Trees from Functions as Processes

Davide Sangiorgi, Xian Xu

Trees from Functions as Processes

Davide Sangiorgi[1] and Xian Xu[2]

[1] University of Bologna/INRIA
[2] East China University of Science and Technology

Abstract. Lévy-Longo Trees and Böhm Trees are the best known tree structures on the λ-calculus. We give general conditions under which an encoding of the λ-calculus into the π-calculus is sound and complete with respect to such trees. We apply these conditions to various encodings of the call-by-name λ-calculus, showing how the two kinds of tree can be obtained by varying the behavioural equivalence adopted in the π-calculus and/or the encoding. The conditions are presented in the π-calculus but can be adapted to other concurrency formalisms.

1 Introduction

The π-calculus is a well-known model of computation with processes. Since its introduction, its comparison with the λ-calculus has received a lot of attention. Indeed, a deep comparison between a process calculus and the λ-calculus is interesting for several reasons: it is a significant test of expressiveness, and helps in getting deeper insight into its theory. From the λ-calculus perspective, it provides the means to study λ-terms in contexts other than purely sequential ones, and with the instruments available in the process calculus. A more practical motivations for describing functions as processes is to provide a semantic foundation for languages which combine concurrent and functional programming and to develop parallel implementations of functional languages.

Beginning with Milner's seminal work [8], a number of encodings λ-calculus strategies have been encoded into the π-calculus, including call-by-name, strong call-by-name (and call-by-need variants), call-by-value, parallel call-by-value (see [12, Chapter 15]). In each case, several variant encodings have appeared, by varying the target language or details of the encoding itself. Usually, when an encoding is given, a few basic results about its correctness are established, such as operational correctness and validity of reduction (i.e., the property that the encoding of a λ-term and of a derivative of its are behaviourally undistinguishable). Only in a few cases the question of the equality on λ-terms induced by the encoding has been tackled, e.g., [3–5, 11, 12]. In this paper, we refer to this question as the *full abstraction* issue: for an encoding $[\![\ ]\!]$ of the λ-calculus into π-calculus, an equality $=_\lambda$ on the λ-terms, and an equality $=_\pi$ on the π-terms, full abstraction is achieved when for all λ-terms M, N we have $M =_\lambda N$ iff $[\![M]\!] =_\pi [\![N]\!]$. Full abstraction has two parts: soundness, which is the implication from right to left, and completeness, which is its converse.

The equality $=_\lambda$ usually is not the ordinary Morris-style contextual equivalence on the λ-terms: the π-calculus is richer — and hence more discriminating — than

the λ-calculus; the latter is purely sequential, whereas the former can also express parallelism and non-determinism. (Exception to this are encodings into forms of π-calculus equipped with rigid constraints, e.g., typing constraints, which limit the set of legal π-calculus contexts.)

Indeed, the interesting question is understanding what $=_\lambda$ is when $=_\pi$ is a well-know behavioural equivalence on π-terms. This question essentially amounts to using the encoding in order to build a λ-model, and then understanding the λ-model itself. While seldomly tackled, the outcomes of this study have been significant: for a few call-by-name encodings it has been shown that, taking (weak) bisimulation on the π-terms, then $=_\lambda$ corresponds to a well-known tree structure in the λ-calculus theory, namely the *Lévy-Longo Trees* (LTs) [12].

There is however another kind of tree structure in the λ-calculus, even more important: the *Böhm Trees* (BTs). BTs play a central role in the classical theory of the λ-calculus. The local structure of some of the most influential models of the λ-calculus, like Scott and Plotkin's P_ω [13], Plotkin's T^ω [10], is precisely the BT equality; and the local structure of Scott's D_∞ (historically the first mathematical, i.e., non-syntactical, model of the untyped λ-calculus) is the equality of the 'infinite η contraction' of BTs. The full abstraction results in the literature for encodings of λ-calculus into π-calculus, however, only concern LTs.

A major reason for the limited attention that the full abstraction issue for encodings of λ-calculus into π-calculus has received is that understanding what kind of the structure the encoding produces may be difficult, and the full abstraction proof itself is long and tedious. The contribution of this paper is twofold:

1. We present general conditions for soundness and completeness of an encoding of the λ-calculus with respect to both LTs *and* BTs. The conditions can be used both on coinductive equivalences such as bisimilarity, and on contextual equivalences such as may and must equivalences.
2. We show that by properly tuning the notion of observability and/or the details of the encoding it is possible to recover BTs in place of LTs.

Some conditions only concern the behavioural equivalence chosen for the π-calculus, and are independent of the encoding; a few conditions are purely syntactic (e.g., certain encoded contexts should be guarded); the only behavioural conditions are equality of β-convertible terms, equality among certain unsolvable terms, and existence of an inverse for certain contexts resulting from the encoding (i.e., the possibility of extracting their immediate subterms, up-to the behavioural equivalence chosen in the π-calculus). We use these properties to derive full abstraction results for BTs and LTs for various encodings and various behavioural equivalence of the π-calculus. For this we exploit a few basic properties of the encodings, making a large reuse of proofs.

In the paper we use the conditions with the π-calculus, but they could also be used in other concurrency formalisms.

Structure of the paper. Section 2 collects background material. Section 3 introduces the notion of encoding of the λ-calculus, and concepts related to this. Section 4 presents the conditions for soundness and completeness. Section 5 applies the conditions on a few encodings of call-by-name and strong call-by-name from

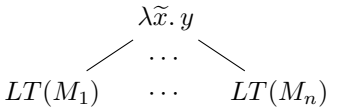
the literature, and for various behavioural equivalences on the π-calculus. Section 7 briefly discusses refinements of the π-calculus, notably with linear types. Some conclusions are reported in Section 8.

2 Background

The λ-calculus We use M, N to range over the set Λ of λ-terms, and x, y, z to range over variables. The standard syntax of λ-terms, and the rules for call-by-name and strong call-by-name (where reduction may continue underneath a λ-abstraction), are recalled in Appendix A. We assume the standard concepts of free and bound variables and substitutions, and identify α-convertible terms. We write Ω for the divergent term $(\lambda x.\, xx)(\lambda x.\, xx)$. Intuitively, a term M has *order of unsolvability* n ($0 \leqslant n < \omega$) if it behaves like Ω after n initial abstractions; M has *order of unsolvability* ∞ if it can reduce to an unbounded number of nested abstractions; M is *solvable* otherwise, with a head normal form of the shape $\lambda\widetilde{x}.\, yM_1 \dots M_n$.

Definition 1 (Lévy-Longo trees and Böhm trees). *The* Lévy–Longo Tree *of $M \in \Lambda$ is the labelled tree, LT(M), defined coinductively as follows:*

1. *$LT(M) = \top$ if M is an unsolvable of order ∞;*
2. *$LT(M) = \lambda x_1 \dots x_n.\, \bot$ if M is an unsolvable of order n;*
3. *$LT(M) =$*



if M has head normal form $\lambda\widetilde{x}.\, yM_1 \dots M_n$, $n \geqslant 0$.

Two terms M, N have the same LT if $LT(M) = LT(N)$. The definition of Böhm trees (BTs) is obtained from that of LTs using BT in place of LT in the definition above, and demanding that $BT(M) = \bot$ whenever M is unsolvable (in place of clauses (1) and (2)).

See [6] for a thorough tutorial on observational equivalences for such trees.

The (asynchronous) π-calculus We first consider encodings into the *asynchronous* π-calculus because its theory is simpler and because it is the usual target language for encodings of the λ-calculus. In all encodings we consider, the encoding of a λ-term is parametric on a name, that is, is a function from names to π-calculus processes. We call such expressions *abstractions*. For the purposes of this paper unary abstractions, i.e., with only one parameter, suffice. The actual instantiation of the parameter of an abstraction F is done via the *application* construct $F\langle a\rangle$. We use P, Q for process, F for abstractions. Processes and abstractions form the set of *π-agents* (or simply *agents*), ranged over by A. Small letters $a, b, \dots, x, y, \dots$ range over the infinite set of names. We use a tilde to indicate tuples; and given a

tuple $\widetilde{t}$, we write t_i for the i-th component of the tuple. Substitutions are ranged over by σ. The grammar of the calculus is thus:

$$\begin{array}{lll} A := P \mid F & & \text{(agents)} \\ P := \mathbf{0} \mid a(\widetilde{b}).\, P \mid \overline{a}\langle\widetilde{b}\rangle \mid P_1 \mid P_2 \mid \boldsymbol{\nu} a\, P \mid\ !a(\widetilde{b}).\, P \mid F\langle a\rangle & & \text{(processes)} \\ F := (a)\, P & & \text{(abstractions)} \end{array}$$

Since the calculus is polyadic, we assume a *sorting system* [9] to avoid disagreements in the arities of the tuples of names carried by a given name. We will not present the sorting system because not essential. The reader should take for granted that all agents described obey a sorting. A *context* C of π is a π-agent in which some subterms have been replaced by the hole $[\cdot]$ or, if the context is polyadic, with indexed holes $[\cdot]_1, \ldots, [\cdot]_n$; then $C[A]$ or $C[\widetilde{A}]$ is the agent resulting from replacing the holes with the terms A or $\widetilde{A}$. If the initial expression was an abstraction, we call the context an *abstraction π-context*; otherwise it is a *process π-context*. (A hole itself may stand for an abstraction or a process.) A name in a statement is *fresh* if it does not occur in the objects of the statement.

The operational semantics of the π-calculus is recalled in Appendix B. Transitions are of the form $P \xrightarrow{a(\widetilde{b})} P$ (an input, $\widetilde{b}$ are the bound names of the input prefix that has been fired), $P \xrightarrow{\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle} P'$ (an output, where $\widetilde{d} \subseteq \widetilde{b}$ are private names extruded in the output), and $P \xrightarrow{\tau} P'$ (an internal action). We use μ to range over the labels of transitions. We write $\Longrightarrow$ for the reflexive transitive closure of $\xrightarrow{\tau}$, and $\stackrel{\mu}{\Longrightarrow}$ for $\Longrightarrow\xrightarrow{\mu}\Longrightarrow$; then $\stackrel{\widehat{\mu}}{\Longrightarrow}$ is $\stackrel{\mu}{\Longrightarrow}$ if μ is not τ, and $\Longrightarrow$ otherwise; finally $P \xrightarrow{\widehat{\mu}} P'$ holds if $P \xrightarrow{\mu} P'$ or ($\mu = \tau$ and $P = P'$). In bisimulations or similar coinductive relations for the asynchronous π-calculus, no name instantiation is required in the input clause or elsewhere (provided α-convertible processes are identified); i.e., the *ground* versions of the relations are congruences or precongruences [12].

Definition 2 (bisimilarity). Bisimilarity *is the largest symmetric relation $\approx$ on π-processes such that whenever $P \approx Q$ and $P \xrightarrow{\mu} P'$ then $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and $P \approx Q'$.*

A key preorder in our work will be *expansion* [1, 12]; this is a refinement of bisimulation that takes into account the number of internal actions in simulation. Intuitively, Q expands P if they are weak bisimilar and moreover Q has no fewer internal actions when simulating P.

Definition 3 (expansion relation). *A relation $\mathcal{R}$ on π-processes is an* expansion relation *if whenever $P\ \mathcal{R}\ Q$:*

- *if $P \xrightarrow{\mu} P'$ then $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P'\ \mathcal{R}\ Q'$;*
- *if $Q \xrightarrow{\mu} Q'$ then $P \xrightarrow{\widehat{\mu}} P'$ and $P'\ \mathcal{R}\ Q'$;*

We write $\preccurlyeq$ for the largest expansion relation, and simply call it *expansion*. We also need the 'divergence-sensitive' variant of expansion, written $\preccurlyeq^{\Uparrow}$, as an auxiliary relation when dealing with must equivalences. Using $\Uparrow$ to indicate divergence (i.e., $P \Uparrow$ if P can undergo an infinite sequence of τ transitions), then $\preccurlyeq^{\Uparrow}$

is obtained by adding into Definition 3 the requirement that $Q \Uparrow$ implies $P \Uparrow$. We write $\succcurlyeq$ and ${}^{\Uparrow}\succcurlyeq$ for the inverse of $\preccurlyeq$ and $\preccurlyeq^{\Uparrow}$, respectively. The predicate $\Downarrow$ indicates barb-observability (i.e., $P \Downarrow$ if $P \Longrightarrow \xrightarrow{\mu}$ for some μ other than τ). As instance of a contextual divergence-sensitive equivalence, we consider *must-termination*, because of the simplicity of its definition — other choices would have been possible.

Definition 4 (may and must equivalences). *The π-processes P and Q are* may equivalent*, written $P \sim_{\mathrm{may}} Q$, if in all process contexts C we have $C[P] \Downarrow$ iff $C[Q] \Downarrow$. They are* must-termination equivalent *(briefly* must equivalent*), written $P \sim_{\mathrm{must}} Q$, if in all process contexts C we have $C[P] \Uparrow$ iff $C[Q] \Uparrow$.*

The behavioural relations defined above make use of the standard observables of the π-calculus; the relations can be made coarser by using the observables of asynchronous calculi, where one takes into account that, since outputs are not blocking, only output transitions from tested processes are immediately detected by an observer. In our examples, the option of asynchronous observable will make a difference only in the case of may equivalence. In *asynchronous may equivalence*, $\sim^{\mathrm{asy}}_{\mathrm{may}}$, the barb-observability predicate $\Downarrow$ is replaced by the asynchronous barb-observability predicate $\Downarrow_{\mathrm{asy}}$, whereby $P \Downarrow_{\mathrm{asy}}$ holds if $P \Longrightarrow \xrightarrow{\mu}$ and μ is an output action. We have $\preccurlyeq \subseteq \approx \subseteq \sim_{\mathrm{may}} \subseteq \sim^{\mathrm{asy}}_{\mathrm{may}}$, and $\preccurlyeq^{\Uparrow} \subseteq \sim_{\mathrm{must}}$. The following results will be useful later. A process is *inactive* if it may never perform a visible action.

Lemma 1. *For all process context C, we have:.*

1. *if P is inactive, then $C[P] \Downarrow$ implies $C[Q] \Downarrow$, and $C[P] \Downarrow_{\mathrm{asy}}$ implies $C[Q] \Downarrow_{\mathrm{asy}}$, and $C[a(\widetilde{x}).\, P] \Downarrow_{\mathrm{asy}}$ implies $C[P] \Downarrow_{\mathrm{asy}}$;*
2. *if $P \Uparrow$ then $C[Q] \Uparrow$ implies $C[P] \Uparrow$.*

Lemma 2. $\boldsymbol{\nu} a\, (\overline{a}\langle \widetilde{b} \rangle \mid a(\widetilde{x}).\, P) \; {}^{\Uparrow}\succcurlyeq P\{\widetilde{b}/\widetilde{x}\}$.

3 Encodings of the λ-calculus and full abstraction

In this paper an 'encoding of the λ-calculus into π-calculus' is supposed to be *compositional* (a mapping to π-calculus agents defined structurally on λ-terms), and *uniform*. The 'uniformity' condition refers to the treatment of the free variables: if the λ-term M and M' are the same modulo a renaming of free variables, then also their encodings should be same modulo a renaming of free names; since, in our encodings, λ-variables are included in the set of π-calculus names, a way of ensuring uniformity is to require that the encoding commutes with substitution, i.e., $[\![M\sigma]\!] \equiv [\![M]\!]\sigma$.

A compositional encoding can be extended to contexts. We will sometimes use:

- $C^{x}_{\lambda} \stackrel{\mathrm{def}}{=} [\![\lambda x.\, [\cdot]]\!]$, an *abstraction contexts of* $[\![\,]\!]$ (the hole represents the body of an abstraction);
- $C^{x,n}_{\mathrm{var}} \stackrel{\mathrm{def}}{=} [\![x[\cdot]_1 \cdots [\cdot]_n]\!]$ (for $n \geqslant 0$), a *variable contexts of* $[\![\,]\!]$ (an application context in which the head is a variable and the holes represent the following sequence of terms).

In the remainder of the paper, 'encoding' refers to a 'compositional and uniform encoding of the λ-calculus into the π-calculus'.

Definition 5 (soundness, completeness, full abstraction, validity of β rule). *An encoding $[\![\,]\!]$ and a relation $\mathcal{R}$ on π-agents are*

- sound for LTs *if $[\![M]\!]\ \mathcal{R}\ [\![N]\!]$ implies $LT(M) = LT(N)$, for all $M, N \in \Lambda$;*
- complete for LTSs *if $LT(M) = LT(N)$ implies $[\![M]\!]\ \mathcal{R}\ [\![N]\!]$, for all $M, N \in \Lambda$;*
- fully abstract for LTs *if they are both sound and complete for LTs.*

The same definitions will also be applied to BTs — just replace 'LT' with 'BT'.
Moreover, $[\![\,]\!]$ and $\mathcal{R}$ validate rule β *if $[\![(\lambda x.\, M)N]\!]\ \mathcal{R}\ [\![M\{^N/_x\}]\!]$, for all x, M, N.*

4 Conditions for completeness and soundness

We first give the conditions for completeness of an encoding $[\![\,]\!]$ from the λ-calculus into π with respect to a relation $\asymp$ on π-agents; then those for soundness. In both cases, the conditions involve an auxiliary relation $\leq$ on on π-agents.

Completeness conditions. In the conditions for completeness the auxiliary precongruence $\leq$ is required so to validate an 'up-to $\leq$ and contexts' technique. Such technique is inspired by the the 'up-to expansion and contexts' technique for bisimulation [12], which allows us the following flexibility in the bisimulation game required on a candidate relation $\mathcal{R}$: given a pair of derivatives P and Q, it is not necessary that the pair (P, Q) itself be in $\mathcal{R}$, as in the ordinary definition of bisimulation; it is sufficient to find processes $\widetilde{P}, \widetilde{Q}$, and a context C such that $P \succcurlyeq C[\widetilde{P}]$, $Q \succcurlyeq C[\widetilde{Q}]$, and $\widetilde{P}\ \mathcal{R}\ \widetilde{Q}$; that is, we can manipulate the original derivatives in terms of $\preccurlyeq$ so to isolate a common context C; this context is removed and only the resulting processes $\widetilde{P}, \widetilde{Q}$ need to be in $\mathcal{R}$. In the technique, the expansion relation is important: replacing it with bisimilarity breaks correctness. Also, some care is necessary when a hole of the contexts occurs underneath an input prefix, in which case a closure under name substitutions is required. Below, the technique is formulated in an abstract manner, using generic relations $\asymp$ and $\leq$. In the encodings we shall examine, $\asymp$ will be any of the congruence relations in Section 2, whereas $\leq$ will always be the expansion relation (or its divergence-sensitive variant, when $\asymp$ is must equivalence).

Definition 6 (up-to-$\leq$-and-contexts technique). *Relation $\asymp$* validates the up-to-$\leq$-and-contexts technique *if for any symmetric relation $\mathcal{R}$ on π-processes we have $\mathcal{R} \subseteq \asymp$ whenever for any pair $(P, Q) \in \mathcal{R}$, if $P \xrightarrow{\mu} P'$ then $Q \stackrel{\widehat{\mu}}{\Longrightarrow} Q'$ and there are processes $\widetilde{P}, \widetilde{Q}$ and a context C such that $P' \geq C[\widetilde{P}]$, $Q' \geq C[\widetilde{Q}]$, and, if $n \geqslant 0$ is the length of the tuples $\widetilde{P}$ and $\widetilde{Q}$, at least one of the following statements is true, for each $i \leqslant n$:*

1. *$P_i \asymp Q_i$;*
2. *$P_i\ \mathcal{R}\ Q_i$ and, if $[\cdot]_i$ occurs under an input in C, also $P_i\sigma\ \mathcal{R}\ Q_i\sigma$ for all substitutions σ.*

Below is the core of the completeness conditions. Some of these conditions ((1)-(3)) only concern the chosen behavioural equivalence $\asymp$ and its auxiliary relation $\leq$, and are independent of the encoding; the most important condition is the validity of the up-to-$\leq$-and-contexts technique. Other conditions (such as (4)) are purely syntactic. The only behavioural conditions on the encoding are (5), (6) (plus (ii) in Theorem 1). They concern validity of β rule and equality of certain unsolvables — very basic requirements for the operational correctness of an encoding.

Definition 7. *Let $\asymp$ and $\leq$ be relations on π-agents such that:*

1. *$\asymp$ is a congruence and $\asymp \supseteq \geq$;*
2. *$\leq$ is an expansion relation and is a precongruence;*
3. *$\asymp$ validates the up-to-$\leq$-and-contexts technique.*

Now, an encoding $[\![\,]\!]$ of the λ-calculus into π-calculus is faithful for $\asymp$ under $\leq$ *if*

4. *the variable contexts of $[\![\,]\!]$ are guarded;*
5. *$[\![\,]\!]$ and $\geq$ validate rule β;*
6. *if M is an unsolvable of order 0 then $[\![M]\!] \asymp [\![\Omega]\!]$.*

Theorem 1 (completeness). *Let $[\![\,]\!]$ be an encoding of the λ-calculus into π-calculus, and $\asymp$ a relation on π-agents. Suppose there is relation $\leq$ on π-agents such that $[\![\,]\!]$ is faithful for $\asymp$ under $\leq$. We have:*

(i) if the abstraction contexts of $[\![\,]\!]$ are guarded, then $[\![\,]\!]$ and $\asymp$ are complete for LTs;
(ii) if $[\![M]\!] \asymp [\![\Omega]\!]$ whenever M is unsolvable of order ∞, then $[\![\,]\!]$ and $\asymp$ are complete for BTs.

The proofs for LTs and BTs are similar. In the proof for LTs, for instance, we consider the relation $\mathcal{R} \stackrel{\text{def}}{=} \{([\![M]\!], [\![N]\!]) \text{ s.t. } LT(M) = LT(N)\}$ and show that for each $([\![M]\!], [\![N]\!]) \in \mathcal{R}$ one of the following conditions is true, for some abstraction context C^x_λ, variable context $C^{x,n}_{\text{var}}$, and terms M_i, N_i:

(a) $[\![M]\!] \asymp [\![\Omega]\!]$ and $[\![N]\!] \asymp [\![\Omega]\!]$;
(b) $[\![M]\!] \geq C^x_\lambda[[\![M_1]\!]]$, $[\![N]\!] \geq C^x_\lambda[[\![N_1]\!]]$ and $([\![M_1]\!], [\![N_1]\!]) \in \mathcal{R}$;
(c) $[\![M]\!] \geq C^{x,n}_{\text{var}}[[\![M_1]\!], \ldots, [\![M_n]\!]]$, $[\![N]\!] \geq C^{x,n}_{\text{var}}[[\![N_1]\!], \ldots, [\![N_n]\!]]$ and $([\![N_i]\!], [\![N_i]\!]) \in \mathcal{R}$ for all i.

Here, (a) is used when M and N are unsolvable of order 0, by appealing to clause (6) of Definition 7. In the remaining cases we obtain (b) or (c), depending on the shape of the LT for M and N, and appealing to clause (5) of Definition 7. The crux of the proof is exploiting the property that $\asymp$ validates the up-to-$\leq$-and-contexts technique so to derive $\mathcal{R} \subseteq \asymp$. Intuitively, this is possible because the variable and abstraction contexts of $[\![\,]\!]$ are guarded, and therefore the first action from terms such as $C^x_\lambda[[\![M_1]\!]]$ and $C^{x,n}_{\text{var}}[[\![M_1]\!], \ldots, [\![M_n]\!]]$ only consumes the context, and because $\leq$ is an expansion relation (clause (2) of Definition 7). Note that condition (2) of Definition 6 requires closure under substitutions when a hole is underneath a prefix. In clause (c) above we can derive closure under substitutions

from $([\![N_i]\!], [\![N_i]\!]) \in \mathcal{R}$ because the LT equality is preserved by variable renaming and because we assume an encoding to act uniformly on the free names (Section 3).

In the results for BTs, the condition on abstraction contexts being guarded is not needed because the condition can be proved redundant in presence of the condition in the assertion (ii) of the theorem. Intuitively, the reason is that, if in a term the head reduction never unveils a variable, then the term is unsolvable and can be equated to Ω using condition (ii); if it does unveil a variable, then in the encoding the subterms following the variable are underneath at least one prefix (because the variable contexts of the encoding are guarded, by condition (4)) and then we are able to apply a reasoning similar to that in clause (c) above. Also, we do not need to explicitly prove $[\![\lambda x.\, \Omega]\!] \asymp [\![\Omega]\!]$, this can be derived from condition (ii) and clauses (5) and (6) of Definition 7.

Soundness conditions In the conditions for soundness, one of the key requirements will be that certain contexts have an *inverse*. This intuitively means that it is possible to extract any of the processes in the holes of the context, up to the chosen behavioural equivalence. To have some more flexibility, we allow the appearance of the process of a hole after a rendez-vous with the external observer. This allows us to: initially restrict some names that are used to consume the context; then export such names before revealing the process of the hole. The reason why the restriction followed by the export of these names is useful is that the names might occur in the process of the hole; initially restricting them allows us to hide the names to the external environment; exporting them allows to remove the restrictions once the inversion work on the context is completed. The drawback of this initial rendez-vous is that we have to require a prefix-cancellation property on the behavioural equivalence; however, the requirement is straightforward to check in common behavioural equivalences.

We give the definition of inversion only for abstraction π-contexts whose holes are themselves abstractions. We only need this form of contexts when reasoning on λ-calculus encodings.

Definition 8. *Let C be an abstraction π-context with n holes, each occurring exactly once, each hole itself standing for an abstraction. We say that C* has inverse *with respect to a relation $\mathcal{R}$ on π-agents, if for every $i = 1, \ldots, n$ and for every $\widetilde{A}$ there exists a process π-context D_i and fresh names a, z, b such that*

$$D_i[C[\widetilde{A}]]\ \mathcal{R}\ (\boldsymbol{\nu}\widetilde{b}\,)(\overline{a}\langle\widetilde{c}\rangle \mid b(z).\, A_i\langle z\rangle)\ , \qquad \text{for } b \in \widetilde{b} \subseteq \widetilde{c}.$$

It is useful to establish inverse properties for contexts for the finest possible behavioural relation, so to export the result to coarser relations. In our work, the finest such relation is the divergence-sensitive expansion ($\preccurlyeq^{\Uparrow}$).

Example 1. We show examples of inversion using contexts that are similar to some abstraction and variable contexts in encodings of λ-calculus.

1. Consider a context $C \stackrel{\text{def}}{=} (p)\ p(x,q).\, ([\cdot]\langle q\rangle)$. If F fills the context, then an inverse for ${}^{\Uparrow}\!\succcurlyeq$ is the context

$$D \stackrel{\text{def}}{=} \boldsymbol{\nu} b\, (\overline{a}\langle b\rangle \mid b(r).\, \boldsymbol{\nu} p\, ([\cdot]\langle p\rangle \mid \overline{p}\langle x, r\rangle))$$

where all names are fresh (i.e., not free in F). Indeed we have, using simple algebraic manipulations (such as the law of Lemma 2):

$$\begin{aligned} D[C[F]] &\Uparrow\succcurlyeq \boldsymbol{\nu} b\,(\overline{a}\langle b\rangle \mid b(r).\,\boldsymbol{\nu} p\,(p(x,q).\,F\langle q\rangle \mid \overline{p}\langle x,r\rangle)) \\ &\Uparrow\succcurlyeq \boldsymbol{\nu} b\,(\overline{a}\langle b\rangle \mid b(r).\,F\langle r\rangle) \end{aligned}$$

2. Consider now a context $C \stackrel{\text{def}}{=} (p)\,(\boldsymbol{\nu} r, y\,)(\overline{x}\langle r\rangle \mid \overline{r}\langle y,p\rangle \mid !y(q).\,[\cdot]\langle q\rangle)$. If F fills the hole, then an inverse context is

$$D \stackrel{\text{def}}{=} ((\boldsymbol{\nu} x,p,b\,)([\cdot]\langle p\rangle \mid x(r).\,r(y,z).\,(\overline{a}\langle x,b\rangle \mid b(u).\,\overline{y}\langle u\rangle)) \qquad (1)$$

where again all names are fresh with respect to F. We have:

$$\begin{aligned} D[C[F]] \;=\; & (\boldsymbol{\nu} x,p,b\,)((C[F])\langle p\rangle \mid x(r).\,r(y,z).\,(\overline{a}\langle x,b\rangle \mid b(u).\,\overline{y}\langle u\rangle)) \\ \Uparrow\succcurlyeq\; & (\boldsymbol{\nu} x,p,b\,)(\boldsymbol{\nu} r,y\,)(\overline{x}\langle r\rangle \mid \overline{r}\langle y,p\rangle \mid !y(q).\,F\langle q\rangle) \mid x(r).\,r(y,z).\,(\overline{a}\langle x,b\rangle \mid b(u).\,\overline{y}\langle u\rangle) \\ \Uparrow\succcurlyeq\; & (\boldsymbol{\nu} x,b)(\boldsymbol{\nu} y\,(!y(q).\,F\langle q\rangle \mid (\overline{a}\langle x,b\rangle \mid b(u).\,\overline{y}\langle u\rangle))) \\ \Uparrow\succcurlyeq\; & (\boldsymbol{\nu} x,b)(\overline{a}\langle x,b\rangle \mid b(r).\,(\boldsymbol{\nu} y\,(!y(q).\,F\langle q\rangle \mid \overline{y}\langle r\rangle))) \\ \Uparrow\succcurlyeq\; & (\boldsymbol{\nu} x,b)(\overline{a}\langle x,b\rangle \mid b(r).\,F\langle r\rangle) \end{aligned}$$

Definition 9. *A relation $\mathcal{R}$ on π-agents* has the rendez-vous cancellation *property if whenever $\boldsymbol{\nu}\widetilde{b}\,(\overline{a}\langle\widetilde{c}\rangle \mid b(r).\,P)\;\mathcal{R}\;\boldsymbol{\nu}\widetilde{b}\,(\overline{a}\langle\widetilde{c}\rangle \mid b(r).\,Q)$ where $b \in \widetilde{b} \subseteq \widetilde{c}$ and a, b are fresh, then also $P\,\mathcal{R}\,Q$.*

The cancellation property is straightforward for a behavioural relation $\asymp$ because, in the initial processes, the output $\overline{a}\langle\widetilde{c}\rangle$ is the only possible initial action, after which the input at b must fire (the assumption 'a, b fresh' facilitates matters, though it is not essential).

As for completeness, so for soundness we isolate the common conditions for LTs and BTs. Besides the conditions on inverse of contexts, the other main requirement is about the inequality among some structurally different λ-terms (condition 6).

Definition 10. *Let $\asymp$ and $\leq$ be relations on π-agents where*

1. *$\asymp$ is a congruence, $\leq$ a precongruence,*
2. *$\asymp \supseteq \geq$;*
3. *$\asymp$ has the rendez-vous cancellation property.*

An encoding $[\![\,]\!]$ of the λ-calculus into π-calculus is respectful for $\asymp$ under $\leq$ *if*

4. *$[\![\,]\!]$ and $\geq$ validate rule β;*
5. *if M is an unsolvable of order 0, then $[\![M]\!] \asymp [\![\Omega]\!]$;*
6. *the terms $[\![\Omega]\!]$, $[\![x\widetilde{M}]\!]$, $[\![x\widetilde{M'}]\!]$, and $[\![y\widetilde{M''}]\!]$ are pairwise unrelated by $\asymp$, assuming that $x \neq y$ and that tuples $\widetilde{M}$ and $\widetilde{M'}$ have different lengths;*
7. *the abstraction and variable contexts of $[\![\,]\!]$ have inverse with respect to $\geq$.*

The condition on variable context having an inverse is the most delicate one. In the encodings of the π-calculus we have examined, however, the condition is simple to achieve.

Theorem 2 (soundness). *Let $[\![\,]\!]$ be an encoding of the λ-calculus into π-calculus, and $\asymp$ a relation on π-agents. Suppose there is a relation $\leq$ on π-agents such that $[\![\,]\!]$ is is respectful for $\asymp$ under $\leq$. We have:*

1. *if, for any M, the term $[\![\lambda x.\, M]\!]$ is unrelated by $\asymp$ to $[\![\Omega]\!]$ and to any term of the form $[\![x\widetilde{M}]\!]$, then $[\![\,]\!]$ and $\asymp$ are sound for LTs;*
2. *if*
   (a) $[\![M]\!] \asymp [\![\Omega]\!]$ whenever M is unsolvable of order ∞,
   (b) M solvable implies that the term $[\![\lambda x.\, M]\!]$ is unrelated by $\asymp$ to $[\![\Omega]\!]$ and to any term of the form $[\![x\widetilde{M}]\!]$,
   then $[\![\,]\!]$ and $\asymp$ are sound for BTs.

For the proof of Theorem 2, we use a coinductive definition of LT and BT equality, as forms of bisimulation. Then we show that the relation $\{(M, N) \mid [\![M]\!] \asymp [\![N]\!]\}$ implies the corresponding tree equality. In the case of internal nodes of the trees, we exploit conditions such as (6) and (7) of Definition 10.

Full abstraction We put together Theorems 1 and 2.

Theorem 3. *Let $[\![\,]\!]$ be an encoding of the λ-calculus into π-calculus, $\asymp$ a congruence relation on π-agents. Suppose there is a precongruence $\leq$ on π-agents such that*

1. *$\leq$ is an expansion relation and $\asymp \supseteq \geq$;*
2. *$\asymp$ validates the up-to-$\leq$-and-contexts technique;*
3. *the variable contexts of $[\![\,]\!]$ are guarded;*
4. *the abstraction and variable contexts of $[\![\,]\!]$ have inverse with respect to $\geq$;*
5. *$[\![\,]\!]$ and $\geq$ validate rule β;*
6. *if M is an unsolvable of order 0 then $[\![M]\!] \asymp [\![\Omega]\!]$;*
7. *the terms $[\![\Omega]\!]$, $[\![x\widetilde{M}]\!]$, $[\![x\widetilde{M'}]\!]$, and $[\![y\widetilde{M''}]\!]$ are pairwise unrelated by $\asymp$, assuming that $x \neq y$ and that tuples $\widetilde{M}$ and $\widetilde{M'}$ have different lengths.*

We have:

(i) *if*
   (a) the abstraction contexts of $[\![\,]\!]$ are guarded, and
   (b) for any M the term $[\![\lambda x.\, M]\!]$ is unrelated by $\asymp$ to $[\![\Omega]\!]$ and to any term of the form $[\![x\widetilde{M}]\!]$,
   then $[\![\,]\!]$ and $\asymp$ are fully abstract for LTs;

(ii) *if*
   (a) M solvable implies that the term $[\![\lambda x.\, M]\!]$ is unrelated by $\asymp$ to $[\![\Omega]\!]$ and to any term of the form $[\![x\widetilde{M}]\!]$, and
   (b) $[\![M]\!] \asymp [\![\Omega]\!]$ whenever M is unsolvable of order ∞,
   then $[\![\,]\!]$ and $\asymp$ are fully abstract for BTs.

In Theorems 1(i) and 3(i) for LTs the abstraction contexts are required to be guarded. This is reasonable in encodings of strategies, such as call-by-name, where evaluation does not continue underneath a λ-abstraction, but it is too demanding when evaluation can go past a λ-abstraction, such as strong call-by-name. We therefore present also the following alternative condition:

$$M, N \text{ unsolvable of order } \infty \text{ implies } [\![M]\!] \asymp [\![N]\!]. \qquad (*)$$

Theorem 4. *Theorems 1(i) and 3(i) continue to hold when the condition that the abstraction contexts be guarded is replaced by* $(*)$ *above.*

5 Examples with call-by-name

In this section we apply the theorems on soundness and completeness in the previous section to two well-known encodings of call-by-name λ-calculus: the one in Figure 1.a is Milner's original encoding [8]. The one in Figure 1.b is a variant encoding in which a function communicates with its environment via a rendez-vous (request/answer) pattern. An advantage of this encoding is that it can be easily tuned to call-by-need, or even used in combination with call-by-value [12].

For each encoding we consider soundness and completeness with respect to four behavioural equivalences: bisimilarity ($\approx$), may ($\sim_{\mathrm{may}}$), must ($\sim_{\mathrm{must}}$), and asynchronous may ($\sim_{\mathrm{may}}^{\mathrm{asy}}$). The following lemma allows us to apply the up-to-$\leq$-and-contexts technique.

Lemma 3. *Relations* $\approx$, $\sim_{\mathrm{may}}$, *and* $\sim_{\mathrm{may}}^{\mathrm{asy}}$ *validate the up-to-*$\preccurlyeq$*-and-contexts technique; relation* $\sim_{\mathrm{must}}$ *validates the up-to-*$\preccurlyeq^{\Uparrow}$*-and-contexts technique.*

The result in Lemma 3 for bisimulation is from [12]. The proofs for the may equivalences follow the definitions of the equivalences, reasoning by induction on the number of steps required to bring out an observable. The proof for the must equivalence uses coinduction to reason on divergent paths. Both for the may and for the must equivalences, the role of expansion ($\preccurlyeq$) is similar to its role in the technique for bisimulation.

Theorem 5. *The encoding of Figure 1.a is fully abstract for LTs when the behavioural equivalence for* π*-calculus is* $\approx, \sim_{\mathrm{may}}$, *or* $\sim_{\mathrm{must}}$*; and fully abstract for BTs when the behavioural equivalence is* $\sim_{\mathrm{may}}^{\mathrm{asy}}$.

The encoding of Figure 1.b is fully abstract for LTs under any of the equivalences $\approx, \sim_{\mathrm{may}}$, $\sim_{\mathrm{must}}$, *or* $\sim_{\mathrm{may}}^{\mathrm{asy}}$.

As Lemma 3 brings up, in the proofs, the auxiliary relation for $\approx$, $\sim_{\mathrm{may}}$, and $\sim_{\mathrm{may}}^{\mathrm{asy}}$ is $\preccurlyeq$; for $\sim_{\mathrm{must}}$ it is $\preccurlyeq^{\Uparrow}$.

With Lemma 3 at hand, the proofs for the soundness and completeness statements are simple. Moreover, there is a large reuse of proofs and results. For instance, in the completeness results for LTs, we only have to check that: the variable and abstraction contexts of the encoding are guarded; β rule is validated; all unsolvable of order 0 are equated. The first check is straightforward and is done only once. For the β rule, it suffices to establish its validity for $\preccurlyeq^{\Uparrow}$, which is the finest among the behavioural relations considered; this is done using distributivity laws for private replications [12], which are valid for strong bisimilarity and hence for $\preccurlyeq^{\Uparrow}$, and the law of Lemma 2. Similarly, for the unsolvable terms of order 0 it suffices to prove that they are all 'purely divergent' (i.e., divergent and unable to even perform some visible action), which follows from the validity of the β rule for $\preccurlyeq^{\Uparrow}$.

$$
\begin{aligned}
[\![\lambda x.\, M]\!] &\stackrel{\text{def}}{=} (p)\, p(x,q).\, [\![M]\!]\langle q\rangle \\
[\![x]\!] &\stackrel{\text{def}}{=} (p)\, \overline{x}\langle p\rangle \\
[\![MN]\!] &\stackrel{\text{def}}{=} (p)\, (\boldsymbol{\nu} r, x\,)\Big([\![M]\!]\langle r\rangle \mid \overline{r}\langle x, p\rangle \mid \\
&\qquad\qquad !x(q).\, [\![N]\!]\langle q\rangle\Big) \qquad \text{(for } x \text{ fresh)}
\end{aligned}
$$

Fig. 1.a: Milner's encoding

$$
\begin{aligned}
[\![\lambda x.\, M]\!] &\stackrel{\text{def}}{=} (p)\, \boldsymbol{\nu} v\, (\overline{p}\langle v\rangle \mid\ v(x,q).\, [\![M]\!]\langle q\rangle) \\
[\![x]\!] &\stackrel{\text{def}}{=} (p)\, \overline{x}\langle p\rangle \\
[\![MN]\!] &\stackrel{\text{def}}{=} (p)\, \boldsymbol{\nu} r\, \Big([\![M]\!]\langle r\rangle \mid \\
&\qquad\qquad r(v).\, \boldsymbol{\nu} x\, (\overline{v}\langle x, p\rangle \mid \\
&\qquad\qquad !x(q).\, [\![N]\!]\langle q\rangle)\Big) \qquad \text{(for } x \text{ fresh)}
\end{aligned}
$$

Fig. 1.b: a variant encoding

Fig. 1: The two encodings of call-by-name

Having checked the conditions for completeness, the only two additional conditions needed for soundness for LTs are conditions (6) and (7) of Definition 10, where we have to prove that certain terms are unrelated and that certain contexts have an inverse. The non-equivalence of the terms in condition (6) can be established for the coarsest equivalences, namely $\sim_{\text{may}}^{\text{asy}}$ and $\sim_{\text{must}}$, and then exported to the other equivalences. It suffices to look at visible traces of length 1 at most, except for terms of the form $[\![x\widetilde{M}]\!]$ and $[\![x\widetilde{M'}]\!]$, when tuples $\widetilde{M}$ and $\widetilde{M'}$ have different lengths, in which case one reasons by induction on the shortest of the two tuples.

The most delicate point is the existence of an inverse for the abstraction and the variable contexts. This can be established for the finest equivalence ($\preccurlyeq^{\Uparrow}$), and then exported to coarser equivalences. The two constructions needed for this are similar to those examined in Example 1.

For Milner's encoding, in the case of $\sim_{\text{may}}^{\text{asy}}$, we actually obtain the BT equality. One may find this surprising at first: BTs are defined from weak head reduction, in which evaluation continues underneath a λ-abstraction; however Milner's encoding mimics the call-by-name strategy, where reduction stops when a λ-abstraction is uncovered. We obtain BTs with $\sim_{\text{may}}^{\text{asy}}$ by exploiting Lemma 1(1) as follows. The encoding of a term $\lambda x.\, M$ is $(p)\, p(x,q).\, [\![M]\!]\langle q\rangle$. In an asynchronous semantics, an input is not directly observable; with $\sim_{\text{may}}^{\text{asy}}$ an input prefix can actually be erased provided, intuitively, that an output is never liberated. We sketch the proof of $[\![M]\!] \sim_{\text{may}}^{\text{asy}} [\![\Omega]\!]$ whenever M is unsolvable of order ∞, as required in condition (ii) of Theorem 3. Consider a context C with $C[[\![M]\!]] \Downarrow$, and suppose the observable is reached after n internal reductions. Term M, as ∞-unsolvable, can be β-reduced to $M' \stackrel{\text{def}}{=} (\lambda x)^n.\, N$, for some N. By validity of β-rule for $\succcurlyeq$, also $C[[\![M']\!]] \Downarrow$ in at most n steps; hence the subterm $[\![N]\!]$ of $[\![M']\!]$ does not contribute to the observable, since the abstraction contexts of the encodings are guarded and M' has n initial abstractions. We thus derive $C[[\![(\lambda x)^n.\, \Omega]\!]] \Downarrow$ and then, by repeatedly applying the third statement of Lemma 1(1) (as Ω is inactive), also $C[[\![\Omega]\!]] \Downarrow$. (The converse implication is given by the first statement in Lemma 1(1).)

$$[\![\lambda x.\, M]\!] \stackrel{\text{def}}{=} (p)\,(\boldsymbol{\nu} x, q)(\overline{p}\langle x, q\rangle \mid [\![M]\!]\langle q\rangle) \qquad [\![x]\!] \stackrel{\text{def}}{=} (p)\,(x(p').\,(p' \rhd p))$$

$$[\![MN]\!] \stackrel{\text{def}}{=} (p)\,(\boldsymbol{\nu} q, r)([\![M]\!]\langle q\rangle \mid q(x, p').\,(p' \rhd p \mid\, !\overline{x}\langle r\rangle.\,[\![N]\!]\langle r\rangle)) \qquad \text{(for } x \text{ fresh)}$$

where $r \rhd q \stackrel{\text{def}}{=} r(y, h).\,\overline{q}\langle y, h\rangle$

Fig. 2: Encoding of strong call-by-name

6 An example with strong call-by-name

In this section we consider a different λ-calculus strategy, strong call-by-name, where the evaluation of a term may continue underneath a λ-abstraction. The main reason is that we wish to see the impact of this difference on the equivalences induced by the encodings. Intuitively, evaluation underneath a λ-abstraction is fundamental in the definition of BTs and therefore we expect that obtaining the BT equality will be easier. However, the LT equality will still be predominant: in BTs a λ-abstraction is sometimes unobservable, whereas in an encoding into π-calculus a λ-abstraction always introduces a few prefixes, which are observable in the most common behavioural equivalences.

The encoding of strong call-by-name, from [7], is in Figure 2. The encoding behaves similarly to that in Figure 1.b; reduction underneath a 'λ' is implemented by exploting special wire processes (such as $q \rhd p$). They allow us to split the body M of an abstraction from its head λx; then the wires make the liaison between the head and the body. It actually uses the *synchronous* π-calculus, because some of the output prefixes have a continuation. Therefore the encoding also offers us the possibility of discussing the portability of our conditions to the synchronous π-calculus. For this, the only point in which some care is needed is that in the synchronous π-calculus, bisimilarity and expansion need some closure under name substitutions, in the input clause (on the placeholder name of the input), and the outermost level (i.e., before the bisimulation or expansion game is started) to become congruence or precongruence relations. Name substitutions may be applied following the early, late or open styles. The move from a style to another one does not affect the results in terms of BTs and LTs in the paper. We omit the definitions, see e.g., [12].

In short, for any of the standard behavioural congruences and expansion precongruences of the synchronous π-calculus, the conditions concerning $\asymp$ and $\leq$ of the theorems in Section 4 remain valid. In Theorem 6 below, we continue to use the symbols $\approx$ and $\preccurlyeq$ for bisimilarity and expansion, assuming that these are bisimulation congruences and expansion precongruences in any of the common π-calculus styles (early, late, open). (Again, in the case of must equivalence the expansion preorder should be divergence sensitive.) The proof of Theorem 6 is similar to that of Theorem 5. The main difference is that, since in strong call-by-name the abstraction contexts are not guarded, we have to adopt the modification in one of the conditions for LTs suggested in Theorem 4. Moreover, for the proof of validity

of β rule for $\preccurlyeq$, we use the following law to reason about wire processes $r \rhd q$ (and similarly for $\preccurlyeq^{\Uparrow}$):

- $\boldsymbol{\nu} q\ (q \rhd p \mid P) \succcurlyeq P\{p/q\}$ provided p does not appear free in P, and q only appears free in P only once, in a subexpression of the form $\overline{q}\langle \widetilde{v} \rangle . \mathbf{0}$.

Theorem 6. *The encoding of Figure 2 is fully abstract for LTs when the behavioural equivalence for the π-calculus is $\approx, \sim_{\mathrm{may}}$, or $\sim_{\mathrm{may}}^{\mathrm{asy}}$; and fully abstract for BTs when the behavioural equivalence is $\sim_{\mathrm{must}}$.*

Thus we obtain the BT equality for the must equivalence. Indeed, under strong call-by-name, all unsolvable terms are divergent. In contrast with Milner's encoding of Figure 1.a, under asynchronous may equivalence we obtain LTs because in both encodings of strong call-by-name the first action of an abstraction is an output, rather than an input as in Milner's encoding, and outputs are observable in asynchronous equivalences.

7 Types and asynchrony

We show, using Milner's encoding (Figure 1.a), that we can sometimes switch from LTs to BTs by taking into account some simple *type* information together with asynchronous forms of behavioural equivalences. The type information needed is the linearity of the parameter name of the encoding (names p, q, r in Figure 1.a). Linearity ensures us that the external environment can never cause interferences along these names: if the input capability is used by the process encoding a λ-term, then the external environment cannot exercise the same (competing) capability. In an asynchronous behavioural equivalence input prefixes are not directly observable (as discussed earlier for asynchronous may).

Linear types and asynchrony can easily be incorporate in a bisimulation congruence by using a contextual form of bisimulation such as *barbed congruence* [12]. In this case, barbs (the observables of barbed congruence) are only produced by output prefixes (as in asynchronous may equivalence); and the contexts in which processes may be tested should respect the type information ascribed to processes (in particular the linearity mentioned earlier). We write $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$ for the resulting asynchronous typed barbed congruence. Using Theorem 3(ii) we obtain:

Theorem 7. *The encoding of Figure 1.a is fully abstract for BTs when the behavioural equivalence for the π-calculus is $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$.*

The auxiliary relation is still $\preccurlyeq$; here asynchrony and linearity are not needed.

8 Conclusions and future work

In this paper we have studied soundness and completeness conditions with respect to BTs and LTs for encodings of λ-calculus into the π-calculus. While the conditions have been presented on the π-calculus, they can be adapted to other concurrency formalisms. For instance, expansion, a key preorder in our conditions,

can always be extracted from bisimilarity as its "efficiency" preorder. It might be difficult, in contrast, to adapt our conditions to sequential languages; a delicate condition, for instance, appears to be the one on inversion of variable contexts.

We have used the conditions to derive tree characterisations for various encodings and various behavioural equivalences, including bisimilarity, may and must equivalences, and asynchronous variants of them (see a summary in Appendix C). The proofs of the conditions can often be transported from a behavioural equivalence to another one, with little or no extra work (e.g., exploiting containments among equivalences and preorders). Overall, we found the conditions particularly useful when dealing with contextual equivalences, such as may and must equivalences. It is unclear to us how soundness and completeness could be proved for them by relying on, e.g., direct characterisations of the equivalences (such as trace equivalence or forms of acceptance trees) and standard proof techniques for them.

It would be interesting to examine additional conditions on the behavioural equivalences of the π-calculus capable to retrieve, as equivalence induced by an encoding, that of BTs under η contraction, or BTs under infinite η contractions [2]. Works on linearity in the π-calculus, such as [14] might be useful.

In the paper we have considered encodings of call-by-name or strong call-by-name. These strategies fit the definition of BTs and LTs, in which reduction always picks the leftmost redex. We do not know, in contrast, what kind of tree structures could be obtained from encodings of the call-by-value strategy.

References

1. Arun-Kumar, S., Hennessy, M.: An efficiency preorder for processes. Acta Informatica 29, 737–760 (1992)
2. Barendregt, H.P.: The Lambda Calculus: Syntax, semantics. North-Holland (1984)
3. Berger, M., Honda, K., Yoshida, N.: Sequentiality and the pi-calculus. In: Proc. TLCA'01. pp. 29–45 (2001)
4. Berger, M., Honda, K., Yoshida, N.: Genericity and the pi-calculus. Acta Inf. 42(2-3), 83–141 (2005)
5. Demangeon, R., Honda, K.: Full abstraction in a subtyped pi-calculus with linear types. In: Proc. CONCUR'11. pp. 280–296. LNCS, Springer (2011)
6. Dezani-Ciancaglini, M., Giovannetti, E.: From Bohm's theorem to observational equivalences: an informal account. ENTCS. 50(2), 83–116 (2001)
7. Hirschkoff, D., Madiot, J.M., Sangiorgi, D.: Duality and i/o-types in the π-calculus. In: Proc. CONCUR'12. LNCS, vol. 7454, pp. 302–316. Springer (2012)
8. Milner, R.: Functions as processes. Mathematical Structures in Computer Science 2(2), 119–141 (1992), research Report 1154, INRIA, Sofia Antipolis, 1990
9. Milner, R.: Communicating and Mobile Systems: the π-Calculus. CUP (1999)
10. Plotkin, G.D.: T^ω as a universal domain. Jourbal of Computer and System Sciences 17, 209–236 (1978)
11. Sangiorgi, D.: Lazy functions and mobile processes. In: Proof, Language and Interaction: Essays in Honour of Robin Milner. MIT Press (2000)
12. Sangiorgi, D., Walker, D.: The π-calculus: a Theory of Mobile Processes. CUP (2001)
13. Scott, D.: Data types as lattices. SIAM Journal on Computing 5(3), 522–587 (1976)
14. Yoshida, N., Honda, K., Berger, M.: Linearity and bisimulation. J. Log. Algebr. Program. 72(2), 207–238 (2007)

Appendix

A λ-calculus: syntax and call-by-name rules

The set Λ of λ-terms is:

$$M ::= x \;\Big|\; \lambda x.\, M \;\Big|\; MN$$

We will encode call-by-name λ-calculus, in its weak or strong form. In both cases, we have rules β and μ, only in the strong case we have also ξ:

$$\frac{}{(\lambda x.\, M)M' \to M\{M'/x\}}\ \beta \qquad \frac{M \to M'}{\lambda x.\, M \to \lambda x.\, M'}\ \xi \qquad \frac{M \to M'}{MN \to M'N}\ \mu$$

B Operational semantics of the π-calculus

Below is the standard operational semantics for the asynchronous polyadic π-calculus (we assume that α-convertible terms are identified). We write $\mathrm{fn}(P)$ for the free names of a process P, and $\mathrm{bn}(\mu)$ for the bound names of action μ.

$$\texttt{inp}\colon a(\widetilde{b}).\, P \xrightarrow{a(\widetilde{b})} P \qquad \texttt{rep}\colon !a(\widetilde{b}).\, P \xrightarrow{a(\widetilde{b})} P \mid !a(\widetilde{b}).\, P \text{ , if } a \notin \widetilde{b}$$

$$\texttt{out}\colon \overline{a}\langle\widetilde{b}\rangle \xrightarrow{\overline{a}\langle\widetilde{b}\rangle} \mathbf{0} \qquad \texttt{par}\colon \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q} \text{ if } \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\texttt{com}\colon \frac{P \xrightarrow{a(\widetilde{c})} P' \quad Q \xrightarrow{\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu}\widetilde{d}\,(P'\{\widetilde{b}/\widetilde{c}\} \mid Q')} \text{ if } \widetilde{d} \cap \mathrm{fn}(P) = \emptyset$$

$$\texttt{res}\colon \frac{P \xrightarrow{\mu} P'}{\boldsymbol{\nu} a\, P \xrightarrow{\mu} \boldsymbol{\nu} a\, P'}\ a \text{ does not appear in } \mu$$

$$\texttt{open}\colon \frac{P \xrightarrow{\boldsymbol{\nu}\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle} P'}{\boldsymbol{\nu} c\, P \xrightarrow{\boldsymbol{\nu} c,\widetilde{d}\,\overline{a}\langle\widetilde{b}\rangle} P'}\ c \in \widetilde{b} - \widetilde{d},\ a \neq c.$$

$$\texttt{app}\colon \frac{P\{b/a\} \xrightarrow{\mu} P'}{F\langle b\rangle \xrightarrow{\mu} P'} \text{ if } F = (a)\, P$$

In the synchronous π-calculus, the rule for output prefix becomes

$$\texttt{out}\colon \overline{a}\langle\widetilde{b}\rangle.\, P \xrightarrow{\overline{a}\langle\widetilde{b}\rangle} P$$

C Summary of the results for the encoding examined

The tables below summarise the results with respect to BTs and LTs for the encodings and the behavioural equivalences examined in the paper. In a table, a checkmark means that corresponding the result holds; otherwise the result is false.

We recall that $\approx$ is weak bisimilarity; $\sim_{\text{may}}$ is may equivalence; $\sim^{\text{asy}}_{\text{may}}$ is asynchronous may equivalence; $\sim_{\text{must}}$ is must equivalence; and $\approx^{\text{lin,asy}}_{\text{bc}}$ is asynchronous barbed congruence with the linearity type constraints on the location names of the encoded λ-terms (i.e., the abstracted name in the encoding of a λ-term).

		$\approx$	$\sim_{\text{may}}$	$\sim^{\text{asy}}_{\text{may}}$	$\sim_{\text{must}}$	$\approx^{\text{lin,asy}}_{\text{bc}}$
LT	complete	✓	✓	✓	✓	✓
	sound	✓	✓		✓	
BT	complete			✓		✓
	sound	✓	✓	✓	✓	✓

Table 1.a: Results for Figure 1.a

		$\approx$	$\sim_{\text{may}}$	$\sim^{\text{asy}}_{\text{may}}$	$\sim_{\text{must}}$
LT	complete	✓	✓	✓	✓
	sound	✓	✓	✓	✓
BT	complete				
	sound	✓	✓	✓	✓

Table 1.b: Results for Figure 1.b

		$\approx$	$\sim_{\text{may}}$	$\sim^{\text{asy}}_{\text{may}}$	$\sim_{\text{must}}$
LT	complete	✓	✓	✓	✓
	sound	✓	✓	✓	
BT	complete				✓
	sound	✓	✓	✓	✓

Table 2: Results for Figure 2