigwedge_{j\in J-\{k\}} \mathtt{out}(x_j) \Rightarrow [\![P_j]\!])$$

we have $G \models \mathtt{out}(x)$.

If (1) holds then we can show that $x = x_k$ for some $k \in I$ (and x_k does not occur in $\vec{u}$). From Definition 2.2 it follows that $P \Downarrow_{\overline{x}}$.

If (2) holds, we can find a P' such that $P \longrightarrow_{\mathrm{P}\pi} P'$ with $[\![P']\!] = G$. Such P' can be transformed into a structurally equivalent normal form Q such that $M([\![Q]\!]) = M(G)$ and $[\![Q]\!]$ is logically equivalent to G—$[\![Q]\!]$ is exactly G but with all the prefixing existentials of $[\![P_k]\!]$ moved to the outermost position of G. Notice that $M(G) < M(F)$, hence by appeal to induction we get $Q \Downarrow_{\overline{x}}$. We also have $P \longrightarrow_{\mathrm{P}\pi} Q$ since $P \longrightarrow_{\mathrm{P}\pi} P' \equiv Q$. From Definition 2.2 we conclude $P \Downarrow_{\overline{x}}$ as wanted.

□

Remark A.1 (Restriction as Existential in the presence with Mismatch). In the π-calculus restriction is usually seen as a generator of a new (*fresh*) instance of a name while an existential just asserts the existence of *some* instance. Below we give a brief discussion on the correspondence between barbed observability and logical consequence in Theorem 4.14 in the presence of mismatch. It is worth pointing out, however, that often the π-calculus is presented without mismatch.

Consider $(\nu xy)P$ and its corresponding translation $\exists xy[\![P]\!]$. We can say that executing $(\nu xy)P$ means "generate two fresh name instances, call them η_x, η_y, and behave as $P[\eta_x/x, \eta_y/y]$". We can say that asserting $\exists xy[\![P]\!]$ means "For *some* two elements $\varepsilon_x, \varepsilon_y$, $[\![P]\!][\varepsilon_x/x, \varepsilon_y/y]$ holds". Now, we can conclude that $\eta_x \neq \eta_y$ but we cannot conclude neither $\varepsilon_x = \varepsilon_y$ nor $\varepsilon_x \neq \varepsilon_y$. However, as for the outputs performed during *reduction* (or outputs inferred during deduction), knowing that $\eta_x \neq \eta_y$ and not knowing whether $\varepsilon_x = \varepsilon_y$ have the same effect. This holds even if match is present, but obviously it does not if mismatch is added. As an example $(\nu xy)([x \neq y]!\overline{t})$ performs $\overline{t}$ but $\mathtt{out}(t)$ does not follow from $\exists xy : (x \neq y \Rightarrow \mathtt{out}(t))$. □

Theorem. *4.17. Let $\Sigma = \mathrm{P}\pi^0$. Given P, Q in Σ, the questions whether*

1. *(Output Equivalence)$P \mathrel{\mathring{\simeq}}^{\Sigma} Q$,*
2. *(Output Congruence)$P \simeq^{\Sigma} Q$,*
3. *(Barbed Bisimilarity) $P \mathrel{\dot{\approx}}^{\Sigma} Q$*
4. *(Barbed Congruence) $P \approx^{\Sigma} Q$*

are all decidable.

Proof (Outline). The decidability of the first question can be obtained from Lemma 4.16(4) and the simple observation that the number of barbs of a given process is bounded by its number of free names. The decidability of the last two

problems follows from that of the first (applying Lemma 4.3) and the second (applying Corollary 4.3) respectively. We outline below the main steps and constructions for the decidability of the second question.

Let $\mathcal{O}(I) = \{\bar{x} \mid P \Downarrow_{\bar{x}}\}$. From Lemma 4.5, given P and Q the problem reduces to decide whether for every substitution σ and for every T, $\mathcal{O}(P\sigma \mid T) = \mathcal{O}(Q\sigma \mid T)$. Since all input contexts have arity zero $\mathrm{P}\pi^0$ it is easy to show that the problem reduces to decide whether for every T (the tester process)

$$\mathcal{O}(P \mid T) = \mathcal{O}(Q \mid T).$$

We have an infinite number of potential tester processes. We next show how to construct effectively a finite set of testers $T(P, Q)$.

Let $[\![\cdot]\!] : \mathrm{P}\pi \rightharpoonup \mathrm{FOL}$ be the map in Definition 4.12. Notice that $[\![P]\!]$ and $[\![Q]\!]$ are formulae without universal quantifiers. Therefore they can be rewritten as $F_P = \exists \vec{u} F'$ and $G_Q = \exists \vec{w} : G'$, respectively, where F' and G' are quantifier-free (and the images of some $\mathrm{P}\pi^0$ processes) and the variables in $\vec{u}$ and $\vec{w}$ are all fresh.

Define $T(F_P, G_Q)$ as the smallest set such that (1) $\mathtt{out}(z) \in T(F_P, G_Q)$ if $\mathtt{out}(z)$ appears, with z free, in F_P or G_Q (2) $M \wedge N \in T(F_P, G_Q)$ if $M, N \in T(F_P, G_Q)$ and (3) $\mathtt{out}(z) \Rightarrow M \in T(F_P, G_Q)$ if $\mathtt{out}(z), M \in T(F_P, G_Q)$.

Clearly $T(F_P, G_Q)$ is finite (up-to logical equivalence) and can be effectively constructed. Observe that for each $H \in T(F_P, G_Q)$ we can construct the unique $\mathrm{P}\pi^0$ process R such that $[\![R]\!] = H$. Hence, we can effectively construct the set $T(P, Q) = \{R \mid [\![R]\!] \in T(F_P, G_P)\}$.

With the above construction, the problem reduces to decide whether for every $T \in T(P, Q)$

$$\mathcal{O}(P \mid T) = \mathcal{O}(Q \mid T)$$

whose decidability follows from Lemma 4.16, and the construction of $T(P, Q)$.

The correctness of the above problem reduction can be verified from the following observations: Suppose we have a T and a process reduction of $P \mid T$ that produces an output (barb) $\overline{z}$ which no reduction of $Q \mid T$ can produce; i.e., $(P \mid T) \Downarrow_{\overline{z}}$ and $(Q \mid T) \not\Downarrow_{\overline{z}}$ (the other case is analogous.) We observe that in the reduction of $P \mid T$ we have the following interactive behaviour: If (a reduction of) P needs an output $\overline{x}_1$ of T to proceed then $\overline{x}_1 \in T(P, Q)$. Similarly, if T needs an output $\overline{y}_1$ of P to proceed then $\overline{y}_1 \in T(P, Q)$, and so on. For this reduction the interaction behavior between P and T is the same as that of P with $T' = (\overline{x}_1 \mid y_1.(\overline{x}_2 \mid y_2.(\ldots))$ which is in $T(P, Q)$. Now, if $\overline{z}$ is output in the run by (a reduction of) P then it is easy to verify that $(P \mid T') \Downarrow_{\overline{z}}$ and $(Q \mid T') \not\Downarrow_{\overline{z}}$. Else $\overline{z}$ is output by T. In this case we can prove that there must be a $\overline{y}_i$ output by P in the reduction which in no reduction Q in parallel with T can output. Hence $(P \mid T') \Downarrow_{\overline{y}_i}$ and $(Q \mid T') \not\Downarrow_{\overline{y}_i}$ as wanted. □

Proofs for section 4.3: Turing Expressiveness of $\mathrm{P}\pi$

Definition A.2. *An encoded machine $P = [\![M_s(v_0, v_1)]\!]$ such that location $P \downarrow \overline{l_k}$ has location k* enabled *(note the strong observation predicate).*

Definition A.3.
$Dec_0(c) = (\nu s, z)(!\,\overline{c}\langle s, z\rangle \mid !\,z\,.\,!\,\overline{yes})$
$Dec_i(c) = (\nu s, z)(!\,\overline{c}\langle s, z\rangle \mid !\,s(p)\,.\,Dec_{i-1}(p))$ *for* $i > 0$ *with corresponding definitions for* Dec_n^k *with observations on* $\overline{yes}^k$*, for* $k \in \{1, 2\}$

Lemma A.4. $Dec_x(c) \mid [\![y]\!]_c \Downarrow \overline{yes}$ *iff* $x = y$.

Lemma A.5.

1. *For an encoded machine* $P = [\![M_s(v_0, v_1)]\!]$ *with location* k *enabled, for* $Q = P \mid !\,l_k(c_0, c_1)\,.\,(Dec_x^1(c_0) \mid Dec_y^2(c_1))$*, if* $Q \Downarrow \overline{yes}^1$ *and* $Q \Downarrow \overline{yes}^2$*, then* $M_s(v_0, v_1)$ *can execute location* k *with counter values* $c_0 = x, c_1 = y$.
2. *If a machine* $M_s(v_0, v_1)$ *can execute at location* k *with counter values* $c_0 = x, c_1 = y$*, the encoded machine* $P = [\![M_s(v_0, v_1)]\!]$ *will have location* k *enabled, and for* $Q = P \mid !\,l_k(c_0, c_1)\,.\,(Dec_x^1(c_0) \mid Dec_y^2(c_1))$*, then* $Q \Downarrow \overline{yes}^1$ *and* $Q \Downarrow \overline{yes}^2$.

Proof. (Sketch) By induction on the length of computations, using Lemma A.4. □

Theorem. *4.18 A Minsky machine* $M_s(v_0, v_1)$ *computes the value* n *iff*

$$[\![M_s(v_0, v_1)]\!] \mid !\,halt(c)\,.\,Dec_n(c) \Downarrow \overline{yes}$$

Proof. From Lemma A.5 and minimal reasoning about the `halt` instruction. □

Theorem. *4.20. Let* $\Sigma = \mathrm{P}\pi$*. Given* P, Q *in* Σ*, the questions of whether*

1. *(Output Equivalence)* $P \mathrel{\dot{\simeq}}^{\Sigma} Q$,
2. *(Output Congruence)* $P \simeq^{\Sigma} Q$,
3. *(Barbed Bisimilarity)* $P \mathrel{\dot{\approx}}^{\Sigma} Q$,
4. *(Barbed Congruence)* $P \approx^{\Sigma} Q$

are all undecidable.

Proof. The undecidability of the first question can be obtained from Lemma 4.16 and from the reduction: $P \Downarrow_{\overline{z}}$ iff $(\nu \vec{x})P \mathrel{\dot{\simeq}}^{\Sigma} \overline{z}$ where $\vec{x}$ includes all the free names of P but z. The undecidability of the second question can be obtained from the first one as follows. Given P and Q, let $\vec{x} = x_i \ldots x_n$ be a sequence with all the free names $fn(P) \cup fn(Q)$. For each x_i, define a unique $z_i \notin fn(P) \cup fn(Q)$. For the sake of clarity, assume that each $\overline{x}\langle \vec{z}\rangle$ in P or Q, with $x \in \vec{x}$, has arity k and let $\vec{y}$ be a vector of names of length k which does not include any z_i.

We can verify that in $\mathrm{P}\pi$: $P \mathrel{\dot{\simeq}}^{\Sigma} Q$ iff

$$(\nu \vec{x})(P \mid \prod_{i=0}^{i=n} !x_i(\vec{y}).!\overline{z}_i) \simeq^{\Sigma} (\nu \vec{x})(Q \mid \prod_{i=0}^{i=n} !x_i(\vec{y}).!\overline{z}_i)$$

The undecidability of the last two problems follow from that of the first (applying Lemma 4.3) and the second (applying Corollary 4.3) respectively. □

Theorem. *5.9 Let* $\Sigma = \mathrm{PO}\pi^1$. *Given* $P, Q \in \Sigma$, *the question of whether* $P \approx^{\Sigma} Q$ *is undecidable.*

Proof. The undecidability of barbed congruence for $\mathrm{PO}\pi^1$ can be obtained from that of barbed bisimilarity for $\mathrm{PO}\pi^1$ as in the proof of Theorem 4.20. The undecidability of barbed bisimilarity for $\mathrm{PO}\pi^1$ follows that of π^1, Proposition 5.6 and the observation that the encoding $[\![\cdot]\!] : \pi \to \mathrm{PO}\pi$ in Definition 5.5 preserves arity for source $\mathrm{PO}\pi^1$ terms. □

Remark A.6 (5.8, Malicious Contexts). Since our encoding simulates an atomic communication with a sequence of finer communications, one can envisage a malicious context which does not behave according to the protocol thus causing the whole system to break. This is the same kind of problem of the standard encoding of polyadic communication into monadic one for the π-calculus [14]. In fact, the following construction is a counter-example to full-abstraction w.r.t. barbed congruence:

Example A.7. Take $P = x.x.0$ and $Q = x.0 \mid x.0$. Clearly, $P \approx Q$. Let $C_t = !\overline{x}\langle n\rangle \mid !\overline{x}\langle m\rangle \mid !n(\,r\,t\,f\,).!m(\,r'\,t'\,f'\,).!\overline{t}$. Verify that $(C_t \mid [\![Q]\!]) \Downarrow_{\overline{t}}$ but $(C_t \mid [\![P]\!]) \not\Downarrow_{\overline{t}}$. Hence, $[\![P]\!] \not\approx [\![Q]\!]$.

Nevertheless, as for the encoding of polyadic into monadic [18,23], we believe we can provide a type system in order to give a stronger correspondence for the encoding. Basically, the type system would allow contexts that may not behave as dictated by the protocol but do not interfere either. However, this is out of the scope of this paper.