Functions and References in the Pi-Calculus: Full Abstraction and Proof Techniques

Enguerrand Prebet

Functions and References in the Pi-Calculus: Full Abstraction and Proof Techniques

Enguerrand Prebet
Université de Lyon, ENS de Lyon, UCB Lyon 1, CNRS, INRIA, LIP

Abstract

We present a fully abstract encoding of $\lambda^{\mathtt{ref}}$, the call-by-value λ-calculus with references, in the π-calculus. By contrast with previous full abstraction results for sequential languages in the π-calculus, the characterisation of contextual equivalence in the source language uses a labelled bisimilarity. To define the latter equivalence, we refine existing notions of typed bisimulation in the π-calculus, and introduce in particular a specific component to handle divergences.

We obtain a proof technique that allows us to prove equivalences between $\lambda^{\mathtt{ref}}$ programs via the encoding. The resulting proofs correspond closely to normal form bisimulations in the λ-calculus, making proofs in the π-calculus expressible as if reasoning in $\lambda^{\mathtt{ref}}$.

We study how standard and new up-to techniques can be used to reason about diverging terms and simplify proofs of equivalence using the bisimulation we introduce. This shows how the π-calculus theory can be used to prove interesting equivalences between $\lambda^{\mathtt{ref}}$ programs, using algebraic reasoning and up-to techniques.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning

Keywords and phrases Call-by-value λ-calculus, imperative Programming, π-calculus, Bisimulation, Type System

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.114

Funding This work has been supported by the Université Franco-Italienne under the programme Vinci 2020.

Acknowledgements Many thanks to Daniel Hirschkoff and Davide Sangiorgi for fruitful discussions and helpful comments on earlier versions of this article.

1 Introduction

Milner [11] described the first encodings from the λ-calculus to the π-calculus for both call-by-name and call-by-value. These encodings are shown to be sound with respect to contextual equivalence, enabling the use of the π-calculus as a model to prove equivalence between λ-terms. In sequential languages like the λ-calculus or extensions thereof, contextual equivalence is often considered as the canonical equivalence. The π-calculus offers a rich theory of behavioural equivalences and preorders to reason coinductively and algebraically about processes. In the π-calculus, several powerful up-to techniques can be used and combined in a modular way to make bisimulation proofs shorter and more readable [10, 16]. Labelled bisimulations can also be refined by means of type systems.

Nevertheless, for sequential languages, while such equivalences in the π-calculus lead to sound encodings with respect to contextual equivalence, completeness does not hold, due to the parallel nature of the π-calculus. Full abstraction is obtained for stronger equivalences in the source language, generally based on trees: Lévy-Longo Trees for the call-by-name λ-calculus [18], and η-eager normal form bisimilarity for call-by-value [3]. More generally, we are not aware of full abstraction results relating a contextual equivalence in a sequential language and a labelled bisimilarity between process terms in the encoding. Existing results either use a finer relation than contextual equivalence in the source language, or obtain full abstraction for a contextually-defined equivalence in the π-calculus [1, 21].

In this work, we study $\lambda^{\mathtt{ref}}$, the call-by-value λ-calculus with references. Our first main contribution is to characterise contextual equivalence in $\lambda^{\mathtt{ref}}$ using a labelled bisimilarity in the π-calculus. The encoding of functions is extended with references as described in [4]. To define our labelled bisimulation, we rely on *well-bracketed bisimulation* [5], wb-bisimulation for short. Wb-bisimulations enforce a well-bracketing discipline between function calls and returns, and is formalised by a type system. Using a type system limits the interactions a process can have with its environment, and thus makes the corresponding bisimulation coarser than an untyped one.

This study allows us to understand the expressiveness of wb-bisimulation and to improve it. As mentioned in [2], to capture contextual equivalence in $\lambda^{\mathtt{ref}}$, *deferred divergent terms* need to be taken into account. Intuitively such terms hide a divergence behind stuck terms, like e.g. in $(\lambda y.\,\Omega)(xV)$, where Ω is an always diverging term and V is a value: the stuck term xV prevents the β-reduction yielding Ω to be applied. Still, this term is contextually equivalent to Ω. We define an equivalent notion for typeable π-terms, called π-divergence, which is used to refine wb-bisimulation yielding a notion of *bisimulation with divergence* in the π-calculus.

Another main contribution of this work is to define this new bisimulation with divergence and to study how up-to techniques, existing and new, can be used to simplify bisimulation proofs for this new equivalence. As π-divergence is defined coinductively, up-to techniques can also be defined to enhance the proofs of divergence. We also introduce a new up-to technique, to which we return below; this technique is *compatible*, meaning that it fits in the existing theory of up-to techniques for the π-calculus.

We do not prove full abstraction with respect to contextual equivalence in $\lambda^{\mathtt{ref}}$ directly, but rather exploit its characterisation using *normal form bisimilarity* [2]. Normal form bisimulations in [2], which we call nfb in the sequel, and bisimulations with divergence in the π-calculus are strongly connected, and we can establish the following result:

> If $\mathcal{R}$ is a nfb, then the encoding of $\mathcal{R}$ is a bisimulation with divergence up to expansion in the π-calculus. (1)

This property allows us to prove that bisimilarity with divergence is complete w.r.t. nfb. Thanks to (1), when proving equivalences of $\lambda^{\mathtt{ref}}$-terms, the π-calculus machinery running under the hood is almost invisible: the bisimulations we manipulate are mainly built using (the encoding of) $\lambda^{\mathtt{ref}}$-terms. This makes π-calculus a powerful environment to prove equivalences between $\lambda^{\mathtt{ref}}$ programs. We illustrate this power via several examples throughout the paper. In some cases, we are able to compare $\lambda^{\mathtt{ref}}$ programs using equivalences which are finer, and simpler to use, than bisimulation with divergence.

We now give more details about the encoding from $\lambda^{\mathtt{ref}}$ to the π-calculus. Milner's original encoding of the untyped call-by-value λ-calculus does not handle equivalences between terms with free variables well. This issue is amended in [3] by moving to the internal π-calculus [19], where only fresh names can be transmitted. For instance, the resulting encodings of $(\lambda z.\,z)(xV)$ and xV are equivalent, which is not the case for Milner's original encoding.

Our results are based on the optimised version of the encoding of [3], where administrative reductions introduced in the encoding of the application are removed. In the optimised encoding, a stuck term like xV is translated into a process that cannot reduce. Additionally, the translation of an evaluation context always yields an evaluation context, which is not the case in the encoding of [3]. These two properties lead to a tight statement of operational

correspondence: the encoding of a $\lambda^{\mathtt{ref}}$-term can perform an internal communication only if the source term has a reduction.

When considering non-internal transitions, the picture becomes more complicated because residual processes appear. We write $[\![M]\!]_p$ for the encoding of M at p — as usual, the π-calculus encoding of a term is parametric over some name p. To illustrate operational correspondence for non-internal transitions, we have for instance $[\![E[xV]]\!]_p \xrightarrow{\overline{x}(y,q)} \gtrsim [\![V]\!]_y \mid q(z).\,[\![E[z]]\!]_p$, where E is an evaluation context and $\gtrsim$ stands for the expansion preorder. The encoding of the stuck term $E[xV]$ sends two links at x, one to its argument V (via y) and one to its continuation E (via q). Both processes are then accessible by the context, and can be executed (possibly in parallel). Intuitively, the value and continuation belong to some environment. This is close to what happens in a nfb, whereby terms are related in the presence of an environment, containing both values and contexts. Thus, it is natural to encode a nfb as a binary relation on π-calculus processes.

To prevent the context from performing unwanted transitions (like parallel calls), we use the type system for well-bracketing from [5]. Typing constraints give rise to *type-allowed* transitions. With this restriction, we obtain a typed version of operational correspondence which is in one-to-one correspondence with the clauses defining nfb. In particular, in the transition of $[\![E[xV]]\!]_p$ above, the resulting process is the encoding of the corresponding state in the nfb. Soundness and completeness of the encoding, as well as property (1), are obtained as consequences of such operational correspondence.

We explain briefly the new up-to technique we introduce. Intuitively, transitions in $\lambda^{\mathtt{ref}}$ may generate several copies of the same value, which are stored in the environment. In [2], the environment is a set, and thus duplicates are removed for free. In the standard π-calculus, these copies would be the same replicated process, and duplicates could be removed using the standard law $!P \mid !P \sim !P$. In the internal π-calculus, each copy is accessible via a distinct name. However, when these copies are only accessible by the context, it is sound to remove duplicates. The new up-to technique we introduce, called *up-to body*, formalises this idea. Together with standard up-to techniques for the pi-calculus, we use the up-to body technique to prove several examples of equivalences between $\lambda^{\mathtt{ref}}$ programs.

Paper outline.

We start by presenting the necessary background: we describe $\lambda^{\mathtt{ref}}$ and normal form bisimulations from [2] in Section 2; in Section 3, we introduce the Asynchronous Internal π-calculus, AIπ, and the type system for well-bracketing. We continue with the contributions of this work, describing the encoding for $\lambda^{\mathtt{ref}}$ terms in Section 4. We also show an operational correspondence and examples that can be proved with untyped equivalences, using algebraic reasoning and standard algebraic laws. The extension of the encoding to bisimulation states, together with the definition of bisimulation with divergence, are given in Section 5, where we establish full abstraction. We present up-to techniques for wb-bisimulation in Section 6, and show how they can be used on some examples.

2 Normal Form Bisimulation for a λ-Calculus with References

We define the syntax of the λ-calculus with references, noted $\lambda^{\mathtt{ref}}$:

Terms:	$M, N ::= V \mid M\ N \mid \ell := M; N \mid\ !\ell \mid \mathtt{new}\ \ell := V\ \mathtt{in}\ M$
Values:	$V, W ::= x \mid \lambda x.\, M$
Evaluation contexts:	$E, F ::= [\cdot] \mid E\ M \mid V\ E \mid \ell := E; M$

Free variables for terms and contexts are defined as usual with $\lambda x.\, M$ as binder. We write $M\{^V/_x\}$ for the usual capture-avoiding substitution of x by V in M. And we let $\int$ range over simultaneous substitutions $\{^{V_1}/_{x_1}\} \dots \{^{V_n}/_{x_n}\}$ where $x_1, \dots, x_n$ are pairwise different variables.

Free references for terms and contexts, noted $\mathrm{fr}(M)$ and $\mathrm{fr}(E)$, are defined similarly, with $\texttt{new}\ \ell \coloneqq V\ \texttt{in}\ M$ binding ℓ in M. Notice that ℓ is not a value, meaning that in $\texttt{new}\ \ell \coloneqq V\ \texttt{in}\ M$, ℓ is local to the term M. To give access to a local reference, M may pass functions $\lambda x.\, !\ell$ and $\lambda x.\, \ell \coloneqq x$ instead of ℓ.

A store, noted $h, g, \dots$, is a partial fonction with finite domain from references to values. We write $\emptyset$ for the empty store and $\mathrm{dom}(h)$ for the set of references on which h is defined. For any ℓ, V, we write $h \uplus \ell = V$ for the store h extended with a reference ℓ containing V and $h[\ell \coloneqq V]$ for the store h with the content of ℓ updated to the value V.

The semantics are defined on *configurations* $\langle h \mid M \rangle$ where h is a store.

We assume that for all configurations $\langle h \mid M \rangle$, we have $\mathrm{fr}(M) \subseteq \mathrm{dom}(h)$ and for all $\ell \in \mathrm{dom}(h)$, $\mathrm{fr}(h(\ell)) \subseteq \mathrm{dom}(h)$. This assumption ensures that any free reference used in terms is defined in h.

Reductions between configurations are defined as follows:

$$\langle h \mid (\lambda x.\, M)V \rangle \to \langle h \mid M\{^V/_x\} \rangle \qquad (\beta)$$

$$\langle h \mid !\ell \rangle \to \langle h \mid h(\ell) \rangle \qquad \text{(Read)}$$

$$\langle h \mid \ell \coloneqq v; M \rangle \to \langle h[\ell \coloneqq v] \mid M \rangle \qquad \text{(Write)}$$

$$\langle h \mid \texttt{new}\ \ell \coloneqq v\ \texttt{in}\ M \rangle \to \langle h \uplus \ell = v \mid M \rangle \qquad \text{(Alloc)}$$

$$\langle h \mid E[M] \rangle \to \langle g \mid E[N] \rangle \qquad \text{if } \langle h \mid M \rangle \to \langle g \mid N \rangle \qquad \text{(Eval)}$$

A term M in a configuration $\langle h \mid M \rangle$ which cannot reduce is called a normal form. It can either be a value, or a stuck term of the form $E[yV]$.

▶ **Lemma 1.** *For any $\langle h \mid M \rangle$ we have:*

1. *either $\langle h \mid M \rangle \to \langle h' \mid M' \rangle$ for some $\langle h' \mid M' \rangle$*
2. *or M is a value*
3. *or M is of the form $E[yV]$.*

We write $\Rightarrow$ for $\to^*$ and we say that $\langle h \mid M \rangle$ and $\langle g \mid N \rangle$ *co-terminate* when $\langle h \mid M \rangle \Rightarrow \langle h' \mid M' \rangle$ with M' being a normal form iff $\langle g \mid N \rangle \Rightarrow \langle g' \mid N' \rangle$ with N' being a normal form. A term (resp. context) is *closed* (resp. *reference-closed*) if its set of free variables (resp. free references) is empty. A substitution $\int$ is *closing terms* $M, N, \dots$ if $M\int, N\int, \dots$ are closed.

▶ **Definition 2.** *Two reference-closed terms are contextually equivalent, written $M \asymp N$, if for all closing substitutions $\int$ and reference-closed contexts E, $\langle \emptyset \mid E[M\int] \rangle$ and $\langle \emptyset \mid E[N\int] \rangle$ co-terminate.*

To define normal form bisimulations, we need to introduce the notion of *triples*, used in [2].

We use a tilde, like in $\widetilde{V_i}$, to denote a (possibly empty) tuple. Triples are of the form $(\widetilde{V_i}, \sigma, c)$ and $(\widetilde{V_i}, \sigma, h)$ where: $\widetilde{V_i}$ is a tuple of values, σ is a stack of evaluation contexts, c is a configuration and h is a store. These are the elements being compared in a normal form bisimulation. We provide some comments about the role of triples.

The tuple $\widetilde{V_i}$ stores the values accumulated by the environment, and we write $(\widetilde{V_i}, V)$ for the tuple $\widetilde{V_i}$ extended with the additional value V. A stack of evaluation contexts σ corresponds to interrupted contexts that must be executed to complete the computation. ⊙

stands for the empty stack and $E :: \sigma$ for the stack obtained by adding E on top of σ. The store is accessible by all the other objects of the triple, values or contexts – in $(\widetilde{V_i}, \sigma, c)$, the store is part of c.

Intuitively, a triple of the form $(\widetilde{V_i}, \sigma, c)$ is active, meaning it is computing up until a normal form term is obtained, at which point the computation proceeds to triples of the form $(\widetilde{V_i}, \sigma, h)$ where the environment is carrying on the computation, deciding to call one of the accumulated functions or to resume the computation by evaluating the context at the top of the stack.

For instance, in the triple $(\widetilde{V_i}, \sigma, \langle h \mid E[yV]\rangle)$, the environment is about to get access to V and the function corresponding to y will run before eventually (in absence of divergence) returning the result that will be used by E. Thus, the "next" triple is $((\widetilde{V_i}, V), E :: \sigma, h)$: value V and context E are added to the corresponding tuple and stack, and h is kept identical while the environment is deciding for the next move. This evolution can be seen in Definitions 4 and 5.

Following [8], we first need to introduce an auxiliary relation which performs an immediate beta reduction in a term of the form VW whenever possible.

▶ **Definition 3** (Relation $\succ$). *We write $VW \succ N$ when $V = \lambda y.\, N'$ and $N = N'\{^W/_y\}$ or when $V = z$ and $N = VW$.*

We say that a term is deferred diverging if it hides a diverging behaviour behind a stuck term, e.g $(\lambda x.\, \Omega)(yV)$. This notion can be extended to triples to capture all diverging behaviours, including deferred ones, as defined below. Intuitively, a triple is diverging if it contains a diverging, possibly deferred, context or configuration.

Formally, the set of diverging triples is defined coinductively using a diverging set.

▶ **Definition 4** (Diverging set). *A set S of triples is* diverging *if the two following conditions hold:*

1. $(\widetilde{V_i}, \sigma, c) \in S$ *implies*
   - **a.** *if $c \to c'$, then $(\widetilde{V_i}, \sigma, c') \in S$*
   - **b.** *if $c = \langle h \mid V\rangle$, then $\sigma \neq \emptyset$ and $((\widetilde{V_i}, V), \sigma, h) \in S$*
   - **c.** *if $c = \langle h \mid E[yV]\rangle$, then $((\widetilde{V_i}, V), E :: \sigma, h) \in S$*
2. $(\widetilde{V_i}, \sigma, h) \in S$ *implies*
   - **a.** *for every j and fresh x, $(\widetilde{V_i}, \sigma, \langle h \mid M\rangle) \in S$ with $V_j x \succ M$*
   - **b.** *if $\sigma = E :: \sigma'$, then $(\widetilde{V_i}, \sigma', \langle h \mid E[x]\rangle) \in S$ for x fresh*

We note λ_{div} the largest diverging set.

We say that a relation $\mathcal{R}$ is a *triples relation* if the two elements of any pair of triples in $\mathcal{R}$ both contain a tuple of values and stack of the same size, and either both contain a configuration or both contain a store.

We can now define the normal form bisimulation:

▶ **Definition 5** (nfb). *A* symmetric *triples relation $\mathcal{R}$ is a normal form bisimulation when there exists a diverging set S such that:*

1. $(\widetilde{V_i}, \sigma_1, c)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, d)$ *implies*
   - **a.** *if $c \to c'$, then $d \Rightarrow d'$ and $(\widetilde{V_i}, \sigma_1, c')\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, d')$*
   - **b.** *if $c = \langle h \mid V\rangle$, then either*
     - **i.** *$d \Rightarrow \langle g \mid W\rangle$ and $((\widetilde{V_i}, V), \sigma_1, h)\ \mathcal{R}\ ((\widetilde{W_i}, W), \sigma_2, g)$, or*
     - **ii.** *$\sigma_1 \neq \emptyset$ and $((\widetilde{V_i}, V), \sigma_1, h) \in S$*
   - **c.** *if $c = \langle h \mid E[yV]\rangle$, then either*
     - **i.** *$d \Rightarrow \langle g \mid F[yW]\rangle$ and $((\widetilde{V_i}, V), E :: \sigma_1, h)\ \mathcal{R}\ ((\widetilde{W_i}, W), F :: \sigma_2, g)$, or*

   ii. $((\widetilde{V_i}, V), E :: \sigma_1, h) \in S$
2. $(\widetilde{V_i}, \sigma_1, h)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, g)$ *implies*
   a. *for every* j *and fresh* x, $(\widetilde{V_i}, \sigma_1, \langle h \mid M\rangle)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2, \langle g \mid N\rangle)$ *with* $V_j x \succ M$ *and* $W_j x \succ N$
   b. *if* $\sigma_1{=}E :: \sigma_1'$ *and* $\sigma_2{=}F :: \sigma_2'$, *then* $(\widetilde{V_i}, \sigma_1', \langle h \mid E[x]\rangle)\ \mathcal{R}\ (\widetilde{W_i}, \sigma_2', \langle g \mid F[x]\rangle)$ *for* x *fresh*

Normal form bisimilarity, $\approx_\lambda$*, is the largest normal form bisimulation.*

Definition 5 reformulates the bisimulation from [2]. This gives the same equivalence while being more suitable to establish the operational correspondence of the encoding (Theorem 17). We can thus rely on the following result.

▶ **Theorem 6** (Full abstraction [2, Theorems 3,4]). *For all* $\lambda^{\mathtt{ref}}$*-terms* M, N*, we have* $(\emptyset, \odot, \langle \emptyset \mid M\rangle) \approx_\lambda (\emptyset, \odot, \langle \emptyset \mid N\rangle)$ *iff* $M \asymp N$.

This means that to prove that two reference-closed terms are contextually equivalent, we can provide a normal form bisimulation $\mathcal{R}$ relating the two terms and its corresponding diverging set S.

3 AIπ, the Asynchronous Internal π-Calculus

We present the Asynchronous Internal π-calculus, AIπ, and the type system for well-bracketing from [5]. The syntax is given by the following grammar:

Processes	$P, Q ::= \mathbf{0} \mid \overline{a}(\widetilde{b})\ P \mid a(\widetilde{b}).\, P \mid !a(\widetilde{b}).\, P \mid P \mid Q \mid (\boldsymbol{\nu} a)P \mid K\lfloor\widetilde{a}\rfloor$
Recursive definitions	$K \triangleq (\widetilde{a})\ P$

As for $\lambda^{\mathtt{ref}}$ values, $\widetilde{b}$ denotes a tuple of names. We also write $a(\_).\, P$ for $a(z).\, P$ when $z \notin \mathrm{fn}(P)$. Here, $\overline{a}(\widetilde{b})\ P$ is a bound asynchronous output, introduced in [1]. This corresponds to $(\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{b}\rangle.\, \mathbf{0} \mid P)$ in the standard π-calculus, that is, a non-blocking output which carries bound names used in P. Constants are defined as an abstraction $(K \triangleq (\widetilde{a})\ P)$, where $\widetilde{a}$ are distinct names, bound in P. Given an abstraction $(\widetilde{a})\ P$, we note $((\widetilde{a})\ P)\lfloor\widetilde{b}\rfloor$ the process $P\{\widetilde{b}/\widetilde{a}\}$. Thus $K\lfloor\widetilde{a}\rfloor$ represents the process in the definition of K substituted with the names $\widetilde{a}$ (as expressed by the rule Cst in the operational semantics).

The full set of rules defining the Labelled Transition System (LTS) is given in Figure 1. Symmetric rules for Par and Comm have been omitted. It is similar to that of Iπ with the additional rules:

$$\textsc{Async}\ \frac{P \xrightarrow{\mu} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\mu} \overline{a}(\widetilde{b})\ P}\ \widetilde{b} \cap (\mathrm{fn}(\mu) \cup \mathrm{bn}(\mu)) = \emptyset,\ a \notin \mathrm{bn}(\mu) \qquad \textsc{AComm}\ \frac{P \xrightarrow{a(\widetilde{b})} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{b})P'}$$

AComm allows P to communicate with the bound output, and Async allows P to perform any action that does not involve the names bound by the output.

Our type system is based on Milner's *sorting*. Names are split into *sorts*. In our case, we will use 3 sorts: F,R,C. The sorting function Σ is defined by $\Sigma(\mathtt{F}) = (\mathtt{F}, \mathtt{C})$ and $\Sigma(\mathtt{R}) = \Sigma(\mathtt{C}) = \mathtt{F}$. We call *function names*, ranged over with $w, x, y, z, \ldots$, names in F, *reference names*, ranged over with $\ell, \ell', \ldots$, names in R, and *continuation names*, ranged over with $p, q, r, \ldots$, names in C.

On top of the sorting, we adapt the type system for well-bracketing from [5] to AIπ. The type system imposes that continuation names are *receptive linear*, meaning that they can be used only once in output and once in input (hence linear) and the input is immediately

$$\textsc{Inp}\ \frac{}{a(\widetilde{b}).\,P \xrightarrow{a(\widetilde{b})} P} \qquad \textsc{Out}\ \frac{}{\overline{a}(\widetilde{b})\ P \xrightarrow{\overline{a}(\widetilde{b})} P} \qquad \textsc{Async}\ \frac{P \xrightarrow{\mu} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\mu} \overline{a}(\widetilde{b})\ P'}\ \begin{array}{l}\widetilde{b} \cap (\mathrm{fn}(\mu) \cup \mathrm{bn}(\mu)) = \emptyset \\ a \notin \mathrm{bn}(\mu)\end{array}$$

$$\textsc{AComm}\ \frac{P \xrightarrow{a(\widetilde{b})} P'}{\overline{a}(\widetilde{b})\ P \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{b})P'} \qquad \textsc{Rep}\ \frac{a(\widetilde{b}).\,P \xrightarrow{\mu} P'}{!a(\widetilde{b}).\,P \xrightarrow{\mu} P' \mid\, !a(\widetilde{b}).\,P} \qquad \textsc{Res}\ \frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu}a)P \xrightarrow{\mu} (\boldsymbol{\nu}a)P'}\ a \notin \mathrm{fn}(\mu) \cup \mathrm{bn}(\mu)$$

$$\textsc{Par}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset \qquad \textsc{Comm}\ \frac{P \xrightarrow{a(\widetilde{b})} P' \qquad Q \xrightarrow{\overline{a}(\widetilde{b})} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\widetilde{b})(P' \mid Q')} \qquad \textsc{Cst}\ \frac{P\{\widetilde{b}/\widetilde{a}\} \xrightarrow{\mu} P'}{K\lfloor \widetilde{b} \rfloor \xrightarrow{\mu} P'} K \triangleq (\widetilde{a})\ P$$

Figure 1 Labelled Transition Semantics for AIπ

available as soon as it is created (hence receptive). Additionally, creations and communications at continuation names behave in a well-bracketed manner, meaning that the communication will first occur at the name created last.

We use two constants, noted $\triangleright_{\mathtt{F}}$ and $\triangleright_{\mathtt{C}}$, defined as follows:

$$\begin{array}{ll} \triangleright_{\mathtt{C}} \triangleq (p,q)\ p(x).\,\overline{q}(y)\ y \triangleright_{\mathtt{F}} x & \text{with } p,q \text{ being continuation names} \\ \triangleright_{\mathtt{F}} \triangleq (x,y)\ !x(z,p).\,\overline{y}(w,q)\ (q \triangleright_{\mathtt{C}} p \mid w \triangleright_{\mathtt{F}} z) & \text{with } x,y \text{ being function names} \end{array}$$

These constants represent a link, also called forwarder or dynamic wire [17], which transforms outputs at the first name into outputs at the second. As their usage only differs in the linearity of continuation names and the arity, we often use the same symbol $\triangleright$ to denote both $\triangleright_{\mathtt{F}}$ and $\triangleright_{\mathtt{C}}$.

$$\textsc{wb-Nil}\ \frac{}{\emptyset \vdash_{\mathrm{wb}} \mathbf{0}} \qquad \textsc{wb-AOutC}\ \frac{\rho \vdash_{\mathrm{wb}} P}{p : \mathtt{o}, \rho \vdash_{\mathrm{wb}} \overline{p}(y)\ P} \qquad \textsc{wb-AOutF}\ \frac{p : \mathtt{i}, \rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} \overline{x}(y,p)\ P} \qquad \textsc{wb-AOutR}\ \frac{\rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} \overline{\ell}(y)\ P}$$

$$\textsc{wb-InpC}\ \frac{p : \mathtt{o} \vdash_{\mathrm{wb}} P \qquad p \neq q}{q : \mathtt{i}, p : \mathtt{o} \vdash_{\mathrm{wb}} q(y).\,P} \qquad \textsc{wb-InpF}\ \frac{p : \mathtt{o} \vdash_{\mathrm{wb}} P}{\emptyset \vdash_{\mathrm{wb}} x(y,p).\,P,\ !x(y,p).\,P} \qquad \textsc{wb-InpR}\ \frac{p : \mathtt{o} \vdash_{\mathrm{wb}} P}{p : \mathtt{o} \vdash_{\mathrm{wb}} \ell(y).\,P}$$

$$\textsc{wb-ResC1}\ \frac{\rho, p : \star, \rho' \vdash_{\mathrm{wb}} P}{\rho, \rho' \vdash_{\mathrm{wb}} (\boldsymbol{\nu}p)P} \qquad \textsc{wb-ResC2}\ \frac{\rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} (\boldsymbol{\nu}p)P}\ p \notin \rho \qquad \textsc{wb-ResFR}\ \frac{\rho \vdash_{\mathrm{wb}} P}{\rho \vdash_{\mathrm{wb}} (\boldsymbol{\nu}y)P,\ (\boldsymbol{\nu}\ell)P}$$

$$\textsc{wb-Par}\ \frac{\rho \vdash_{\mathrm{wb}} P \qquad \rho' \vdash_{\mathrm{wb}} Q}{\rho'' \vdash_{\mathrm{wb}} P \mid Q}\ \rho'' \in \mathtt{inter}(\rho;\rho') \qquad \textsc{wb-FwC}\ \frac{}{p : \mathtt{i}, q : \mathtt{o} \vdash_{\mathrm{wb}} p \triangleright_{\mathtt{C}} q} \qquad \textsc{wb-FwF}\ \frac{}{\emptyset \vdash_{\mathrm{wb}} x \triangleright_{\mathtt{F}} y}$$

Figure 2 Type system for well-bracketing

We present the typing rules for well-bracketing in Figure 2. Judgements are of the form $\rho \vdash_{\mathrm{wb}} P$ with ρ being a stack. In rules WB-InpF and WB-ResFR, the conclusion is of the

form $\rho \vdash_{\text{wb}} P, Q$ to indicate that both $\rho \vdash_{\text{wb}} P$ and $\rho \vdash_{\text{wb}} Q$ can be inferred from the premise.

A stack is a sequence of *tagged* names, the tag being either $\mathtt{i}, \mathtt{o}$ or $\star$ denoting respectively input, output and both capabilities. Upon creation, a continuation comes with both capabilities (WB-RESC1), that are used once as input and output (WB-INPC and WB-AOUTC). In WB-AOUTF, a continuation is created and its input capability is passed to P, while its output capability is sent by the communication. This capability can then be used after the input in WB-INPF. As continuation names are receptive, inputs cannot appear after another input (WB-INPF and WB-INPC).

Intuitively, a stack expresses the expected usage of the free continuation names in a process. Stacks are given by the following grammars:

$$\rho \ ::= \ \rho^{\mathtt{o}} \mid \rho^{\mathtt{i}} \qquad \rho^{\mathtt{o}} \ ::= \ p : \mathtt{o}, \rho^{\mathtt{i}} \mid p : \star, \rho^{\mathtt{o}} \mid \emptyset \qquad \rho^{\mathtt{i}} \ ::= \ p : \mathtt{i}, \rho^{\mathtt{o}} \mid \emptyset$$

Moreover, a name may appear at most once in a stack, so we will say that a name is $\mathtt{o}$-*tagged* in ρ when the name appears with tag $\mathtt{o}$ in ρ and similarly for $\mathtt{i}$ and $\star$.

As a $\star$-tagged name represents both output and input capabilities, a stack can be seen as an alternation of input- and output-tagged names. For instance, if we have

$$p_1 : \mathtt{o}, p_2 : \mathtt{i}, p_3 : \star, p_4 : \mathtt{o} \vdash_{\text{wb}} P$$

then $p_1, \ldots, p_4$ are the free continuation names in P; among these, p_1 will be used first, as an output at p_1; then p_2 will be used, in an input interaction with the environment. P possesses both the output and the input capability on p_3, and will use both capabilities by performing a reduction at p_3; the computation for P terminates with an output at p_4.

Thus, to compose two processes P and Q in parallel, the usage of continuations of $P \mid Q$ is an interleaving of the ones of P and Q (WB-PAR). Names with capabilities shared between P and Q can be used as synchronisation point. Formally, the interleaving of two stacks is described by the following definition:

▶ **Definition 7.** *The interleaving relation is defined as a ternary relation between stacks, written* $\rho \in \mathtt{inter}(\rho_1 \ ; \ \rho_2)$*, and defined as follows:*

- $\emptyset \in \mathtt{inter}(\emptyset; \emptyset)$
- $\rho \in \mathtt{inter}(\rho_1 \ ; \ \rho_2)$ *implies* $\rho \in \mathtt{inter}(\rho_2 \ ; \ \rho_1)$*.*
- $\rho \in \mathtt{inter}(\rho_1 \ ; \ \rho_2)$ *implies* $p : \eta, \rho \in \mathtt{inter}(p : \eta, \rho_1 \ ; \ \rho_2)$*, where* $\eta \in \{\mathtt{o}, \mathtt{i}, \star\}$ *and* $p : \eta, \rho_1$ *is a stack.*
- *Whenever* $\rho \in \mathtt{inter}(\rho^{\mathtt{i}}; \rho^{\mathtt{o}})$*, we have* $p : \star, \rho \in \mathtt{inter}(p : \mathtt{o}, \rho^{\mathtt{i}} \ ; \ p : \mathtt{i}, \rho^{\mathtt{o}})$*.*

Even though the grammar allows it, it is not possible to type a process $\rho \vdash P$ with ρ ending with $p : \mathtt{i}$ or $p : \star$.

As continuation names are used linearly, when both input and output are present in the process, i.e., when the name is $\star$-tagged in the stack, this name should not be observable. We call *clean* a stack where no name is $\star$-tagged. We note $\mathtt{c}(\rho)$ the clean stack obtained by removing all $\star$-tagged names from ρ, and $\rho \vDash_{\text{wb}} P$ when there exists ρ' with $\rho' \vdash_{\text{wb}} P$ and $\mathtt{c}(\rho') = \rho$. Using clean stacks, we can define *typed transitions.*

▶ **Definition 8.** *When* $\rho \vDash_{\text{wb}} P$*, we write* $[\rho; P] \xrightarrow{\mu} [\rho'; P']$ *if* $P \xrightarrow{\mu} P'$ *and one of the following holds:*

1. $\mu = \overline{p}(x)$ *and* $\rho = p : \mathtt{o}, \rho'$
2. $\mu = p(x)$ *and* $\rho = p : \mathtt{i}, \rho'$
3. $\mu = \overline{x}(y, p)$ *and* $\rho' = p : \mathtt{i}, \rho$

4. $\mu = x(y,p)$ *and* $\rho' = p : \mathsf{o}, \rho$
5. $\mu \in \{\ell(x), \overline{\ell}(x), \tau\}$ *and* $\rho' = \rho$.

▶ **Lemma 9** (Subject reduction). *If* $[\rho; P] \xrightarrow{\mu} [\rho'; P']$, *then for any* Q *with* $\rho \vDash_{\text{wb}} Q$ *and* $Q \xrightarrow{\mu} Q'$, *we have* $\rho' \vDash_{\text{wb}} Q'$.

A relation $\mathcal{R}$ is *wb-typed* if for any $(\rho, P, Q) \in \mathcal{R}$, we have $\rho \vDash_{\text{wb}} P$ and $\rho \vDash_{\text{wb}} Q$. We define the typed version of the expansion preorder, written $\gtrsim_{wb}$, by restricting to typed transitions. As usual, we write $\xrightarrow{\widehat{\tau}}$ for $\xrightarrow{\tau} \cup \, id$ and $\xrightarrow{\widehat{\mu}}$ for $\xrightarrow{\mu}$ otherwise. The weak variants of the transitions are defined by chaining with $\Rightarrow \stackrel{\text{def}}{=} \xrightarrow{\tau}^*$: for instance, $\stackrel{\mu}{\Rightarrow} \stackrel{\text{def}}{=} \Rightarrow \xrightarrow{\mu} \Rightarrow$.

▶ **Definition 10** (Wb-expansion). *A wb-typed relation* $\mathcal{R}$ *is a* wb-expansion *when* $(\rho, P, Q) \in \mathcal{R}$ *implies:*

- *If we have* $[\rho; P] \xrightarrow{\mu} [\rho'; P']$, *then there exists* Q' *such that* $Q \xrightarrow{\widehat{\mu}} Q'$ *and* $(\rho', P', Q') \in \mathcal{R}$.
- *If we have* $[\rho; Q] \xrightarrow{\mu} [\rho'; Q']$, *then there exists* P' *such that* $P \stackrel{\mu}{\Rightarrow} P'$ *and* $(\rho', P', Q') \in \mathcal{R}$.

We note $\gtrsim_{wb}$ *the largest wb-expansion.*

We omit ρ, writing $P \gtrsim_{wb} Q$ when ρ is obvious from the context. This typed expansion is coarser than its untyped variant, which is written $\gtrsim$ [19].

Wb-bisimulation, and wb-bisimilarity noted $\approx_{\text{wb}}$, are defined as wb-expansion by replacing $Q \xrightarrow{\widehat{\mu}} Q'$ with $Q \stackrel{\widehat{\mu}}{\Rightarrow} Q'$ in Definition 10. The untyped version is written $\approx$.

4 Encoding Terms and Values in AIπ

In this section, we describe the encoding of $\lambda^{\mathtt{ref}}$-terms into AIπ processes. This leads to a clean operational correspondence where the encoding of a term may perform a unique transition according to the case distinction of Lemma 1.

We define the encoding in Figure 3. The encoding $[\![M]\!]$ of a term M is defined as an abstraction $(p)\ P$, and we use the notation $[\![M]\!]_q$ (resp. $[\![V]\!]^{\mathtt{v}}_y$) to note $[\![M]\!]\lfloor q \rfloor$ (resp. $[\![V]\!]^{\mathtt{v}}\lfloor y \rfloor$). Intuitively, $[\![M]\!]_p$ is a computation that returns a value at p while $[\![V]\!]^{\mathtt{v}}_y$ is a value that can be accessed at y.

This encoding extends the one from [3] with the additional constructs for handling references. A reference is represented by an output transmitting the stored value. Access to a reference is performed by receiving that output and emitting it back (possibly updated).

To remove any ambiguity in the rules, we consider that M (appearing in $[\![MN]\!]$, $[\![\ell := M; N]\!]$ and $[\![VM]\!]$) cannot be a value. This distinction enables optimisations that intuitively remove some communications signaling the end of a subcomputation. This is reminiscent of the colon translation from [15], in the setting of continuation-passing style translations. For any reference-closed configurations $\langle h \mid M \rangle$, we have $p : \mathsf{o} \vdash_{\text{wb}} [\![\langle h \mid M \rangle]\!]_p$.

We extend the encoding to evaluation contexts as shown in Figure 4. When an evaluation context is applied to a non-value term, its encoding correspond to encode the context first and then apply it to the encoding of the term:

▶ **Lemma 11.** *For any* E M *such that* M *is not a value, we have* $[\![E[M]]\!]_p = [\![E]\!][\![M]\!]_p$.

Proof. We proceed by induction on E:

1. when $E = [\cdot]$, the result is immediate.
2. when $E = FN$, we have

$$[\![E[M]]\!]_p = (\boldsymbol{\nu} q)([\![F[M]]\!]_q \mid q(y).\,(\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\,\overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))).$$

Functions

$$[\![V]\!] \stackrel{\text{def}}{=} (p)\ \overline{p}(y)\ [\![V]\!]^{\mathrm{v}}_y \qquad [\![xV]\!] \stackrel{\text{def}}{=} (p)\ \overline{x}(z,q)\ ([\![V]\!]^{\mathrm{v}}_z \mid q \rhd p)$$

$$[\![(\lambda x.\, N)V]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} y, w)([\![\lambda x.\, N]\!]^{\mathrm{v}}_y \mid [\![V]\!]^{\mathrm{v}}_w \mid \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))$$

$$[\![VM]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid (\boldsymbol{\nu} r)([\![M]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

$$[\![MN]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![M]\!]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

Imperative constructs

$$[\![!\ell]\!] \stackrel{\text{def}}{=} (p)\ \ell(w).\, (\overline{\ell}(y)\ y \rhd w \mid \overline{p}(z)\ z \rhd w) \qquad [\![\ell \coloneqq V; N]\!] \stackrel{\text{def}}{=} (p)\ \ell(\_).\, (\overline{\ell}(y)\ [\![V]\!]^{\mathrm{v}}_y \mid [\![N]\!]_p)$$

$$[\![\ell \coloneqq M; N]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![M]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p))$$

$$[\![\texttt{new}\ \ell \coloneqq V\ \texttt{in}\ N]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![V]\!]_q \mid q(z).\, (\boldsymbol{\nu} \ell)(\overline{\ell}(y)\ y \rhd z \mid [\![N]\!]_p))$$

Configurations

$$[\![h]\!] \stackrel{\text{def}}{=} \prod_{\ell_0 \in \widetilde{\ell}} (\overline{\ell_0}(y)\ [\![h(\ell_0)]\!]^{\mathrm{v}}_y) \text{ with } \widetilde{\ell} = \mathrm{dom}(h) \qquad [\![\langle h \mid N \rangle]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} \widetilde{\ell})([\![h]\!] \mid [\![N]\!]_p)$$

where $[\![V]\!]^{\mathrm{v}}$ is defined as:

$$[\![\lambda x.\, N]\!]^{\mathrm{v}} \stackrel{\text{def}}{=} (y)\ !y(x, q).\, [\![N]\!]_q \qquad [\![x]\!]^{\mathrm{v}} \stackrel{\text{def}}{=} (y)\ y \rhd x$$

Figure 3 Encoding of terms and configurations into AIπ

$$[\![[\cdot]]\!] \stackrel{\text{def}}{=} [\cdot] \qquad [\![FN]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![F]\!]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

$$[\![\ell \coloneqq F; N]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} q)([\![F]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p))$$

$$[\![VF]\!] \stackrel{\text{def}}{=} (p)\ (\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid (\boldsymbol{\nu} r)([\![F]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)))$$

Figure 4 Encoding of evaluation contexts

By induction, we have that $[\![F[M]]\!]_q = [\![F]\!][[\![M]\!]]_q$ and thus:

$$\begin{aligned}[\![E[M]]\!]_p &= (\boldsymbol{\nu} q)([\![F]\!][[\![M]\!]]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))) \\ &= [\![E]\!][[\![M]\!]]_p\end{aligned}$$

3. when $E = VF$, we have

$$[\![E[M]]\!]_p = (\boldsymbol{\nu} y, r)([\![V]\!]^{\mathrm{v}}_y \mid [\![F[M]]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)).$$

By induction, we have that $[\![F[M]]\!]_r = [\![F]\!][[\![M]\!]]_r$.
So $[\![E[M]]\!]_p = (\boldsymbol{\nu} y, r)([\![V]\!]^{\mathrm{v}}_y \mid [\![F]\!][[\![M]\!]]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) = [\![E]\!][[\![M]\!]]_p$.

4. when $E = \ell := F; N$, then $[\![E[M]]\!]_p = (\boldsymbol{\nu} q)([\![F[M]]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p))$.
   By induction, we have that $[\![F[M]]\!]_q = [\![F]\!][[\![M]]\!]_q$.
   So $[\![E[M]]\!]_p = (\boldsymbol{\nu} q)([\![F]\!][[\![M]]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p)) = [\![E]\!][[\![M]]\!]_p$. ◀

Because we distinguish between values and non-values in the encoding, the previous lemma does not hold when M is a value. In that case, the encoding performs some "optimisations". To relate the two processes, we need that on the encoding, forwarders act like substitutions.

▶ **Lemma 12.** *We have*

1. $(\boldsymbol{\nu} x)([\![M]\!]_p \mid x \rhd y) \gtrsim [\![M\{y/x\}]\!]_p$
2. $(\boldsymbol{\nu} x)([\![V]\!]^{\mathrm{v}}_z \mid x \rhd y) \gtrsim [\![V\{y/x\}]\!]^{\mathrm{v}}_z$
3. $(\boldsymbol{\nu} p)([\![M]\!]_p \mid p \rhd q) \gtrsim [\![M]\!]_q$
4. $(\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid x \rhd y) \gtrsim [\![V]\!]^{\mathrm{v}}_x$

Lemma 12 is proved by induction on the encoding of M or V to prove the four properties in conjonction. Indeed, there are dependencies between these properties which prevent us from treating them separately. This result is proved for the optimised encoding of the plain call-by-value λ-calculus [3] and it extends to the additional constructs of $\lambda^{\mathtt{ref}}$. As a result of the optimisation, we have the following lemma when applying a context to a value:

▶ **Lemma 13.** *For any value V and context E, $[\![E]\!][[\![V]]\!]_p \gtrsim [\![E[V]]\!]_p$.*

Proof. We proceed by induction on E:

- when $E = [\cdot]$, this is trivial.
- When $E = (\lambda x.\, M)\ [\cdot]$, we have

$$\begin{aligned}[\![E]\!][[\![V]]\!]_p &= (\boldsymbol{\nu} y, r)([\![\lambda x.\, M]\!]^{\mathrm{v}}_y \mid [\![V]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) \\ &\to (\boldsymbol{\nu} y, w)([\![\lambda x.\, M]\!]^{\mathrm{v}}_y \mid [\![V]\!]^{\mathrm{v}}_w \mid \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) = [\![E[V]]\!]_p\end{aligned}$$

  As this transition is deterministic by construction, we obtain that $[\![E]\!][[\![V]]\!]_p \gtrsim [\![E[V]]\!]_p$. This also holds for the three following cases.
- When $E = x\ [\cdot]$, we have

$$\begin{aligned}[\![E]\!][[\![V]]\!]_p &= (\boldsymbol{\nu} y, r)([\![x]\!]^{\mathrm{v}}_y \mid [\![V]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p)) \\ &\to^2 (\boldsymbol{\nu} y, w, w', r')([\![x]\!]^{\mathrm{v}}_y \mid \overline{x}(w'', r'')\ (w'' \rhd w' \mid r'' \rhd r') \mid [\![V]\!]^{\mathrm{v}}_w \mid w' \rhd w \mid r' \rhd p) \\ &\gtrsim (\boldsymbol{\nu} y)[\![x]\!]^{\mathrm{v}}_y \mid \overline{x}(w'', r'')\ (\boldsymbol{\nu} w, w', r')(w'' \rhd w' \mid w' \rhd w \mid [\![V]\!]^{\mathrm{v}}_w \mid r'' \rhd r' \mid r' \rhd p) \\ &\gtrsim (\boldsymbol{\nu} w)(\overline{x}(w'', r'')\ ([\![V]\!]^{\mathrm{v}}_{w''} \mid r'' \rhd p)) = [\![xV]\!]_p\end{aligned}$$

- When $E = [\cdot]\ M$, we have

$$\begin{aligned}[\![E]\!][[\![V]]\!]_p &= (\boldsymbol{\nu} q)([\![V]\!]_q \mid q(y).\, (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))) \\ &\to (\boldsymbol{\nu} y)([\![V]\!]^{\mathrm{v}}_y \mid (\boldsymbol{\nu} r)([\![N]\!]_r \mid r(w).\, \overline{y}(w', r')\ (w' \rhd w \mid r' \rhd p))) = [\![xV]\!]_p\end{aligned}$$

- When $E = \ell := [\cdot]; M$, we have

$$\begin{aligned}[\![E]\!][[\![V]]\!]_p &= (\boldsymbol{\nu} q)([\![V]\!]_q \mid q(w).\, \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p)) \\ &\to (\boldsymbol{\nu} w)([\![V]\!]^{\mathrm{v}}_w \mid \ell(\_).\, (\overline{\ell}(y)\ y \rhd w \mid [\![N]\!]_p)) \\ &\gtrsim \ell(\_).\, (\overline{\ell}(y)\ (\boldsymbol{\nu} w)([\![V]\!]^{\mathrm{v}}_w \mid y \rhd w) \mid [\![N]\!]_p) \\ &\gtrsim \ell(\_).\, (\overline{\ell}(y)\ [\![V]\!]^{\mathrm{v}}_y \mid [\![N]\!]_p) = [\![E[V]]\!]_p\end{aligned}$$

- when $E = F\ M$ or $V\ F$ or $\ell := F; M$ with $F \neq [\cdot]$, then $F[V]$ is not a value, so $[\![E[V]]\!]_p = [\![E']\!][[\![F[M]]\!]]_p$ for some E' and the result follows by induction as $\gtrsim$ is a congruence. ◀

We can now establish operational correspondence.

▶ **Proposition 14** (Untyped Operational Correspondence). *For any M, h with $\mathrm{dom}(h) = \widetilde{\ell}$ and fresh q_0, $[\![\langle h \mid M\rangle]\!]_{q_0}$ has exactly one immediate transition, and exactly one of the following clauses holds:*

1. *$\langle h \mid M\rangle \to \langle h' \mid N\rangle$ and $[\![\langle h \mid M\rangle]\!]_{q_0} \xrightarrow{\tau} P$ with $P \gtrsim [\![\langle h' \mid N\rangle]\!]_{q_0}$*
2. *M is a value, $[\![\langle h \mid M\rangle]\!]_{q_0} \xrightarrow{\overline{q_0}(x_0)} P$ and $P = (\boldsymbol{\nu}\widetilde{\ell})([\![h]\!] \mid [\![M]\!]^{\mathtt{v}}_{x_0})$.*
3. *M is of the form $E_0[yV_0]$ for some E_0, y and V_0, and we have $[\![\langle h \mid M\rangle]\!]_{q_0} \xrightarrow{\overline{y}(x_0,p_0)} P$ with $P \gtrsim (\boldsymbol{\nu}\widetilde{\ell})([\![h]\!] \mid [\![V_0]\!]^{\mathtt{v}}_{x_0} \mid p_0(z).\,[\![E_0[z]]\!]_{q_0})$.*

In the first case, the τ transition is deterministic, so we can prove that the following holds: $[\![\langle h \mid M\rangle]\!]_{q_0} \gtrsim [\![\langle h' \mid N\rangle]\!]_{q_0}$.

Using the existing equivalences in AIπ from Section 3, we can prove that the encodings of two $\lambda^{\mathtt{ref}}$-terms are equivalent. The equivalence and preorder used here are finer than bisimilarity with divergence which is sound as stated in Section 5. These examples show how the standard theory of the π-calculus is enough to prove interesting properties of $\lambda^{\mathtt{ref}}$-terms. These properties do not require us to use the triples defined in Section 5 nor to take into account the possibility of deferred divergence.

▶ **Example 15** (Unused reference). For any reference ℓ, value V and any term M with $\ell \notin \mathrm{fr}(M)$, we have $[\![\mathtt{new}\ \ell \coloneqq V\ \mathtt{in}\ M]\!]_p \gtrsim [\![M]\!]_p$.

Proof. We write

$$\begin{aligned}
[\![\mathtt{new}\ \ell \coloneqq V\ \mathtt{in}\ M]\!]_p &\gtrsim [\![\langle \emptyset \mid \mathtt{new}\ \ell \coloneqq V\ \mathtt{in}\ M\rangle]\!]_p \\
&\gtrsim [\![\langle \ell = V \mid M\rangle]\!]_p && \text{by Proposition 14} \\
&\gtrsim (\boldsymbol{\nu}\ell)(\overline{\ell}(y)\ [\![V]\!]^{\mathtt{v}}_y) \mid [\![M]\!]_p \gtrsim [\![M]\!]_p
\end{aligned}$$

We use $\gtrsim$ to perform a deterministic reduction and then to use simple laws and remove inaccessible processes. This result can be extended in presence of store by congruence, giving $[\![\langle h \mid \mathtt{new}\ \ell \coloneqq V\ \mathtt{in}\ M\rangle]\!]_p \gtrsim [\![\langle h \mid M\rangle]\!]_p$. ◀

▶ **Example 16** (One-use context). Let $f_1 \stackrel{\mathrm{def}}{=} \lambda x.\,\mathtt{if}\ !\ell = \mathbf{tt}\ \mathtt{then}\ \ell \coloneqq \mathbf{ff}; \mathbf{tt}\ \mathtt{else}\ \mathbf{ff}$ and $f_2 \stackrel{\mathrm{def}}{=} \lambda x.\,\mathbf{tt}$ and $E = [\cdot]\ (x\lambda y.\,y)$. Then $[\![\mathtt{new}\ \ell \coloneqq \mathbf{tt}\ \mathtt{in}\ E[f_1]]\!]_p \approx [\![E[f_2]]\!]_p$

Proof. By Proposition 14 and congruence of $\gtrsim$ we have:

$$\begin{aligned}
[\![E[f_2]]\!]_p &\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![f_2 w]\!]_p) \\
&\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\mathbf{tt}]\!]_p)
\end{aligned}$$

Using standard laws for $\gtrsim$, we relate the encoding of the first program to the same process:

$$\begin{aligned}
[\![\mathtt{new}\ \ell \coloneqq \mathbf{tt}\ \mathtt{in}\ E[f_1]]\!]_p &\gtrsim (\boldsymbol{\nu}\ell)([\![\ell = \mathbf{tt}]\!] \mid \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![f_1 w]\!]_p)) \\
&\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\langle \ell = \mathbf{tt} \mid f_1 w\rangle]\!]_p) \\
&\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\langle \ell = \mathbf{ff} \mid \mathbf{tt}\rangle]\!]_p) \\
&\gtrsim \overline{x}(z,q)\ ([\![\lambda y.\,y]\!]^{\mathtt{v}}_z \mid q(w).\,[\![\mathbf{tt}]\!]_p)
\end{aligned}$$

Note that we are outside AIπ because we are using constants ($\mathbf{tt}$, $\mathbf{ff}$). Adapting our setting to simple types can be done by having sorts $\mathtt{F}, \mathtt{R}, \mathtt{C}$ and forwarders $\triangleright_{\mathtt{C}}, \triangleright_{\mathtt{F}}$ for each type T, the forwarders being defined inductively on T instead of being constants. ◀

$$[\![\widetilde{V}_i]\!]^{\sim}_{\widetilde{x}} \stackrel{\text{def}}{=} \prod_i [\![V_i]\!]^{\mathtt{v}}_{x_i} \qquad \text{with } \widetilde{x} = \widetilde{x_i}$$

$$[\![E_1 :: \cdots :: E_n]\!]^{\sim}_{\widetilde{pq}} \stackrel{\text{def}}{=} \prod_{i \le n} p_i(z).\, [\![E_i[z]]\!]_{q_i} \qquad \text{with } \widetilde{pq} = p_1, q_1, \ldots, p_n, q_n$$

$$[\![(\widetilde{V}_i, \sigma, h)]\!]^{\sim}_{\widetilde{x};\widetilde{pq}} \stackrel{\text{def}}{=} (\boldsymbol{\nu}\widetilde{\ell})([\![\widetilde{V}_i]\!]^{\sim}_{\widetilde{x}} \mid [\![\sigma]\!]^{\sim}_{\widetilde{pq}} \mid [\![h]\!]) \qquad \text{with } \widetilde{\ell} = \mathrm{dom}(h)$$

$$[\![(\widetilde{V}_i, \sigma, \langle h \mid M\rangle)]\!]^{\sim}_{\widetilde{x};q_0,\widetilde{pq}} \stackrel{\text{def}}{=} (\boldsymbol{\nu}\widetilde{\ell})([\![\widetilde{V}_i]\!]^{\sim}_{\widetilde{x}} \mid [\![\sigma]\!]^{\sim}_{\widetilde{pq}} \mid [\![h]\!] \mid [\![M]\!]_{q_0}) \qquad \text{with } \widetilde{\ell} = \mathrm{dom}(h)$$

Figure 5 Encoding for triples

5 A π-Calculus Characterisation of Contextual Equivalence in $\lambda^{\mathtt{ref}}$

We can now move on to show our full abstraction result. To do so, we first extend the encoding to the triples defined in Section 2. This leads to an operational correspondence similar to Proposition 14 but for triples (Theorem 17). However, thanks to triples, it is possible to state Theorem 17 without using explicit AIπ constructs, so that each transition relates the encoding of triples. The bisimulation with divergence can then be defined and shown fully abstractly using mainly the operational correspondence theorem.

We describe in Figure 5 the encoding of triples (Section 2). It builds on the encoding in Figure 3, with values being encoded as expected. To encode σ, every evaluation context E in σ is encoded as the process $p(z).\,[\![E[z]]\!]_q$ so that, intutively, $E[z]$ can be executed as soon as the input at p is triggered. Both $[\![(\widetilde{V}_i, \sigma, c)]\!]^{\sim}_{\widetilde{x};q_0,\widetilde{pq}}$ and $[\![(\widetilde{V}_i, \sigma, h)]\!]^{\sim}_{\widetilde{x};\widetilde{pq}}$ are typeable with stacks that we write $\rho_{q_0,\widetilde{pq}}$ and $\rho_{\widetilde{pq}}$ respectively.

▶ **Theorem 17** (Operational Correspondence)**.**
We relate transitions for the encoding of both kind of triples:
When $[\rho; [\![(\widetilde{V}_i, \sigma, c)]\!]^{\sim}_{\widetilde{x};q_0,\widetilde{pq}}] \xrightarrow{\mu} [\rho'; P]$ *with* $\widetilde{x} = \widetilde{x_i}$ *and* $\widetilde{pq} = p_1, q_1, \ldots, p_n, q_n$ *then:*

1. *either* $c \to c'$, $\mu = \tau$ *and* $P \gtrsim [\![(\widetilde{V}_i, \sigma, c')]\!]^{\sim}_{\widetilde{x};q_0,\widetilde{pq}}$
2. *or* $c = \langle h \mid V_0\rangle$, $\mu = \overline{q_0}(x_0)$ *and* $P \gtrsim [\![((\widetilde{V}_i, V_0), \sigma, h)]\!]_{x_0,\widetilde{x};\widetilde{pq}}$
3. *or* $c = \langle h \mid E_0[y\ V_0]\rangle$, $\mu = \overline{y}(x_0, p_0)$ *and* $P \gtrsim [\![((\widetilde{V}_i, V_0), E_0 :: \sigma, h)]\!]_{x_0,\widetilde{x};p_0,q_0,\widetilde{pq}}$

When $[\rho; [\![(\widetilde{V}_i, \sigma, h)]\!]^{\sim}_{\widetilde{x};\widetilde{pq}}] \xrightarrow{\mu} [\rho'; P]$ *with* $\widetilde{x} = \widetilde{x_i}$ *and* $\widetilde{pq} = p_1, q_1, \ldots, p_n, q_n$ *then:*

1. *either* $\mu = x_j(z, q_0)$ *and* $P \gtrsim [\![(\widetilde{V}_i, \sigma, \langle h \mid N\rangle)]\!]^{\sim}_{\widetilde{x};q_0,\widetilde{pq}}$ *with* $V_j z \succ N$.
2. *or* $\sigma = E_1 :: \sigma'$ *and* $\mu = p_1(z)$ *and* $P \gtrsim [\![(\widetilde{V}_i, \sigma', \langle h \mid E_1[z]\rangle)]\!]^{\sim}_{\widetilde{x};q_1,p_2,q_2,\ldots,p_n,q_n}$

The following result is useful for the full abstraction proof.

▶ **Corollary 18.** *If* $[\![(\widetilde{V}_i, \sigma, c)]\!]^{\sim}_{\widetilde{x};q_0,\widetilde{pq}} \Rightarrow P'$, *then there exists a configuration* c' *with* $c \Rightarrow c'$ *and* $P' \gtrsim [\![(\widetilde{V}_i, \sigma, c')]\!]^{\sim}_{\widetilde{x};q_0,\widetilde{pq}}$.

Note that we rely on untyped expansion, which does not make any assumption about sequentiality of processes, in Theorem 17 and Corollary 18. This shows the robustness of the encoding.

The following example shows that by contrast with the encoding of configurations, the τ transition in the first case of Theorem 17 is not deterministic.

▶ **Example 19.** $[\![(\{\lambda z.\, \ell \coloneqq z\}, \odot, \langle \ell = y \mid !\ell\rangle)]\!]_{x_1;q_0} \not\gtrsim [\![(\{\lambda z.\, \ell \coloneqq z\}, \odot, \langle \ell = y \mid y\rangle)]\!]_{x_1;q_0}$.

Indeed, as the π-calculus is concurrent, the encoding of $\lambda z.\, \ell := z$ may be executed to change the content of ℓ before the read is executed. However, we can recover this result by using $\gtrsim_{wb}$ which forbids the concurrent transitions. This makes $\gtrsim_{wb}$ useful to handle reductions for triples.

We now introduce the notion of divergence for π-terms. This leads to the definition of bisimulation with divergence which coarsens $\approx_{\mathrm{wb}}$ to account for divergent terms. The induced equivalence for $\lambda^{\mathtt{ref}}$-terms corresponds to nfb.

A *wb-typed set* is a set of pairs (ρ, P) with ρ clean and $\rho \vDash_{\mathrm{wb}} P$.

▶ **Definition 20** (π_{div}). *A wb-typed set S is π-divergent if whenever we have $(\rho, P) \in S$, then $\rho \neq \emptyset$ and for all μ, ρ', P' with $[\rho; P] \xrightarrow{\mu} [\rho'; P']$, we have $(\rho', P') \in S$.*

We write π_{div} for the largest π-divergent set.

Intuitively, the computation ends when the stack gets empty, meaning there is no pending continuation. Processes in π_{div} thus correspond to processes which cannot terminate, hence the name of π-divergence.

The following example shows that the divergence of a process depends on the stack used to type it.

▶ **Example 21.** Let us consider

$$P \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu} x)(p_1(z).\, \overline{x}(y, r_1)\ r_1 \rhd q_1 \mid p_2(z).\, \overline{q_2}(y)\ x(y', r).\, \overline{r}(z')\ \mathbf{0}),$$

$$\rho_1 = p_1 : \mathtt{i}, q_1 : \mathtt{o}, p_2 : \mathtt{i}, q_2 : \mathtt{o}, \qquad \rho_2 = p_2 : \mathtt{i}, q_2 : \mathtt{o}, p_1 : \mathtt{i}, q_1 : \mathtt{o}.$$

We have both $\rho_1 \vDash_{\mathrm{wb}} P$ and $\rho_2 \vDash_{\mathrm{wb}} P$.

$[\rho_1; P] \xrightarrow{p_1(z)} [\rho'; P']$ is the only typed-allowed transition and P' has no typed-allowed transition. Thus, $(\rho_1, P) \in \pi_{\mathrm{div}}$.

On the other hand, $[\rho_2; P] \xrightarrow{p_2(z)} \xrightarrow{\overline{q_2}(y)} \xrightarrow{p_1(z_0)} \xrightarrow{\tau} \xrightarrow{\tau} \xrightarrow{\overline{q_1}(z')} [\emptyset; P']$. So $(\rho_2, P) \notin \pi_{\mathrm{div}}$.

▶ **Definition 22.** *A wb-typed symmetric relation $\mathcal{R}$ is a* bisimulation with divergence *if there exists a π-divergent set S such that whenever we have $(\sigma, P, Q) \in \mathcal{R}$ and $[\sigma; P] \xrightarrow{\mu} [\sigma'; P']$, then one of the following holds:*

1. *there exists Q' with $Q \stackrel{\widehat{\mu}}{\Rightarrow} Q'$ and $(\sigma', P', Q') \in \mathcal{R}$;*
2. *μ is an output and $(\sigma', P') \in S$.*

We write $\approx_{\mathrm{div}}$ for the largest bisimulation with divergence. We write $P \approx^{\rho}_{\mathrm{div}} Q$ when $(\rho, P, Q) \in \approx_{\mathrm{div}}$.

Bisimilarity with divergence is coarser than wb-bisimilarity and thus also coarser than the untyped bisimilarity.

To establish soundness, we rely on Theorem 17 (operational correspondence). The set of triples whose encoding is π-diverging is itself diverging, so we can prove that the relation induced by $\approx_{\mathrm{div}}$ is a nfb.

▶ **Theorem 23** (Soundness). *If $[\![M]\!]_p \approx^{p:\mathtt{o}}_{\mathrm{div}} [\![N]\!]_p$, then $M \asymp N$.*

The completeness is proved by showing that the encoding of a divergent set is π-divergent up to $\gtrsim$ and then Property (1) from the introduction, namely that the encoding of a nfb is a bisimulation with divergence up to $\gtrsim$.

▶ **Theorem 24** (Completeness). *If $M \asymp N$, then $[\![M]\!]_p \approx^{p:\mathtt{o}}_{\mathrm{div}} [\![N]\!]_p$.*

6 Up-to Techniques for $\approx_{\text{div}}$ in AIπ, and Applications

Up-to techniques are defined as functions from relations to relations. The idea of up-to techniques is to weaken the requirement by applying the up-to technique to the relation $\mathcal{R}$ in clause 1 of Definition 22. Standard up-to techniques like up-to expansion or up-to context can be adapted to our typed setting as in [5].

Similar up-to techniques for sets can also be defined and exploited for π-divergent sets. In order for up-to context to be sound, we must forbid contexts where the hole is guarded by a replicated input. Indeed, replicated input may only be typed with an empty set so they cannot be diverging. This is similar to $\lambda^{\mathtt{ref}}$, where Ω is divergent but $\lambda x.\,\Omega$ is not.

Thanks to up-to techniques, to prove that two processes are equivalent, we can give $\mathcal{R}$, a bismulation with divergence up to using S, a π-divergent set up to. This is used in the equivalences proven in Section 6.2.

6.1 A new up-to technique for AIπ: up-to body

In AIπ, functions are encoded as a replicated process of the form $!x(y,p).\,T$, which we denote T^x. When identical calls result in the same value being sent, it creates copies of that value, leading to processes of the form $T^x \mid T^y \mid \ldots \mid T^z$, with $x, y, \ldots, z$ being all different names. This behaviour makes bisimulations infinite, as they would need to contain processes with an arbitrary number of processes in parallel, despite all of them sharing the same body. We introduce a new technique, up-to body, which allows us to remove these duplicated copies. Indeed, it is sound to keep only one copy when comparing processes: any discriminating interaction with the environment involving multiple copies can be mimicked by a similar interaction with only one copy. The up-to body technique is defined by the following rule:

$$\frac{(\rho,\ E[T_1^x],\ F[T_2^x]) \in \mathcal{R}}{(\rho,\ E[T_1^z \mid T_1^x],\ F[T_2^z \mid T_2^x]) \in \mathtt{body}(\mathcal{R})}\, x, z \notin \mathrm{n}(E) \cup \mathrm{n}(F) \cup \mathrm{fn}(T_1) \cup \mathrm{fn}(T_2)$$

Up-to body differs from up-to context because we keep the possibly different evaluation contexts, E and F. These contexts correspond to private resources shared among the copies. In our case, the private resource is the local store, but this technique can be exported to the plain π-calculus. The technique for sets is defined similarly, with $(\rho, E[T^z \mid T^x]) \in \mathtt{body}(S)$ whenever $(\rho, E[T^x]) \in S$ and $x, z \notin \mathrm{n}(E) \cup \mathrm{fn}(T)$.

Using up-to body, it is possible to prove the analog of (1) from Section 1 but using normal form bisimulations from [2] instead of the formulation we have given in Definition 5.

We now present a simple example that demonstrates the use of up-to body.

▶ **Example 25.** $\mathtt{new}\ \ell \coloneqq z\ \mathtt{in}\ \lambda x.\,\lambda y.\,!\ell \asymp \lambda x.\,\lambda y.\,z$

Proof. We reason using the soundness of the encoding, and define

$$\begin{aligned}\mathcal{R} = \{&(\rho_q, [\![\mathtt{new}\ \ell \coloneqq z\ \mathtt{in}\ \lambda x.\,\lambda y.\,!\ell]\!]_q, [\![\lambda x.\,\lambda y.\,z]\!]_q),\\ &(\emptyset, [\![(\lambda x.\,\lambda y.\,!\ell,), \odot, \ell = z]\!]_{x_0}, [\![(\lambda x.\,\lambda y.\,z,), \odot, \emptyset]\!]_{x_0})\\ &(\emptyset, [\![(\lambda x.\,\lambda y.\,!\ell, \lambda y.\,!\ell), \odot, \ell = z]\!]_{x_0,x_1}, [\![(\lambda x.\,\lambda y.\,z, \lambda y.\,z), \odot, \emptyset]\!]_{x_0,x_1})\}.\end{aligned}$$

We can show that $\mathcal{R}$ is a bisimulation with divergence up to $\gtrsim_{wb}$, context and body. The up-to body technique makes it possible to stop the relation after the third pair. ◀

6.2 Examples

We now present an example that involves the encoding of triples, but does not require us to take into account deferred divergences. To validate the laws below, we thus rely on $\approx_{\text{wb}}$ which is included in $\approx_{\text{div}}$ (Definition 22).

▶ **Example 26** (Optimised access)**.** Two consecutive read and/or write operations can be transformed into an equivalent single operation.

For any pair (f_1, f_2) from the following (() is the unique inhabitant of the unit type):

$$\begin{array}{ll}
\big(f_1 = \lambda().\, \ell \coloneqq !\ell, & f_2 = \lambda().\, ()\big) \\
\big(f_1 = \lambda n.\, \lambda m.\, \ell \coloneqq n; \ell \coloneqq m, & f_2 = \lambda n.\, \lambda m.\, \ell \coloneqq m\big) \\
\big(f_1 = \lambda n.\, \ell \coloneqq n; !\ell, & f_2 = \lambda n.\, \ell \coloneqq n; n\big) \\
\big(f_1 = \lambda().\, \texttt{let } x = !\ell \texttt{ in let } y = !\ell \texttt{ in } M, & f_2 = \lambda().\, \texttt{let } x = !\ell \texttt{ in } M\{x/y\}\big)
\end{array}$$

we have for any E, y, V_0, we have $[\![\texttt{new } \ell \coloneqq V_0 \texttt{ in } E[y\ f_1]]\!]_{q_1} \approx_{\text{wb}} [\![\texttt{new } \ell \coloneqq V_0 \texttt{ in } E[y\ f_2]]\!]_{q_1}$.

Proof. First, we have that $[\![((f_1,), E :: \odot, \ell \coloneqq V_0)]\!]_{x;p_1,q_1} \approx_{\text{wb}} [\![((f_2,), E :: \odot, \ell \coloneqq V_0)]\!]_{x;p_1,q_1}$ using $(f_i,)$ to denote the singleton containing f_i. Indeed, we define a relation $\mathcal{R}$ as follows:

$$\mathcal{R} = \{(\rho_{\widetilde{pq}},\ [\![((\widetilde{V_i}, f_1), \sigma, h \uplus \ell = V)]\!]_{\widetilde{x};\widetilde{pq}}, [\![((\widetilde{V_i}, f_2), \sigma, h \uplus \ell = V)]\!]_{\widetilde{x};\widetilde{pq}}) \mid \text{ for all } \widetilde{V_i}, \sigma, h, V, \widetilde{x}, \widetilde{pq}\}$$

and we show that $\mathcal{R}$ is a *wb*-bisimulation up to $\gtrsim_{wb}$ and evaluation context.

Then we use Theorem 17 to show

$$[\![\texttt{new } \ell \coloneqq V_0 \texttt{ in } E[x\ f_i]]\!]_{q_1} \gtrsim [\![(\emptyset, \odot, \langle \ell = V_0 \mid E[y\ f_i]\rangle)]\!]_{q_1} \xrightarrow{\overline{y}(x,p_1)} \gtrsim [\![(f_i,), E :: \odot, \ell = V_0)]\!]_{x;p_1,q_1}$$

with the output being the only transition that the intermediate process can perform. ◀

The proof of this law could become even simpler by adopting the type system of [4]: we could prove prove directly that $[\![f_1]\!]_p$ and $[\![f_2]\!]_p$ are equivalent.

Relation $\mathcal{R}$ above would have the same base if we were to reason in the source language. If we were to show $\texttt{new } \ell \coloneqq V \texttt{ in } E[y\ f_1] \asymp \texttt{new } \ell \coloneqq V \texttt{ in } E[y\ f_2]$ using nfb, we would need to add triples with the same normal form term on both sides.

We present some examples involving deferred divergence from the literature.

▶ **Example 27.** $\langle \emptyset \mid x\ V\ \Omega\rangle \asymp \langle \emptyset \mid \Omega\rangle$, where V is a value and Ω is an always diverging term.

Proof. Take $\mathcal{R} = \{(\rho_q, [\![\langle \emptyset \mid x\ V\ \Omega\rangle]\!]_q, [\![\langle \emptyset \mid \Omega\rangle]\!]_q)\}$ and $S = \{(\rho_r, [\![\Omega]\!]_r)\}$.

S is π-divergent, so $\gtrsim_{wb}\texttt{ctxt}(S)$ is too, where `ctxt` stands for the up-to context technique. Then we show that $\mathcal{R}$ is a bisimulation with divergence up to context with $\gtrsim_{wb}\texttt{ctxt}(S)$ as the π-divergent set.

$[\rho_q; [\![\langle \emptyset \mid x\ V\ \Omega\rangle]\!]_q] \xrightarrow{\overline{x}(y,p)} [\rho_{p,q}; P]$ is the only type-allowed transition.

By Theorem 17, we know that $P \gtrsim [\![(\{V\}, [\cdot]\ \Omega :: \odot, \emptyset)]\!]_{y;pq}$.

As $(\rho_{p,q}, [\![(\{V\}, [\cdot]\ \Omega :: \odot, \emptyset)]\!]_{y;pq}) \in \texttt{ctxt}(S)$, we indeed have $(\rho_{p,q}, P) \in \gtrsim_{wb}\texttt{ctxt}(S)$. ◀

▶ **Example 28** (Example 9 from [2])**.**

$$V_1 = \lambda x.\, \texttt{if } !\ell \texttt{ then } \Omega \texttt{ else } k \coloneqq \mathbf{tt} \qquad W_1 = \lambda x.\, \Omega$$

$$V_2 = \lambda f.\, f\ V_1; \texttt{if } !k \texttt{ then } \Omega \texttt{ else } \ell \coloneqq \mathbf{tt} \qquad W_2 = \lambda f.\, f\ W_1$$

We have $\texttt{new } \ell \coloneqq \mathbf{ff} \texttt{ in new } k \coloneqq \mathbf{ff} \texttt{ in } V_2 \asymp W_2$.

Proof. Given a context E, we write E^n for $E :: E :: \cdots :: E :: \odot$ with n occurences of E.

Let $E \stackrel{\text{def}}{=} (\lambda().\,\texttt{if}\ !k\ \texttt{then}\ \Omega\ \texttt{else}\ \ell := \mathbf{tt})[\cdot]$, and $F \stackrel{\text{def}}{=} (\lambda().\,())[\cdot]$. We define $\mathcal{R}$ as

$$
\begin{array}{lll}
\{(\rho_q, & [\![\texttt{new}\ \ell := \mathbf{ff}\ \texttt{in new}\ k := \mathbf{ff}\ \texttt{in}\ V_2]\!]_q, & [\![W_2]\!]_q), \\
(\emptyset, & [\![(V_2,), \odot, \ell = \mathbf{ff} \uplus k = \mathbf{ff}]\!]_{x_2}, & [\![(W_2,), \odot, \emptyset]\!]_{x_2})\} \bigcup \\
\{(\rho_{\widetilde{pq}}, & [\![(V_1, V_2), E^n, \ell = \mathbf{ff} \uplus k = \mathbf{ff}]\!]_{\widetilde{x_i};\widetilde{pq}}, & [\![(W_1, W_2), F^n, \emptyset]\!]_{\widetilde{x_i};\widetilde{pq}}), \\
(\rho_{q_0,\widetilde{pq}}, & [\![(V_1, V_2), E^n, \langle \ell = \mathbf{ff} \uplus k = \mathbf{tt} \mid ()\rangle]\!]_{\widetilde{x_i};q_0,\widetilde{pq}}, & [\![(W_1, W_2), F^n, \langle \emptyset \mid \Omega\rangle]\!]_{\widetilde{x_i};q_0,\widetilde{pq}}), \\
(\rho_{\widetilde{pq}}, & [\![(V_1, V_2), E^n, \ell = \mathbf{tt} \uplus k = \mathbf{ff}]\!]_{\widetilde{x_i};\widetilde{pq}}, & [\![(W_1, W_2), F^n, \emptyset]\!]_{\widetilde{x_i};\widetilde{pq}}) \\
 & & \mid \text{ for all } \widetilde{pq}, n \text{ with } |\widetilde{pq}| = 2n\ \}
\end{array}
$$

and S as $\{(\rho_q, [\![\Omega]\!]_q),\ (\rho_{\widetilde{pq}}, [\![(V_1, V_2), E^n, \ell = \mathbf{ff} \uplus k = \mathbf{tt}]\!]_{\widetilde{x_i};\widetilde{pq}})$ for all $\widetilde{pq}, n$ with $|\widetilde{pq}| = 2n\}$.

$\mathcal{R}$ is a bisimulation up to $\gtrsim_{wb}$, context and body. Multiple calls to V_1, W_1 create multiples continuations, but thanks to up-to body, multiples calls to V_2, W_2 do not create multiples copies of V_1, W_1. All in all, we have the same number of pairs in the candidate bisimulation relation as in the relation in [2]. ◀

By modifying this example so as to allocate the references inside V_2, we get Example 15 from [8]. In that case, V_2 does not have free reference names. This makes it possible to use up-to parallel composition between the encoding of V_2 and V_1 preventing multiples calls to V_2. This usage of up-to parallel composition is similar to the up-to separation technique introduced in [8].

7 Related and Future Works

7.1 Related works

The equivalence in [8] is similar to nfb, for an extension of $\lambda^{\texttt{ref}}$. It is also fully abstract w.r.t. contextual equivalence. Up-to techniques for nfb are defined in [2], and used to prove several equivalences between $\lambda^{\texttt{ref}}$ programs. We can redo essentially the same proofs in our setting. Both works use techniques that are specific to nfb or its variant, and are arguably less standard than the up-to techniques we exploit in the π-calculus. In particular, the *up-to separation* from [8] is expressible using up-to context in the π-calculus.

A full abstraction result for PCF programs in AIπ is presented in [1], using a contextual equivalence in a typed setting. The type system captures closely the behaviour of the encoding of PCF terms, in the sense that any process whose type is the translation of a PCF type is behaviourally equivalent to the encoding of a source term, and in particular cannot be stateful. It seems difficult to find a labelled bisimilarity for this equivalence, and thus to use proof techniques as in our paper.

Our encoding is based on the one of [3], which has been used to show a close connection between operational game semantics and the π-calculus for call-by-value in [7]. Both works focus on this stateless calculus to show the correspondence with Lassen trees.

The type system of Section 3 is inspired by evaluation stacks used to ensure the bracketed condition in game semantics [9]. Game semantics can also be used to provide a fully abstract model for RefML, which is a language similar to $\lambda^{\texttt{ref}}$ [12, 13]. By being more operational, our approach is usable more directly to reason about $\lambda^{\texttt{ref}}$ programs. This is similar to operational game semantics which are complete for recursion-free programs with integer references [6].

7.2 Future works

The type system of [4] makes it possible to reason about references in a parallel setting. We believe that its addition to our type system would allow us to extend our full abstraction result to *open references*, i.e., references that can be returned by functions.

It would be interesting to see whether $\approx_{\text{div}}$ characterises a contextually-defined equivalence in AIπ. In comparison, such a result holds for $\approx_{\text{wb}}$ [5]: this equivalence corresponds to a typed barbed equivalence with a notion of *typed barb* to restrict the visibility of observables. One lead would be limiting barbs to be only outputs on continuation names. This may work only for the image of the encoding, but not for all typeable processes.

We would like to see whether the techniques we have developed can be exploited in other languages. Idealized ALGOL has both functional and imperative aspects, so our techniques may adapt to it. When extending $\lambda^{\mathtt{ref}}$ with control operators [20], well-bracketing is not required, so we can weaken the type system to simply ensure sequentiality. Because of the links with game semantics, object-oriented languages [14] can be interesting too.

References

1 Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the pi-calculus. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Proceedings*, volume 2044 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2001. doi:10.1007/3-540-45413-6_7.

2 Dariusz Biernacki, Sergueï Lenglet, and Piotr Polesiuk. A complete normal-form bisimilarity for state. In Mikolaj Bojanczyk and Alex Simpson, editors, *Foundations of Software Science and Computation Structures - 22nd International Conference, FOSSACS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11425 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2019. doi:10.1007/978-3-030-17127-8_6.

3 Adrien Durier, Daniel Hirschkoff, and Davide Sangiorgi. Eager Functions as Processes (long version). working paper or preprint, December 2021. URL: https://hal.archives-ouvertes.fr/hal-03466150.

4 Daniel Hirschkoff, Enguerrand Prebet, and Davide Sangiorgi. On the representation of references in the pi-calculus. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020*, volume 171 of *LIPIcs*, pages 34:1–34:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.34.

5 Daniel Hirschkoff, Enguerrand Prebet, and Davide Sangiorgi. On sequentiality and well-bracketing in the π-calculus. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470559.

6 Guilhem Jaber. Syteci: automating contextual equivalence for higher-order programs with references. *Proc. ACM Program. Lang.*, 4(POPL):59:1–59:28, 2020. doi:10.1145/3371127.

7 Guilhem Jaber and Davide Sangiorgi. Games, mobile processes, and functions. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*., Göttingen, Germany, February 2022. URL: https://hal.archives-ouvertes.fr/hal-03407123.

8 Vasileios Koutavas, Yu-Yang Lin, and Nikos Tzevelekos. From bounded checking to verification of equivalence via symbolic up-to techniques. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part II*, volume 13244 of *Lecture Notes in Computer Science*, pages 178–195. Springer, 2022. doi:10.1007/978-3-030-99527-0_10.

9 James Laird. A fully abstract trace semantics for general references. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *Automata, Languages and Programming, 34th International Colloquium, ICALP 2007, Wroclaw, Poland, July 9-13, 2007, Proceedings*, volume 4596 of *Lecture Notes in Computer Science*, pages 667–679. Springer, 2007. doi:10.1007/978-3-540-73420-8_58.

10 Jean-Marie Madiot, Damien Pous, and Davide Sangiorgi. Bisimulations up-to: Beyond first-order transition systems. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 93–108. Springer, 2014. doi:10.1007/978-3-662-44584-6_8.

11 Robin Milner. Functions as processes. *Math. Struct. Comput. Sci.*, 2(2):119–141, 1992. doi:10.1017/S0960129500001407.

12 Andrzej S. Murawski and Nikos Tzevelekos. Game semantics for good general references. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 75–84. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.31.

13 Andrzej S. Murawski and Nikos Tzevelekos. Algorithmic games for full ground references. *Formal Methods Syst. Des.*, 52(3):277–314, 2018. doi:10.1007/s10703-017-0292-9.

14 Andrzej S. Murawski and Nikos Tzevelekos. Game Semantics for Interface Middleweight Java. *J. ACM*, 68(1):4:1–4:51, 2021. doi:10.1145/3428676.

15 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

16 Damien Pous. Coinduction all the way up. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 307–316. ACM, 2016. doi:10.1145/2933575.2934564.

17 Davide Sangiorgi. Locality and interleaving semantics in calculi for mobile processes. *Theor. Comput. Sci.*, 155(1):39–83, 1996. doi:10.1016/0304-3975(95)00020-8.

18 Davide Sangiorgi. Lazy functions and mobile processes. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 691–720. The MIT Press, 2000.

19 Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

20 Kristian Støvring and Søren B. Lassen. A complete, co-inductive syntactic theory of sequential control and state. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 161–172. ACM, 2007. doi:10.1145/1190216.1190244.

21 Bernardo Toninho and Nobuko Yoshida. On polymorphic sessions and functions: A tale of two (fully abstract) encodings. *ACM Trans. Program. Lang. Syst.*, 43(2):7:1–7:55, 2021. doi:10.1145/3457884.