Trees from functions as processes

Davide Sangiorgi, Xian Xu

TREES FROM FUNCTIONS AS PROCESSES

DAVIDE SANGIORGI AND XIAN XU

Università di Bologna (Italy) and INRIA (France)

East China University of Science and Technology (China)

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.

1. INTRODUCTION

The π-calculus is a well-known model of computation with processes. Since its intro-
duction, 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 [13], a number of 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 [18, Chapter 15]). In each case, several
variant encodings have appeared, by varying the target language or details of the encoding
itself, see [18, Part VI] for details. Usually, when an encoding is given, a few basic results
about its correctness are established, such as operational correctness and validity of reduc-
tion (i.e., the property that the encoding of a λ-term and the encoding of a reduct of it 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., [3,4,6,16–18]; in [17,18] for encodings

of call-by-name and with respect to the ordinary bisimilarity of the π-calculus, in [3, 4, 6] for various forms of λ-calculi, including polymorphic ones, and with respect to contextual forms of behavioural equivalence enhanced with types so to obtain coarser relations.

In this paper, we refer to the above 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 [3, 4, 6].

Indeed, the interesting question here is understanding what $=_\lambda$ is when $=_\pi$ is a well-known 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 seldom tackled, the outcomes of this study have been significant: for a few call-by-name encodings presented in [18] 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) [18].

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 [21], Plotkin's T^ω [15], 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 expansions' of BTs. Details on these and other models of the λ-calculus can be found in the comprehensive books [2, 10]. The full abstraction results in the literature for encodings of λ-calculus into π-calculus, however, only concern LTs [18].

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 [20].

(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 sub-terms, 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 potentially 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 and Section 6 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

2.1. **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 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 standard [2]. The set Λ of λ-terms is given by the grammar:

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

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:

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

We sometimes omit λ in nested abstractions, thus for example, $\lambda x_1 x_2.\, M$ stands for $\lambda x_1.\, \lambda x_2.\, M$. We assume the standard concepts of free and bound variables and substitutions, and identify α-convertible terms. Thus, throughout the paper '=' is syntactic equality modulo α-conversion. We write Ω for the purely divergent term $(\lambda x.\, xx)(\lambda x.\, xx)$. We sometimes use $\widetilde{\cdot}$ for a tuple of elements; for instance $\lambda \widetilde{x}.\, M$ stands for $\lambda x_1 \ldots x_n.\, M$ and $\widetilde{M}$ for $M_1 M_2 \cdots M_n$, for some n. We write $|\, \widetilde{e}\,|$ for the cardinality of the tuple $\widetilde{e}$, and $\widetilde{e}_i$ for the i-th component of the tuple.

In order to define Lévy-Longo trees and Böhm trees, we need the notions of *solvability*, and of *head reduction*, which we now introduce (see [7] for a thorough tutorial on such trees). We use n to range over the set of non-negative integers and ω to represent the first limit ordinal.

A λ-term is either of the form $\lambda \widetilde{x}.\, y\widetilde{M}$ or of the form $\lambda \widetilde{x}.\, (\lambda x.\, M_0)M_1 \ldots M_n$, $n \geqslant 1$. In the latter, the redex $(\lambda x.\, M_0)M_1$ is called the *head redex*. If M has a head redex, then $M \to_{\mathrm{h}} N$ holds if N results from M by β-reducing its head redex. *Head reduction*, $\Longrightarrow_{\mathrm{h}}$, is the reflexive and transitive closure of $\to_{\mathrm{h}}$. Head reduction is different from the call-by-name reduction ($\Longrightarrow$): a call-by-name redex is also a head redex, but the converse is false as a head redex can also be located underneath an abstraction. The terms of the form $\lambda \widetilde{x}.\, y\widetilde{M}$, that is, the terms that cannot be head-reduced, are the *head normal forms*. Since head reduction is deterministic, the head normal form for a term M, that is, a head normal form N such that $M \Longrightarrow_{\mathrm{h}} N$, if it exists, is unique. The terms that have head normal forms are the *solvable* terms. The remaining terms are called *unsolvable*. These are the terms in which head reduction never terminates. It may be however that head reductions

on an unsolvable term uncover some abstractions. The number of such abstraction defines the *order of unsolvability* for that term. Formally, an unsolvable term M has *order of unsolvability* n, for $0 \leq n < \omega$ if n is the largest integer such that $M \Longrightarrow_{\mathrm{h}} \lambda \widetilde{x}.\, M$, for some M and $\widetilde{x}$ with $\mid \widetilde{x} \mid = n$; the unsolvable M has *order of unsolvability* ω if for all $n \geq 0$ we have $M \Longrightarrow_{\mathrm{h}} \lambda \widetilde{x}.\, M$, for some M and $\widetilde{x}$ with $\mid \widetilde{x} \mid = n$. The unsolvable of order ω can produce unboundedly many abstractions while performing head reductions.

Definition 2.1 (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) $LT(M) = \lambda x_1 \ldots x_n.\, \bot$ if M is an unsolvable of order n;
  (3) $LT(M) = $ tree with $\lambda \widetilde{x}.\, y$ as the root and $LT(M_1)$,...,$LT(M_n)$ as the children, if M has head normal form $\lambda \widetilde{x}.\, y M_1 \ldots M_n$, $n \geqslant 0$. That is,
  $LT(M) = $

$$
\begin{array}{c}
\lambda \widetilde{x}.\, y \\
\diagup \quad \cdots \quad \diagdown \\
LT(M_1) \quad \cdots \quad LT(M_n)
\end{array}
$$

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. That is, clauses (1) and (2) are replaced by the following one:

$$BT(M) = \bot \qquad \text{if } M \text{ is unsolvable}$$

2.2. **The (asynchronous) π-calculus.** We first consider encodings into the *asynchronous* π-calculus because its theory is simpler than that of the synchronous π-calculus (notably bisimulation does not require closure under name instantiations and has sharper congruence properties [5]) 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 $a, b, \ldots, x, y, \ldots$ range over the infinite set of names. Substitutions, ranged over by σ, act on names; for instance $\{\widetilde{c}/\widetilde{b}\}$ represents the substituting in which the i-th component of $\widetilde{b}$ is replaced by the i-th component of $\widetilde{c}$. The grammar of the calculus is thus:

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

Since the calculus is polyadic, we assume a *sorting system* [14] to avoid disagreements in the arities of the tuples of names carried by a given name. We will not present the sorting system because it is 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

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

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

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

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

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

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

FIGURE 1. Operational semantics of the π-calculus

$C[A]$ or $C[\widetilde{A}]$ is the agent resulting from replacing the holes with the terms A or $\widetilde{A}$. If the initial π-agent 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 context is *guarded* if the holes in it only appear underneath some prefix (input or output) [12, 18]; for example context $a(\widetilde{b}).\,(P \mid [\cdot])$ is guarded whereas $\boldsymbol{\nu}a\,(P \mid [\cdot])$ is not. A name is *fresh* if it does not occur in the objects under consideration. In a restriction $\boldsymbol{\nu}b\,P$, inputs $a(\widetilde{b}).\,P$ or $!a(\widetilde{b}).\,P$, and abstraction $(b)P$, names b and $\widetilde{b}$ are binders with scope P. As for the λ-calculus, we assume that α-convertible terms are identified.

The operational semantics of the asynchronous polyadic π-calculus is standard [18], and given in Figure 1. We write $\mathrm{fn}(P)$ for the free names of a process P, and $\mathrm{bn}(\mu)$ for the bound names of action μ.

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 $\xLongrightarrow{\mu}$ for $\Longrightarrow \xrightarrow{\mu} \Longrightarrow$; then $\xLongrightarrow{\widehat{\mu}}$ is $\xLongrightarrow{\mu}$ 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 bisimilarity or similar coinductive relations for the asynchronous π-calculus, no name instantiation is required in the input clause or elsewhere because such relations are already closed under name substitutions.

Definition 2.2 (bisimilarity). A symmetric relation $\mathcal{R}$ on π-processes is a *bisimulation*, if whenever $P \mathcal{R} Q$ and $P \xrightarrow{\mu} P'$, then $Q \xLongrightarrow{\widehat{\mu}} Q'$ for some Q' and $P' \mathcal{R} Q'$.

Processes P and Q are *bisimilar*, written $P \approx Q$, if $P \mathcal{R} Q$ for some bisimulation $\mathcal{R}$.

In a standard way, we can extend $\approx$ to abstractions: $F \approx G$ if $F\langle b \rangle \approx G\langle b \rangle$ for every b. As usual, strong bisimilarity, written $\sim$, is obtained by replacing $\stackrel{\widehat{\mu}}{\Longrightarrow}$ with $\stackrel{\mu}{\rightarrow}$ in the definition of weak bisimilarity,

A key preorder in our work will be *expansion* [1,18]; this is a refinement of bisimulation that takes into account the number of internal actions. Intuitively, Q expands P if they are weakly bisimilar and moreover Q has no fewer internal actions when simulating P.

Definition 2.3 (expansion relation). A relation $\mathcal{R}$ on π-processes is an *expansion relation* if whenever $P \mathcal{R} Q$:

(1) if $P \stackrel{\mu}{\rightarrow} P'$ then $Q \stackrel{\mu}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$;

(2) if $Q \stackrel{\mu}{\rightarrow} Q'$ then $P \stackrel{\widehat{\mu}}{\longrightarrow} P'$ and $P' \mathcal{R} Q'$.

We write $\preccurlyeq$ for the largest expansion relation, and call it *expansion*.

We also need the 'divergence-sensitive' variant of expansion, written $\preccurlyeq^{\Uparrow}$, as an auxiliary relation when tackling 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 2.3 the requirement that $Q\Uparrow$ implies $P\Uparrow$. We write $\succcurlyeq$ and $^{\Uparrow}\!\succcurlyeq$ for the inverse of $\preccurlyeq$ and $\preccurlyeq^{\Uparrow}$, respectively. 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. The predicate $\Downarrow$ indicates barb-observability, i.e., $P\Downarrow$ if $P \Longrightarrow \stackrel{\mu}{\rightarrow}$ for some μ other than τ.

Definition 2.4 (may and must equivalences). The π-processes P and Q are *may equivalent*, written $P \sim_{\text{may}} Q$, if in all process contexts C we have $C[P]\Downarrow$ iff $C[Q]\Downarrow$. They are *must-termination equivalent* (briefly *must equivalent*), written $P \sim_{\text{must}} Q$, if in all process contexts C we have $C[P]\Uparrow$ iff $C[Q]\Uparrow$.

The behavioural relations defined above use the standard observables of π-calculus; they 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*, $\sim_{\text{may}}^{\text{asy}}$, the barb-observability predicate $\Downarrow$ is replaced by the asynchronous barb-observability predicate $\Downarrow_{\text{asy}}$, whereby $P\Downarrow_{\text{asy}}$ holds if $P \Longrightarrow \stackrel{\mu}{\rightarrow}$ and μ is an output action. We have $\preccurlyeq \subseteq \approx \subseteq \sim_{\text{may}} \subseteq \sim_{\text{may}}^{\text{asy}}$, and $\preccurlyeq^{\Uparrow} \subseteq \sim_{\text{must}}$. The following results will be useful later. A process is *inactive* if it may never perform a visible action, i.e., an input or output; formally P is inactive if there is no P' such that $P \Longrightarrow \stackrel{\mu}{\rightarrow} P'$ and μ is an input or output.

Lemma 2.5. *For all process contexts C, we have:*

(1) *if P is inactive, then*
  - $C[P]\Downarrow$ *implies* $C[Q]\Downarrow$ *for all Q,*
  - $C[P]\Downarrow_{\text{asy}}$ *implies* $C[Q]\Downarrow_{\text{asy}}$ *for all Q,*
  - $C[a(\widetilde{x}).P]\Downarrow_{\text{asy}}$ *implies* $C[P]\Downarrow_{\text{asy}}$;

(2) *if $P\Uparrow$ then for all Q, $C[Q]\Uparrow$ implies $C[P]\Uparrow$.*

Lemma 2.6. $\nu a\,(\overline{a}\langle \widetilde{b} \rangle \mid a(\widetilde{x}).P) \;^{\Uparrow}\!\succcurlyeq\; P\{\widetilde{b}/\widetilde{x}\}$.

3. Encodings of the λ-calculus and full abstraction

To make the encodings more readable, we shall assume that λ-variables are included in the set of π-calculus names. In this paper, an 'encoding of the λ-calculus into π-calculus' is supposed to be *compositional* and *uniform*. Compositionality means that the definition of the encoding on a term should depend only upon the definition on the term's immediate constituents, following the grammar of the encoded language. In the specific case of an encoding $[\![\,]\!]$ of λ into π, this means that the encoding is defined thus:

$$
\begin{aligned}
[\![x]\!] &\stackrel{\text{def}}{=} T_x \\
[\![\lambda x.\, M]\!] &\stackrel{\text{def}}{=} C_\lambda^x[[\![M]\!]] \\
[\![MN]\!] &\stackrel{\text{def}}{=} C_{\text{app}}[[\![M]\!], [\![N]\!]]
\end{aligned}
$$

where T_x is a π-term that may contain free occurrence of x, C_λ^x is a π-context in which x only appears as a bound name with a scope that embraces the hole, and C_{app} is a two-hole π-context.

Uniformity 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 the corresponding free names. A way of ensuring this is to require that the encoding commutes with name substitution; i.e., if σ is a λ-calculus variable renaming (a substitution from variables to variables) that, since λ-variables are included in the set of π-calculus names, also represents a π-calculus substitution from names to names, then it holds that

$$[\![M\sigma]\!] = [\![M]\!]\sigma. \tag{3.1}$$

This condition, involving substitutions, is meaningful provided that the encoding respects α-conversion; that is, if M and N are α-convertible terms, then $[\![M]\!] = [\![N]\!]$. Moreover, if we take σ to mean a substitution acting on the set of π-calculus names (a superset of the set of λ-calculus variables), then (3.1) also says that the encoding does not introduce extra free names; that is, for any M, the free names of $[\![M]\!]$ are also free variables of M. Thus uniformity comes with condition (3.1), plus the these conditions on α-conversion and on free names.

A compositional encoding can be extended to contexts, by extending the encoding mapping a λ-calculus context into the corresponding π-calculus context. Two such contexts will be useful in this work, for a given encoding $[\![\,]\!]$:

(1) $C_\lambda^x \stackrel{\text{def}}{=} [\![\lambda x.\, [\cdot]]\!]$, called an *abstraction context of* $[\![\,]\!]$. We have already mentioned this encoding when describing the meaning of compositionality.

(2) $C_{\text{var}}^{x,n} \stackrel{\text{def}}{=} [\![x[\cdot]_1 \cdots [\cdot]_n]\!]$ (for $n \geqslant 0$), called a *variable context of* $[\![\,]\!]$. This context will be used to represent the encoding of terms of the form $xM_1 \cdots M_n$, for some $M_1 \cdots M_n$, as we have

$$[\![xM_1 \cdots M_n]\!] = C_{\text{var}}^{x,n}[[\![M_1]\!], \cdots, [\![M_n]\!]]$$

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

Definition 3.1 (soundness, completeness, full abstraction, validity of β rule). An encoding $[\![\,]\!]$ and a relation $\mathcal{R}$ on π-agents are:

(1) *sound for LTs* if $[\![M]\!] \,\mathcal{R}\, [\![N]\!]$ implies $LT(M) = LT(N)$, for all $M, N \in \Lambda$;

(2) *complete for LTs* if $LT(M) = LT(N)$ implies $[\![M]\!] \,\mathcal{R}\, [\![N]\!]$, for all $M, N \in \Lambda$;

(3) *fully abstract for LTs* if they are both sound and complete for LTs.

The same definitions are also 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 π-agents.

4.1. **Completeness conditions.** In the conditions for completeness the auxiliary precongruence $\leq$ is required to validate an 'up-to $\leq$ and contexts' technique. Such technique is inspired by the 'up-to expansion and contexts' technique for bisimulation [18], 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 4.1 (up-to-$\leq$-and-contexts technique)**.**
- A symmetric relation $\mathcal{R}$ on π-processes is an *up-to-$\leq$-and-contexts candidate for $\asymp$* if 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 two statements is true, for each $1 \leqslant i \leqslant n$:
  (1) $P_i \asymp Q_i$;
  (2) $P_i \,\mathcal{R}\, Q_i$ and, if $[\cdot]_i$ occurs underneath an input-prefix in C (that is, guarded by an input), also $P_i\sigma \,\mathcal{R}\, Q_i\sigma$ for all substitutions σ.
- Relation $\asymp$ *validates the up-to-$\leq$-and-contexts technique* if for any up-to-$\leq$-and-contexts candidate for $\asymp$ $\mathcal{R}$ we have $\mathcal{R} \subseteq \asymp$.

Below is the core of the completeness conditions (Definition 4.3). Some of these conditions ((1) to (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; we use the standard concept of *guarded context*, in which each hole appears underneath some prefix [12, 18]. The only behavioural conditions on the encoding are (5) and (6) in Definition 4.3, plus (ii) in Theorem 4.4. They concern validity of β rule and equality of certain unsolvables — very basic requirements for the operational correctness of an encoding.

We recall that a relation $\mathcal{R}$ in a language that is preserved by the constructs of the language is:

- a *precongruence* if $\mathcal{R}$ is a preorder relation;
- a *congruence* if $\mathcal{R}$ is an equivalence relation.

Note that for any abstraction $F \stackrel{\text{def}}{=} (a)P$, the terms $F\langle b \rangle$ and $P\{b/a\}$ have exactly the same transitions. Hence we expect any behavioural relation $\mathcal{R}$ to identify such processes, i.e., $F\langle b \rangle \; \mathcal{R} \; P\{b/a\}$ as well as $P\{b/a\} \; \mathcal{R} \; F\langle b \rangle$. We call *plain* a relation on processes in which this holds.

Lemma 4.2. *Let $\mathcal{R}$ be a plain precongruence on π-agents. We have:*

(1) *$\mathcal{R}$ is preserved by name substitutions, i.e., $P \; \mathcal{R} \; Q$ implies $P\{a/b\} \; \mathcal{R} \; Q\{a/b\}$, for all a, b;*

(2) *if $F \stackrel{\text{def}}{=} (a)P$ and $G \stackrel{\text{def}}{=} (a)Q$ then $F \; \mathcal{R} \; G$ implies $P \; \mathcal{R} \; Q$, and $F\langle z \rangle \; \mathcal{R} \; G\langle z \rangle$ implies $F \; \mathcal{R} \; G$, for any fresh name z.*

Proof. For the first item, from $P \; \mathcal{R} \; Q$, by the precongruence property we have $(a)P \; \mathcal{R} \; (a)Q$ and then also $((a)P)\langle b \rangle \; \mathcal{R} \; ((a)Q)\langle b \rangle$; since R is plain we conclude $P\{b/a\} \; \mathcal{R} \; Q\{b/a\}$.

For the second item, in the first case from $F \; \mathcal{R} \; G$ we derive $F\langle a \rangle \; \mathcal{R} \; G\langle a \rangle$, hence also $P\{a/a\} \; \mathcal{R} \; Q\{a/a\}$, which is $P \; \mathcal{R} \; Q$. The second case is similar: $F\langle z \rangle \; \mathcal{R} \; G\langle z \rangle$ implies $(z)(F\langle z \rangle) \; \mathcal{R} \; (z)(G\langle z \rangle)$ that, using the plain and precongruence properties, implies $(z)(P\{z/a\}) \; \mathcal{R} \; (z)(Q\{z/a\})$, which is the same as $F \; \mathcal{R} \; G$, since z is fresh and we identify α-convertible terms. $\qquad\square$

Definition 4.3. 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 plain precongruence;
(3) $\asymp$ validates the up-to-$\leq$-and-contexts technique.

Now, an encoding $[\![\,]\!]$ of 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 4.4 (completeness)**.** *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 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 proof of Theorem 4.4 is placed in Appendix A. We provide some intuitive account below. The proofs for LTs and BTs are similar. In the proof for LTs, for instance, we consider the relation

$$\mathcal{R} \stackrel{\text{def}}{=} \{([\![M]\!]\langle r \rangle, [\![N]\!]\langle r \rangle) \mid LT(M) = LT(N), \text{ and } r \text{ fresh}\}$$

and show that for each $([\![M]\!], [\![N]\!]) \in \mathcal{R}$ one of the following conditions is true, for some abstraction context C_λ^x, variable context $C_{\text{var}}^{x,n}$, and terms M_i, N_i:

(a) $[\![M]\!] \asymp [\![\Omega]\!]$ and $[\![N]\!] \asymp [\![\Omega]\!]$;
(b) $[\![M]\!] \geq C_\lambda^x[[\![M_1]\!]]$, $[\![N]\!] \geq C_\lambda^x[[\![N_1]\!]]$ and $([\![M_1]\!], [\![N_1]\!]) \in \mathcal{R}$;

(c) $[\![M]\!] \geq C_{\mathrm{var}}^{x,n}[\![M_1]\!], \ldots, [\![M_n]\!]], [\![N]\!] \geq C_{\mathrm{var}}^{x,n}[\![N_1]\!], \ldots, [\![N_n]\!]]$ and $([\![M_i]\!], [\![N_i]\!]) \in \mathcal{R}$ for all i.

Here, (a) is used when M and N are unsolvable of order 0, by appealing to clause (6) of Definition 4.3. 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 4.3. The crux of the proof is exploiting the property that $\asymp$ validates the up-to-$\leq$-and-contexts technique so to derive $\mathcal{R} \subseteq \asymp$ (i.e., the continuations of $[\![M]\!]$ and $[\![N]\!]$ are related, using expansion and cutting a common context). Intuitively, this is possible because the variable and abstraction contexts of $[\![\ ]\!]$ are guarded and because $\leq$ is an expansion relation (clause (2) of Definition 4.3). 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.

4.2. **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; that is, contexts that are obtained by a π-abstraction by replacing subterms that are themselves abstractions with holes. We only need this form of contexts when reasoning on λ-calculus encodings, and each hole of a context will be filled with the encoding of a λ-term.

Definition 4.5. 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 results onto coarser relations. In our work, the finest such relation is the divergence-sensitive expansion ($\preccurlyeq^{\Uparrow}$).

Example 4.6. 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. Indeed we have, using simple algebraic manipulations (such as the law of Lemma 2.6):

$$D[C[F]] \quad \Uparrow_{\succcurlyeq} \quad \boldsymbol{\nu}b \, (\overline{a}\langle b \rangle \mid b(r). \, \boldsymbol{\nu}p \, (p(x,q). \, F\langle q \rangle \mid \overline{p}\langle x, r \rangle))$$
$$\Uparrow_{\succcurlyeq} \quad \boldsymbol{\nu}b \, (\overline{a}\langle b \rangle \mid b(r). \, F\langle r \rangle)$$

(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

$$D \stackrel{\text{def}}{=} ((\boldsymbol{\nu}x, p, b \,)([\cdot]\langle p \rangle \mid x(r). \, r(y,z). \, (\overline{a}\langle x, b \rangle \mid b(u). \, \overline{y}\langle u \rangle))) \tag{4.1}$$

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

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

Definition 4.7. 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 4.8. Let $\asymp$ and $\leq$ be relations on π-agents where

(1) $\asymp$ is a congruence, $\leq$ a plain 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 4.9 (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 respectful for $\asymp$ under $\leq$. We have:*

(i) *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;*
(ii) *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 4.9, 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 4.8. The details are given in Appendix A.

Full abstraction. We put together Theorems 4.4 and 4.9.

Theorem 4.10. *Let $[\![\,]\!]$ be an encoding of the λ-calculus into π-calculus, $\asymp$ a congruence on π-agents. Suppose there is a plain 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 4.4(i) and 4.10(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 4.11. *Theorems 4.4(i) and 4.10(i) continue to hold when the condition that the abstraction contexts be guarded is replaced by $(*)$ above.*

The proof of Theorem 4.11 can be found in Appendix A.

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 2.a is Milner's original encoding [13]. The one in Figure 2.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 [18].

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 5.1. *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 5.1 for the bisimulation is from [18]. 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. Detailed discussion can be found in Appendix C.

Theorem 5.2. *The encoding of Figure 2.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 2.b is fully abstract for LTs under any of the equivalences $\approx, \sim_{\mathrm{may}}, \sim_{\mathrm{must}}, \text{ or } \sim_{\mathrm{may}}^{\mathrm{asy}}$.

As Lemma 5.1 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 5.1 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 [18], which are valid for strong bisimilarity and hence for $\preccurlyeq^{\Uparrow}$, and the law of Lemma 2.6. 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, and this follows from the validity of the β rule for $\preccurlyeq^{\Uparrow}$.

Having checked the conditions for completeness, the only two additional conditions needed for soundness for LTs are conditions (6) and (7) of Definition 4.8, 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 $\sim_{\mathrm{may}}^{\mathrm{asy}}$ and $\sim_{\mathrm{must}}$, 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 (this argument is straightforward on the basis of the 'inverse context' property explained below).

$$[\![\lambda x.\, M]\!] \quad \overset{\text{def}}{=} \quad (p)\, p(x,q).\, [\![M]\!]\langle q\rangle$$

$$[\![x]\!] \quad \overset{\text{def}}{=} \quad (p)\, \overline{x}\langle p\rangle$$

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

Figure 2.a: Milner's encoding

$$[\![\lambda x.\, M]\!] \quad \overset{\text{def}}{=} \quad (p)\, \boldsymbol{\nu} v\, (\overline{p}\langle v\rangle \mid v(x,q).\, [\![M]\!]\langle q\rangle)$$

$$[\![x]\!] \quad \overset{\text{def}}{=} \quad (p)\, \overline{x}\langle p\rangle$$

$$[\![MN]\!] \quad \overset{\text{def}}{=} \quad (p)\, \boldsymbol{\nu} r\, \Big([\![M]\!]\langle r\rangle \mid$$
$$r(v).\, \boldsymbol{\nu} x\, (\overline{v}\langle x,p\rangle \mid$$
$$!x(q).\, [\![N]\!]\langle q\rangle)\Big) \quad \text{(for } x \text{ fresh)}$$

Figure 2.b: a variant encoding

FIGURE 2. The two encodings of call-by-name

The most delicate point is the 'inverse context' property, i.e., 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 4.6. We give detailed proofs concerning 'inverse context' for the examples in Appendix B.

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 2.5(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 4.10. 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 $M' \overset{\text{def}}{=} \lambda x_1 \ldots \lambda x_n.\, N$, 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 $C[[\![\lambda x_1 \ldots \lambda x_n.\, \Omega]\!]]\Downarrow$ and then, by repeatedly applying the third statement of Lemma 2.5(1) (as Ω is inactive), also $C[[\![\Omega]\!]]\Downarrow$. The converse implication is given by the first statement in Lemma 2.5(1).

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 [11], is in Figure 3. The encoding behaves similarly to that in Figure 2.b; reduction underneath a 'λ' is implemented by exploiting special wire processes (such as $q \triangleright p$). 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

$$\llbracket \lambda x.\,M \rrbracket \;\overset{\text{def}}{=}\; (p)\,(\boldsymbol{\nu} x,q)(\overline{p}\langle x,q\rangle \mid \llbracket M \rrbracket \langle q\rangle) \qquad\qquad\qquad \llbracket x \rrbracket \;\overset{\text{def}}{=}\; (p)\,(x(p').\,(p' \rhd p))$$

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

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

FIGURE 3. Encoding of strong call-by-name

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 (that is, 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., [18].

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 6.1 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 6.1 is similar to that of Theorem 5.2. 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 4.11. 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 $\preccurlyeq^{\Uparrow}$); see [11, 18] for more discussion:

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

This law is also needed when proving the existence of inverse context (the most involved condition). The detailed proof of the existence of inverse context is given in Appendix B.

Theorem 6.1. *The encoding of Figure 3 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 2.a, under asynchronous may equivalence we obtain LTs because in the encoding 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 2.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 2.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 incorporated in a bisimulation congruence by using a contextual form of bisimulation such as *barbed congruence* [18]. 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 4.10(ii) we obtain:

Theorem 7.1. *The encoding of Figure 2.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. We give the detailed development of Theorem 7.1 in Appendix D.

8. CONCLUSIONS AND FUTURE WORK

In this paper we have studied soundness and completeness conditions with respect to BTs and LTs for encodings of the λ-calculus into the π-calculus. While the conditions have been presented on the π-calculus, they can be adapted to some 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 characterizations for various encodings and various behavioural equivalences, including bisimilarity, may and must equivalences, and asynchronous may equivalence. Tables (2.a), (2.b), and (3) summarize the results with respect to BTs and LTs for the encodings and the behavioural equivalences examined in the paper. In a table, a check mark means that corresponding result holds; otherwise a symbol X indicates that the result is false. The results in the first column of Tables (2.a) and (2.b) appear in the literature [17, 18]. Concerning the remaining columns and tables, the results are new, though some of them could have been obtained with variants of the proofs in [17, 18]. The main contribution of the current paper, more than the results themselves, is the identification of some general and abstract conditions that allow one to derive such results. Some of the check marks are not stated in Theorems 5.2, 6.1 and 7.1; they are inferred from the following facts: soundness for LT implies soundness for BT; completeness for BT implies completeness for LT, since LT equality implies BT equality. We recall that $\approx$ is weak bisimilarity; $\sim_{\mathrm{may}}$ is may equivalence; $\sim_{\mathrm{may}}^{\mathrm{asy}}$ is asynchronous may equivalence; $\sim_{\mathrm{must}}$ is must equivalence; and $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$ 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). The negative results in the tables (i.e., the occurrences of symbol X) are consequences of the difference between LTs and BTs: an encoding that is fully abstract for LTs cannot be complete for BTs, whereas an encoding fully abstract for BTs cannot be sound for LTs.

		$\approx$	$\sim_{\text{may}}$	$\sim_{\text{may}}^{\text{asy}}$	$\sim_{\text{must}}$	$\approx_{\text{bc}}^{\text{lin,asy}}$
LT	complete	!	!	!	!	!
	sound	!	!	X	!	X
BT	complete	X	X	!	X	!
	sound	!	!	!	!	!

Table 2.a: Results for Figure 2.a

		$\approx$	$\sim_{\text{may}}$	$\sim_{\text{may}}^{\text{asy}}$	$\sim_{\text{must}}$
LT	complete	!	!	!	!
	sound	!	!	!	!
BT	complete	X	X	X	X
	sound	!	!	!	!

Table 2.b: Results for Figure 2.b

		$\approx$	$\sim_{\text{may}}$	$\sim_{\text{may}}^{\text{asy}}$	$\sim_{\text{must}}$
LT	complete	!	!	!	!
	sound	!	!	!	X
BT	complete	X	X	X	!
	sound	!	!	!	!

Table 3: Results for Figure 3

   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 characterizations 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 η-BTs,

or BTs under infinite η expansions [2]. Works on linearity in the π-calculus, such as [22] might be useful; another possibility might be to exploit *receptive types*, which have a strong impact on the the sequentiality constraints imposed by input prefixes, see e.g., [19].

In the paper we have considered encodings of call-by-name. It would be challenging to apply the study to call-by-value; some preliminary result in this direction has been recently obtained [9] (based however on different proof techniques, namely unique solutions of equations [8]).

Acknowledgements. We thank the anonymous referees for their constructive comments, which have allowed us to improve the presentations and amend a number of problems in the original document.

REFERENCES

[1] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[2] H. P. Barendregt. *The Lambda Calculus—Its Syntax and Semantics*. North-Holland, 1984.

[3] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the pi-calculus. In *Proceedings of TLCA'01*, pages 29–45, 2001.

[4] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the pi-calculus. *Acta Informatica*, 42(2-3):83–141, 2005.

[5] M. Boreale and D. Sangiorgi. Some congruence properties for π-calculus bisimilarities. *Theoretical Computer Science*, 198:159–176, 1998.

[6] Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *Proceedings of CONCUR'11*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011.

[7] M. Dezani-Ciancaglini and E. Giovannetti. From Bohm's theorem to observational equivalences: an informal account. *Electronic Notes in Theoretical Computer Science*, 50(2):83–116, 2001.

[8] A. Durier, D. Hirschkoff, and D. Sangiorgi. Divergence and unique solution of equations. In *CONCUR 2017*, volume 85 of *LIPIcs*, pages 11:1–11:16. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[9] A. Durier, D. Hirschkoff, and D. Sangiorgi. Eager functions as processes. In *LICS 2018*. IEEE Computer Society, 2018.

[10] J.R. Hindley and J.P. Seldin. *Introduction to Combinators and λ-calculus*. Cambridge University Press, 1986.

[11] D. Hirschkoff, J.-M. Madiot, and D. Sangiorgi. Duality and i/o-types in the π-calculus. In *Proceedings of CONCUR'12*, volume 7454 of *LNCS*, pages 302–316. Springer, 2012.

[12] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[13] R. Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992. Research Report 1154, INRIA, Sofia Antipolis, 1990.

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

[15] G. D. Plotkin. T^ω as a universal domain. *Journal of Computer and System Sciences*, 17:209–236, 1978.

[16] D. Sangiorgi. An investigation into functions as processes. In *Proc. Ninth International Conference on the Mathematical Foundations of Programming Semantics (MFPS'93)*, volume 802 of *Lecture Notes in Computer Science*, pages 143–159. Springer Verlag, 1993.

[17] D. Sangiorgi. Typed pi-calculus at work: A correctness proof of jones's parallelisation transformation on concurrent objects. *TAPOS*, 5(1):25–33, 1999.

[18] D. Sangiorgi. Lazy functions and mobile processes. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[19] D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

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

[21] D. Scott. Data types as lattices. *SIAM Journal on Computing*, 5(3):522–587, 1976.

[22] Nobuko Yoshida, Kohei Honda, and Martin Berger. Linearity and bisimulation. *Journal of Logic and Algebraic Programming*, 72(2):207–238, 2007.

Appendix

APPENDIX A. THE PROOFS FOR THE CONDITIONS IN SECTION 4

We first present some auxiliary results.

A.1. **Auxiliary results.**

Proposition A.1. *If $[\![\,]\!]$ and $\mathcal{R}$ validate rule β and $\mathcal{R}$ is a precongruence, then $M \Longrightarrow_{\mathrm{h}} N$ implies $[\![M]\!] \mathcal{R} [\![N]\!]$.*

Proposition A.1. The proof proceeds by induction on the length of $M \Longrightarrow_h N$. The case when the length is zero is trivial. Now we suppose the length is $n + 1$ and show that the result holds. We know from $M \to_h^{n+1} N$ that

$$M \to_h^n M' \to_h N .$$

By induction hypothesis, we have

$$[\![M]\!] \mathcal{R} [\![M']\!]$$

Next there are several cases to consider with regard to $M' \to_h N$.
  (1) $M' = (M_1 M_2) M_3 \cdots M_n \to_h M_1' M_3 \cdots M_n = N$ in which $M_1 M_2$ is a (head) redex and $M_1 M_2 \to_h M_1'$. By validity of β rule we know

$$[\![M_1 M_2]\!] \mathcal{R} [\![M_1']\!] .$$

As $\mathcal{R}$ is a precongruence, we can add an arbitrary context, and thus doing we derive

$$
\begin{aligned}
[\![M]\!] \mathcal{R} [\![M']\!] &= C_{\mathrm{app}}^n [[\![M_1 M_2]\!], [\![M_3]\!], \dots, [\![M_n]\!]] \\
&\mathcal{R} \quad C_{\mathrm{app}}^n [[\![M_1']\!], [\![M_3]\!], \dots, [\![M_n]\!]] = [\![N]\!]
\end{aligned}
$$

   where $C_{\mathrm{app}}^n \overset{\mathrm{def}}{=} [\![[\cdot]_1 [\cdot]_2 \cdots [\cdot]_{n-1}]\!]$.
  (2) $M' = \lambda \widetilde{x}.\, M_1 \to_h \lambda \widetilde{x}.\, M_1' = N$ because $M_1 \to_h M_1'$ and M_1 is not an abstraction, and $\widetilde{x}$ denotes $x_1, x_2, \dots, x_n$. Through similar arguments to case (1), we know

$$[\![M_1]\!] \mathcal{R} [\![M_1']\!]$$

   and, exploiting the precongruence property of $\mathcal{R}$,

$$
\begin{aligned}
[\![M]\!] \mathcal{R} [\![M']\!] &= C_\lambda^{\widetilde{x}} [[\![M_1]\!]] \\
&\mathcal{R} \quad C_\lambda^{\widetilde{x}} [[\![M_1']\!]] = [\![N]\!]
\end{aligned}
$$

   where $C_\lambda^{\widetilde{x}} \overset{\mathrm{def}}{=} C_\lambda^{x_1} [\cdots C_\lambda^{x_n} [\cdot] \cdots]$.
This completes the proof. $\qquad\square$

Lemma A.2. *Suppose that*

(1) *$[\![\ ]\!]$ and $\mathcal{R}$ validate rule β, and $\mathcal{R}$ is a congruence;*
(2) *whenever M is an unsolvable of order 0, then $[\![M]\!]\,\mathcal{R}\,[\![\Omega]\!]$;*
(3) *whenever M is an unsolvable of order ∞, then $[\![M]\!]\,\mathcal{R}\,[\![\Omega]\!]$.*

Then, for any unsolvable M of order n ($n = 0, \ldots \infty$), $[\![M]\!]\,\mathcal{R}\,[\![\Omega]\!]$.

Lemma A.2. For order ∞ this is precisely 3. For the remaining cases we proceed by induction on n. For $n = 0$ this is precisely 2. Suppose now $0 < n$ and M is an unsolvable of order n. By definition, there is N s.t. $M \Longrightarrow_h \lambda x.\, N$. Thus we have, writing Ξ for an unsolvable of order ∞,

$$
\begin{array}{rll}
[\![M]\!] & \mathcal{R} & [\![\lambda x.\, N]\!] \qquad\qquad\qquad\qquad\qquad\quad \text{(proposition A.1)}\\
& = & C_\lambda^x[\,[\![N]\!]\,]\\
& \mathcal{R} & C_\lambda^x[\,[\![\Omega]\!]\,] \qquad\qquad\qquad\qquad\quad \text{(inductive hypothesis)}\\
& \mathcal{R} & C_\lambda^x[\,[\![\Xi]\!]\,] \quad \text{(since } \mathcal{R} \text{ is a congruence and, by (3), } \Xi\,\mathcal{R}\,\Omega\text{)}\\
& \mathcal{R} & [\![\Omega]\!] \qquad\qquad\qquad\quad (\lambda x.\,\Xi \text{ is an unsolvable of order } \infty)
\end{array}
$$

which completes the proof. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

Lemma A.3. *Suppose that*

(1) *$[\![\ ]\!]$ and $\mathcal{R}$ validate rule β, and $\mathcal{R}$ is a congruence;*
(2) *whenever M, N are unsolvable of order 0, then $[\![M]\!]\,\mathcal{R}\,[\![N]\!]$;*
(3) *whenever M, N are unsolvable of order ∞, then $[\![M]\!]\,\mathcal{R}\,[\![N]\!]$.*

Then, whenever M, N are unsolvable of order n ($n = 0, \ldots \infty$), $[\![M]\!]\,\mathcal{R}\,[\![N]\!]$.

Lemma A.3. For ∞ this is precisely 3. For the remaining cases we proceed by induction on n. For $n = 0$ this is precisely 2. Suppose now $0 < n$ and M, N are unsolvable of order n. By definition, there are M', N' s.t. $M \Longrightarrow_h \lambda x.\, M'$, $N \Longrightarrow_h \lambda x.\, N'$, and M', N' are unsolvable of order $n-1$. Thus by Proposition A.1,

$$
[\![M]\!] \quad \mathcal{R} \quad [\![\lambda x.\, M']\!] = C_\lambda^x[\,[\![M']\!]\,] \qquad\qquad C_\lambda^x[\,[\![N']\!]\,] = [\![\lambda x.\, N']\!] \quad \mathcal{R} \quad [\![N]\!]
$$

Then by induction hypothesis and congruence property of $\mathcal{R}$,

$$
[\![M]\!] \quad \mathcal{R} \quad C_\lambda^x[\,[\![M']\!]\,] \quad \mathcal{R} \quad C_\lambda^x[\,[\![N']\!]\,] \quad \mathcal{R} \quad [\![N]\!]
$$

Hence $[\![M]\!]\,\mathcal{R}\,[\![N]\!]$. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

A.2. **The completeness theorems.**

Theorem 4.4: completeness. We follow the convention that by 'condition n', for $1 \leqslant i \leqslant 6$, we mean the corresponding condition in Definition 4.3, and by 'condition i' or 'condition ii' we mean the premise of the corresponding clause in Theorem 4.4.

Assume $LT(M) = LT(N)$, then it follows from the definition of LT equality that one of the following cases holds (as usual modulo α conversion).

(I) M, N are unsolvable of order 0.
(II) $M \Longrightarrow_h \lambda x.\, M_1$, $N \Longrightarrow_h \lambda x.\, N_1$, and $LT(M_1) = LT(N_1)$.
(III) $M \Longrightarrow_h x M_1 \ldots M_n$, $N \Longrightarrow_h x N_1 \ldots N_n$, and $LT(M_i) = LT(N_i)$.

Then we have the following observation.

- Suppose (I) holds; then, by condition (6), $[\![M]\!] \asymp [\![\Omega]\!]$ and $[\![N]\!] \asymp [\![\Omega]\!]$. Thus $[\![M]\!] \asymp [\![N]\!]$, because $\asymp$ is an equivalence relation.

- Suppose (II) holds; then, by Proposition A.1, we infer $[\![M]\!] \geq [\![\lambda x.\, M_1]\!] = C_\lambda^x[\,[\![M_1]\!]\,]$ and, in the same way, $[\![N]\!] \geq C_\lambda^x[\,[\![N_1]\!]\,]$.
- Suppose (III) holds; then, by Proposition A.1, we infer $[\![M]\!] \geq [\![xM_1 \ldots M_n]\!] = C_{\mathrm{var}}^{x,n}[\![M_1]\!] \ldots [\![M_n]\!]]$ and, in the same way, $[\![N]\!] \geq C_{\mathrm{var}}^{x,n}[\![N_1]\!] \ldots [\![N_n]\!]]$.

So we are left with cases (II) and (III), which we handle in the remainder of the proof. Define $\mathcal{R}$ thus:

$$\mathcal{R} \overset{\text{def}}{=} \{([\![M]\!]\langle r\rangle, [\![N]\!]\langle r\rangle) \mid \quad LT(M) = LT(N),$$
$$\text{neither } M \text{ nor } N \text{ is unsolvable of order 0,}$$
$$r \text{ fresh}\}$$

In the remainder we sometimes write $[\![M]\!]\,\mathcal{R}\,[\![N]\!]$ to mean $[\![M]\!]\langle r\rangle\mathcal{R}[\![N]\!]\langle r\rangle$, for some fresh r. We first note that for each $[\![M]\!]\,\mathcal{R}\,[\![N]\!]$, based on (II) and (III) above and the following corresponding observations, one of the following cases is true.

(a) $[\![M]\!] \geq C_\lambda^x[\![M_1]\!]]$, $[\![N]\!] \geq C_\lambda^x[\![N_1]\!]]$ and $([\![M_1]\!], [\![N_1]\!]) \in \mathcal{R}$.
(b) $[\![M]\!] \geq C_{\mathrm{var}}^{x,n}[\![M_1]\!], \ldots, [\![M_n]\!]]$, $[\![N]\!] \geq C_{\mathrm{var}}^{x,n}[\![N_1]\!], \ldots, [\![N_n]\!]]$ and $([\![N_i]\!], [\![N_i]\!]) \in \mathcal{R}$ for all i.

Now, the crux of the proof is to show that $\mathcal{R}$ is an up-to-$\leq$-and-contexts candidate (Definition 4.1), which allows us to conclude $\mathcal{R} \subseteq \asymp$, exploiting the property that $\asymp$ validates the up-to-$\leq$-and-contexts technique, according to condition (3). This intuitively will be possible because $[\![M]\!]$ and $[\![N]\!]$ are related, via the preorder $\geq$, to terms that have a common context, as shown in (a) and (b) above, because $\leq$ is an expansion relation (condition (2) of Definition 4.3), and because the variable and abstraction contexts of $[\![\,]\!]$ are guarded (conditions (i) and (4)), hence the first action from terms such as $C_\lambda^x[\![M_1]\!]]$ and $C_{\mathrm{var}}^{x,n}[\![M_1]\!], \ldots, [\![M_n]\!]]$ only consumes the context. In both (a) and (b), one does not need to worry about closure under substitution (of variables) when a hole is underneath an input prefix, since $\mathcal{R}$ is closed under substitution. That is, $([\![M]\!], [\![N]\!]) \in \mathcal{R}$ implies $([\![M\sigma]\!], [\![N\sigma]\!]) \in \mathcal{R}$, because LT equality is preserved by variable renaming [18, Lemma 18.2.6 and Theorem 18.2.7]), and because the encoding is uniform (which implies that the free names of the encoding of a λ term are included in the free variables of that term). Below are the details for the diagram-chasing requirements. In the diagrams, the implications of the vertical transitions should be read from the left to the right.

- If (a) is true, then since $\leq$ is an expansion (condition (2)), we have the following diagram

$$\begin{array}{ccccccc}
 & & & \overset{\mathcal{R}}{\cdots\cdots\cdots} & & & \\
[\![M]\!]\langle r\rangle & \geq & C_\lambda^x[\![M_1]\!]]\langle r\rangle & & C_\lambda^x[\![N_1]\!]]\langle r\rangle & \leq & [\![N]\!]\langle r\rangle \\
\downarrow{\scriptstyle\mu} & & \downarrow{\scriptstyle\widehat{\mu}} & & \downarrow{\scriptstyle\widehat{\mu}} & & \parallel{\scriptstyle\widehat{\mu}} \\
S & \geq & C_1[\![M_1]\!]] & & C_1[\![N_1]\!]] & \leq & T
\end{array}$$

The existence of context C_1 is due to the fact that C_λ^x is guarded (condition (i)), so the action merely consumes the context C_λ^x, and does not affect the term in the hole. In the case M_1, N_1 are unsolvable of order 0, one directly applies $[\![M_1]\!] \asymp [\![N_1]\!]$; otherwise, $[\![M_1]\!]\,\mathcal{R}\,[\![N_1]\!]$ holds.

- If (b) is true then, again because $\leq$ is an expansion, we have

$$\begin{array}{ccccccc}
& & \overset{\mathcal{R}}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} & & & & \\
[\![M]\!]\langle r\rangle \geq C_{\mathrm{var}}^{x,n}[[\![M_1]\!],\dots,[\![M_n]\!]]\langle r\rangle & & C_{\mathrm{var}}^{x,n}[[\![N_1]\!],\dots,[\![N_n]\!]]\langle r\rangle \leq [\![N]\!]\langle r\rangle \\
\Big\downarrow\mu & \Big\downarrow\widehat{\mu} & & \Big\downarrow\widehat{\mu} & & \Big\Vert\widehat{\mu} \\
S \quad\geq\quad C_2[[\![M_1]\!],\dots,[\![M_n]\!]] & & C_2[[\![N_1]\!],\dots,[\![N_n]\!]] \quad\leq\quad T
\end{array}$$

As in the previous case, the existence of context C_2 is due to the fact that $C_{\mathrm{var}}^{x,n}$ is guarded (condition (4)). Moreover, for each i, if M_i, N_i are unsolvable of order 0, we have $[\![M_i]\!] \asymp [\![N_i]\!]$; otherwise, we have $[\![M_i]\!] \mathcal{R} [\![N_i]\!]$.

This completes the case for LTs.

For BTs, intuitively, if in a term the head reduction never unveils a variable, then the term is unsolvable and can be equated to Ω using the premise of (ii); if head reduction 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 (b) above for LTs. Formally, assume $BT(M) = BT(N)$. Then, from the definition of BT equality, one of the following cases holds:

(I') M, N are unsolvable.
(II') $M \Longrightarrow_{\mathrm{h}} \lambda\widetilde{x}.\, x M_1 \dots M_n$, $N \Longrightarrow_{\mathrm{h}} \lambda\widetilde{x}.\, x N_1 \dots N_n$, and $BT(M_i) = BT(N_i)$.

Suppose (I') holds; then by condition (ii) of this theorem and (5) and (6) of Definition 4.3, we have $[\![M]\!] \asymp [\![N]\!] \asymp \Omega$ by Lemma A.2, which closes the case. Suppose now that (II') holds. We proceed in a similar way to that for LTs. Define $\mathcal{R}'$ as below.

$$\mathcal{R}' \overset{\mathrm{def}}{=} \{([\![M]\!]\langle r\rangle, [\![N]\!]\langle r\rangle) \mid BT(M) = BT(N), \text{neither } M \text{ nor } N \text{ is unsolvable} \atop r \text{ fresh}\}$$

As before, we sometimes write $[\![M]\!] \mathcal{R}' [\![N]\!]$ to mean $[\![M]\!]\langle r\rangle \mathcal{R}' [\![N]\!]\langle r\rangle$, for some fresh r. As for LTs, so here we do not need to worry closure under substitution of the 'up-to-$\leq$-and-contexts' technique because BT equality, as LT equality, is preserved by substitution of variables. For each $[\![M]\!] \mathcal{R}' [\![N]\!]$, from (II') and Proposition A.1, we have

$$[\![M]\!] \geq [\![\lambda\widetilde{x}.\, x M_1 \dots M_n]\!], [\![N]\!] \geq [\![\lambda\widetilde{x}.\, x N_1 \dots N_n]\!], \text{ and } M_i \mathcal{R}' N_i$$

Thus

$$[\![M]\!] \geq C_{\lambda-\mathrm{var}}^{\widetilde{x},x}[[\![M_1]\!]\dots[\![M_n]\!]], [\![N]\!] \geq C_{\lambda-\mathrm{var}}^{\widetilde{x},x}[[\![N_1]\!]\dots[\![N_n]\!]], \text{ and } M_i \mathcal{R}' N_i$$

for some context $C_{\lambda-\mathrm{var}}^{\widetilde{x},x} \overset{\mathrm{def}}{=} C_\lambda^{x_1}[C_\lambda^{x_2}[\dots C_\lambda^{x_l}[C_{\mathrm{var}}^{x,n}]\dots]]$ (in which $\widetilde{x}=x_1,\dots,x_l$). We know $C_{\lambda-\mathrm{var}}^{\widetilde{x},x}$ is guarded thanks to condition (4), so some context C_3 exists s.t. we have the following chasing diagram

$$\begin{array}{ccccccc}
& & \overset{\mathcal{R}'}{\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots} & & & & \\
[\![M]\!]\langle r\rangle \geq C_{\lambda-\mathrm{var}}^{\widetilde{x},x}[[\![M_1]\!],\dots,[\![M_n]\!]]\langle r\rangle & & C_{\lambda-\mathrm{var}}^{\widetilde{x},x}[[\![N_1]\!],\dots,[\![N_n]\!]]\langle r\rangle \leq [\![N]\!]\langle r\rangle \\
\Big\downarrow\mu & \Big\downarrow\widehat{\mu} & & \Big\downarrow\widehat{\mu} & & \Big\Vert\widehat{\mu} \\
S \quad\geq\quad C_3[[\![M_1]\!],\dots,[\![M_n]\!]] & & C_3[[\![N_1]\!],\dots,[\![N_n]\!]] \quad\leq\quad T
\end{array}$$

In the case M_i, N_i are unsolvable of any order, one uses $[\![M_i]\!] \asymp [\![N_i]\!]$; for the remaining cases, one applies $[\![M_i]\!] \mathcal{R}' [\![N_i]\!]$.

We have thus shown that $\mathcal{R}'$ is an up-to-$\leq$-and-contexts candidate, and we can finally conclude by condition (3) that $\mathcal{R}' \subseteq \asymp$. $\qquad\square$

We conclude by presenting the proof of Theorem 4.11, which gives us some alternative condition for completeness (which also yields an alternative condition for full abstraction). Precisely, Theorem 4.11 replaces the condition that the abstraction contexts be guarded with the requirement that

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

Theorem 4: alternative completeness conditions. If $LT(M) = LT(N)$, then one of the following holds.

(I) M, N are unsolvable of order m ($m = 0, \ldots, \infty$).

(II) $M \Longrightarrow_{\mathrm{h}} \lambda \widetilde{x}.\, x M_1 \ldots M_n$, and $N \Longrightarrow_{\mathrm{h}} \lambda \widetilde{x}.\, x N_1 \ldots N_n$, and $LT(M_i) = LT(N_i)$ (in which $\widetilde{x} = x_1, \ldots, x_l$, for some l, and $i = 1, \ldots, n$).

In case (I), by Lemma A.3 (using condition (6), and condition $(*)$ in the statement of Theorem 4.11), we derive $[\![M]\!] \asymp [\![N]\!]$, as desired. In case (II), let

$$\mathcal{R} \stackrel{\mathrm{def}}{=} \{([\![M]\!]\langle r \rangle, [\![N]\!]\langle r \rangle) \mid \quad LT(M) = LT(N), \text{neither } M \text{ nor } N \text{ is unsolvable} \\ r \text{ fresh}\}$$

We sometimes write $[\![M]\!] \mathcal{R} [\![N]\!]$ to mean $[\![M]\!]\langle r \rangle \mathcal{R} [\![N]\!]\langle r \rangle$, for some fresh r. We prove that the relation is an up-to-$\leq$-and-contexts candidate (Definition 4.1), which allows us to conclude $\mathcal{R} \subseteq \asymp$, by condition (3). As in the proof of Theorem 4.4, so here relation $\mathcal{R}$ is closed under name substitutions, which is needed for application of condition (3). For each $[\![M]\!] \mathcal{R} [\![N]\!]$, from (ii) and Proposition A.1, we have

$$[\![M]\!] \geq [\![\lambda \widetilde{x}.\, x M_1 \ldots M_n]\!], [\![N]\!] \geq [\![\lambda \widetilde{x}.\, x N_1 \ldots N_n]\!], \text{ and } M_i \mathcal{R} N_i$$

Thus

$$[\![M]\!] \geq C^{\widetilde{x},x}_{\lambda-\mathrm{var}}[[\![M_1]\!] \ldots [\![M_n]\!]], [\![N]\!] \geq C^{\widetilde{x},x}_{\lambda-\mathrm{var}}[[\![N_1]\!] \ldots [\![N_n]\!]], \text{ and } M_i \mathcal{R} N_i$$

for some context $C^{\widetilde{x},x}_{\lambda-\mathrm{var}} \stackrel{\mathrm{def}}{=} C^{x_1}_{\lambda}[C^{x_2}_{\lambda}[\ldots C^{x_l}_{\lambda}[C^{x,n}_{\mathrm{var}}]\ldots]]$. A key point here is that $C^{\widetilde{x},x}_{\lambda-\mathrm{var}}$ is guarded thanks to condition (4). Thus some context C_4 exists s.t. the following chasing diagram holds

In the case M_i, N_i are unsolvable then they are unsolvable of the same order and we have $[\![M_i]\!] \asymp [\![N_i]\!]$; for the remaining cases, we have $[\![M_i]\!] \mathcal{R} [\![N_i]\!]$. This completes the proof. $\qquad\square$

A.3. The soundness theorem.

Theorem 4.9: soundness. We define

$$\mathcal{R} \stackrel{\text{def}}{=} \{(M, N) \mid [\![M]\!] \asymp [\![N]\!]\}$$

and show that $\mathcal{R}$ implies LT equality. To that aim, it suffices to prove that, for any $M \, \mathcal{R} \, N$ (i.e., $M \asymp N$), the following properties hold.

(1) If M is unsolvable of order 0, then so is N;

(2) If $M \Longrightarrow_h \lambda x. M_1$, then $N \Longrightarrow_h \lambda x. N_1$ and $M_1 \, \mathcal{R} \, N_1$;

(3) If $M \Longrightarrow_h x M_1 \cdots M_n$, then $N \Longrightarrow_h x N_1 \cdots N_n$ and $M_i \, \mathcal{R} \, N_i$ for every $i = 1, \ldots, n$.

The proof proceeds by the case analysis below. Similarly to what is done before, here by 'condition n', for $1 \leqslant i \leqslant 6$, we mean the corresponding condition in Definition 4.8, and by 'condition i' or 'condition ii' we mean the premise of the corresponding clause in Theorem 4.9.

(1) M is unsolvable of order 0. By condition (5), we know $[\![M]\!] \asymp [\![\Omega]\!]$, then since $M \asymp N$

$$[\![\Omega]\!] \asymp [\![N]\!]$$

By condition (6) and condition (i), it must be that N is unsolvable of order 0. This is because if not, two cases are possible: (1) N has order other than 0; (2) N (head) reduces to $\lambda \widetilde{y}. z\widetilde{N}$. Either case would contradict the conditions (conditions (6) and (i), using Proposition A.1).

(2) $M \Longrightarrow_h \lambda x. M_1$. By Proposition A.1 (using conditions (1) and (4)),

$$[\![M]\!] \geq [\![\lambda x. M_1]\!]$$

Then we know from condition (2) that $[\![\lambda x. M_1]\!] \asymp [\![M]\!]$. So

$$[\![N]\!] \asymp [\![\lambda x. M_1]\!]$$

Now from condition (6) and condition (i) of this theorem, it must be that N head reduces to $\lambda x. N_1$, for some N_1, so

$$C_\lambda^x[[\![M_1]\!]] = [\![\lambda x. M_1]\!] \asymp [\![\lambda x. N_1]\!] = C_\lambda^x[[\![N_1]\!]] \tag{A.1}$$

By condition (7), we suppose D is the existing context as stated in Definition 4.5. Then we have

$$D[C_\lambda^x[[\![M_1]\!]]] \qquad \asymp \qquad D[C_\lambda^x[[\![N_1]\!]]]$$

$$\vee \mathord{|} \qquad\qquad\qquad\qquad \vee \mathord{|}$$

$$(\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z). [\![M_1]\!]\langle z\rangle) \qquad (\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z). [\![N_1]\!]\langle z\rangle)$$

where a, b, z fresh, and $b \in \widetilde{b} \subseteq \widetilde{c}$; we recall that the encoding of a λ-term is an abstraction of the π-calculus. Thus

$$(\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z). [\![M_1]\!]\langle z\rangle) \asymp (\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z). [\![N_1]\!]\langle z\rangle)$$

By condition (3),

$$[\![M_1]\!]\langle z\rangle \asymp [\![N_1]\!]\langle z\rangle$$

Thus by Lemma 4.2(2)

$$[\![M_1]\!] \asymp [\![N_1]\!]$$

Hence in summary, $N \Longrightarrow_h \lambda x. N_1$ and $M_1 \, \mathcal{R} \, N_1$.

(3) $M \Longrightarrow_h xM_1 \cdots M_n$. By Proposition A.1 (using conditions (1) and (4)),
$$\llbracket M \rrbracket \geq \llbracket xM_1 \cdots M_n \rrbracket$$
Then we know from condition (2) that $\llbracket M \rrbracket \asymp \llbracket xM_1 \cdots M_n \rrbracket$. So
$$\llbracket N \rrbracket \asymp \llbracket xM_1 \cdots M_n \rrbracket$$
Now from condition (6) and condition (i) of this theorem, it must be that N derives (i.e., head reduces to) $xN_1 \cdots N_n$ for some $N_1, \ldots, N_n$, so by Proposition A.1 we have

$$C_{\mathrm{var}}^{x,n}[\llbracket M_1 \rrbracket, \ldots, \llbracket M_n \rrbracket] = \llbracket xM_1 \cdots M_n \rrbracket \asymp \llbracket xN_1 \cdots N_n \rrbracket = C_{\mathrm{var}}^{x,n}[\llbracket N_1 \rrbracket, \ldots, \llbracket N_n \rrbracket] \qquad \text{(A.2)}$$

Then by condition (7), we suppose D_i $(i = 1, \ldots, n)$ is the existing context as stated in Definition 4.5. So we have

$$D_i[C_{\mathrm{var}}^{x,n}[\llbracket M_1 \rrbracket, \ldots, \llbracket M_n \rrbracket]] \quad \asymp \quad D_i[C_{\mathrm{var}}^{x,n}[\llbracket N_1 \rrbracket, \ldots, \llbracket N_n \rrbracket]]$$

$$\lor\!| \qquad\qquad\qquad\qquad\qquad \lor\!|$$

$$(\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z).\llbracket M_i \rrbracket \langle z\rangle) \qquad (\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z).\llbracket N_i \rrbracket \langle z\rangle)$$

where a, b, z fresh, and $b \in \widetilde{b} \subseteq \widetilde{c}$. Thus
$$(\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z).\llbracket M_i \rrbracket \langle z\rangle) \asymp (\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle\widetilde{c}\rangle \mid b(z).\llbracket N_i \rrbracket \langle z\rangle)$$
By condition (3),
$$\llbracket M_i \rrbracket \langle z\rangle \asymp \llbracket N_i \rrbracket \langle z\rangle$$
Thus by Lemma 4.2(2)
$$\llbracket M_i \rrbracket \asymp \llbracket N_i \rrbracket$$
Hence in summary, $N \Longrightarrow_h xN_1 \cdots N_n$ and $M_i \,\mathcal{R}\, N_i$ for every $i = 1, \ldots, n$.
This completes the proof for LTs.

For the BT case, we define
$$\mathcal{S} \stackrel{\text{def}}{=} \{(M, N) \mid \llbracket M \rrbracket \asymp \llbracket N \rrbracket\}$$
and show that $\mathcal{R}$ implies BT equality. To this end, we prove that, for any $M \,\mathcal{R}\, N$ (i.e., $M \asymp N$), the following properties hold.
(1) If M is unsolvable of order n $(n = 0, \ldots, \infty)$, then N is unsolvable of order m $(m = 0, \ldots, \infty)$.
(2) If $M \Longrightarrow_h \lambda\widetilde{x}.\, xM_1 \cdots M_n$, then $N \Longrightarrow_h \lambda\widetilde{x}.\, xN_1 \cdots N_n$ and $M_i \,\mathcal{R}\, N_i$ for every $i = 1, \ldots, n$.
The proof proceeds by the following case analysis.
(1) M is unsolvable of order n $(n = 0, \ldots, \infty)$. By Lemma A.2 (using condition (4), condition (5), and condition (ii).(a) of this theorem), $\llbracket M \rrbracket \asymp \llbracket \Omega \rrbracket$. Since $\llbracket M \rrbracket \asymp \llbracket N \rrbracket$, we have
$$\llbracket N \rrbracket \asymp \llbracket \Omega \rrbracket$$
Thus N must be unsolvable of some order, because if not, a contradiction would arise by appealing to condition (6) and condition (ii).(b) of this theorem, and to Proposition A.1.

(2) $M \Longrightarrow_h \lambda \widetilde{x}. y M_1 \cdots M_n$. This case can be dealt with in a similar way to that for LTs, by combining cases 2 and 3 there; here one uses condition (7) several times (precisely, the length of $\widetilde{x}$ plus one): one for a variable context and the others for abstraction contexts. Also the condition (ii).(b) is used when determining the shape of N.

$\square$

Appendix B. The 'inverse context' property of the encodings

Lemmas B.1, B.2, B.3 provide the inverse context properties of the examples in Section 5, 6. In each of them, we first give the form of the inverse context, then exemplify how it works when fed with encodings of λ-terms, which is the frequent case (see Theorem 4.9), though generic abstraction can be used in the same way. We recall that $\sim$ is strong bisimilarity (Section 2).

Lemma B.1 (On the first call-by-name encoding, Figure 2.a)**.**
 (1) *The abstraction contexts of $[\![\,]\!]$ have inverse with respect to $\Uparrow_{\succcurlyeq}$;*
 (2) *The variable contexts of $[\![\,]\!]$ have inverse with respect to $\Uparrow_{\succcurlyeq}$.*

Proof. **1**. The abstraction contexts are defined by

$$C_\lambda^x = (p)\, p(x,q).\, [\cdot]\langle q \rangle$$

We define context D as below.

$$D \stackrel{\text{def}}{=} (\boldsymbol{\nu} r, b\,)(\overline{a}\langle b \rangle \mid b(r_1).\,([\cdot]\langle r \rangle \mid \overline{r}\langle x, r_1 \rangle))$$

Then

$$D[C_\lambda^x[[\![M]\!]]] \Uparrow_{\succcurlyeq} (\boldsymbol{\nu} r, b\,)(\overline{a}\langle b \rangle \mid b(r_1).\,([\![M]\!]\langle r_1 \rangle)) \sim (\boldsymbol{\nu} b\,)(\overline{a}\langle b \rangle \mid b(r_1).\,([\![M]\!]\langle r_1 \rangle))$$

2. We know from the encoding that the variable contexts are defined as below.

$$
\begin{aligned}
C_{var}^{x,n} &= [\![x[\cdot]_1 \cdots [\cdot]_n]\!] \\
&= (p)\,\boldsymbol{\nu} r, y\,\left([\![x[\cdot]_1 \cdots [\cdot]_{n-1}]\!]\langle r \rangle \mid \overline{r}\langle y, p \rangle \mid {!}y(q).\,[\![[\cdot]_n]\!]\langle q \rangle\right) \quad y \text{ fresh}
\end{aligned}
$$

If $n = 0$, there is nothing to prove. Suppose $n \geqslant 1$. By [17](Lemma 5.2), we know

$$
\begin{aligned}
C_{var}^{x,n}[[\![M_1]\!], \cdots, [\![M_n]\!]]\langle r_n \rangle &= [\![x M_1 \cdots M_n]\!]\langle r_n \rangle \\
&\sim \boldsymbol{\nu} r_0\,(\overline{x}\langle r_0 \rangle \mid \mathcal{O}\langle r_0, r_n, [\![M_1]\!], \ldots, [\![M_n]\!]\rangle)
\end{aligned}
$$

where $r_1, \ldots, r_{n-1}, x_1, \ldots, x_n, q$ are fresh and

$$
\begin{aligned}
\mathcal{O}\langle r_0, r_n, F_1, \ldots, F_n \rangle \stackrel{\text{def}}{=} \;&\boldsymbol{\nu} r_1, \ldots, r_{n-1}, x_1, \ldots, x_n \\
&\left(\; \overline{r_0}\langle x_1, r_1 \rangle \mid \cdots \mid \overline{r_{n-1}}\langle x_n, r_n \rangle \right. \\
&\left. \mid {!}x_1(q).\, F_1\langle q \rangle \mid \cdots \mid {!}x_n(q).\, F_n\langle q \rangle\right)
\end{aligned}
$$

We need to find the contexts $\{D_i^{x,n} \mid 1 \leqslant i \leqslant n\}$ in which

$$D_i^{x,n}[C_{var}^{x,n}[[\![M_1]\!], \ldots, [\![M_n]\!]]] \Uparrow_{\succcurlyeq} (\boldsymbol{\nu} \widetilde{b}\,)(\overline{a}\langle \widetilde{c} \rangle \mid b(z).\, [\![M_i]\!]\langle z \rangle) \quad (b \in \widetilde{b} \subseteq \widetilde{c})$$

The context $D_i^{x,n}$ takes the shape below $(0 < j < i - 1; a, b, z$ fresh$)$.

$$D_i^{x,n} \stackrel{\text{def}}{=} (\boldsymbol{\nu} r_n, x, b)([\cdot]\langle r_n \rangle \mid$$
$$x(r_0).\, r_0(x_1, r_1).\, \ldots .\, r_j(x_{j+1}, r_{j+1}).\, \ldots .\, r_{i-1}(x_i, r_i').\, (\overline{a}\langle x, b \rangle \mid$$
$$b(z).\, \overline{x_i}\langle z \rangle))$$

It can be observed that

$$D_i^{x,n}[C_{var}^{x,n}[\llbracket M_1 \rrbracket, \ldots, \llbracket M_n \rrbracket]]$$
$$\sim \quad (\boldsymbol{\nu} r_n, x, b)(\boldsymbol{\nu} r_0\,(\overline{x}\langle r_0 \rangle \mid \mathcal{O}\langle r_0, r_n, \llbracket M_1 \rrbracket, \ldots, \llbracket M_n \rrbracket \rangle) \mid$$
$$x(r_0).\, r_0(x_1, r_1).\, \ldots .\, r_j(x_{j+1}, r_{j+1}).\, \ldots .\, r_{i-1}(x_i, r_i').\, (\overline{a}\langle x, b \rangle \mid b(z).\, \overline{x_i}\langle z \rangle))$$
$$\Uparrow\!\!\geqslant \quad (\boldsymbol{\nu} r_n, r_i, x, x_i, b)(\mathcal{O}\langle r_i, r_n, \llbracket M_{i+1} \rrbracket, \ldots, \llbracket M_n \rrbracket \rangle \mid$$
$$!x_i(q).\, \llbracket M_i \rrbracket \langle q \rangle \mid \overline{a}\langle x, b \rangle \mid b(z).\, \overline{x_i}\langle z \rangle)$$
$$\sim \quad (\boldsymbol{\nu} x, x_i, b)(!x_i(q).\, \llbracket M_i \rrbracket \langle q \rangle \mid \overline{a}\langle x, b \rangle \mid b(z).\, \overline{x_i}\langle z \rangle)$$
$$\sim \quad (\boldsymbol{\nu} x, x_i, b)(\overline{a}\langle x, b \rangle \mid b(z).\, (!x_i(q).\, \llbracket M_i \rrbracket \langle q \rangle \mid \overline{x_i}\langle z \rangle))$$
$$\Uparrow\!\!\geqslant \quad (\boldsymbol{\nu} x, b)(\overline{a}\langle x, b \rangle \mid b(z).\, \llbracket M_i \rrbracket \langle z \rangle)$$

where the first occurrence of $\Uparrow\!\!\geqslant$ subsumes $(i + 1)$ internal τ actions. So we are done. $\qquad \square$

Lemma B.2 (On the second call-by-name encoding, Figure 2.b)**.**
  (1) *The abstraction contexts of $\llbracket\,\rrbracket$ have inverse with respect to $\Uparrow\!\!\geqslant$;*
  (2) *The variable contexts of $\llbracket\,\rrbracket$ have inverse with respect to $\Uparrow\!\!\geqslant$.*

Proof.
1. The abstraction contexts are

$$C_\lambda^x = (p)\,\boldsymbol{\nu} v\,(\overline{p}\langle v \rangle \mid v(x, q).\, [\cdot]\langle q \rangle)$$

We define context D thus:

$$D \stackrel{\text{def}}{=} (\boldsymbol{\nu} r, b)(\overline{a}\langle b \rangle \mid b(r_1).\, ([\cdot]\langle r \rangle \mid r(v).\, \overline{v}\langle x, r_1 \rangle))$$

It then holds that

$$D[C_\lambda^x[\llbracket M \rrbracket]] \Uparrow\!\!\geqslant (\boldsymbol{\nu} r, b)(\overline{a}\langle b \rangle \mid b(r_1).\, (\llbracket M \rrbracket \langle r_1 \rangle)) \sim (\boldsymbol{\nu} b)(\overline{a}\langle b \rangle \mid b(r_1).\, (\llbracket M \rrbracket \langle r_1 \rangle))$$

2. In the encoding the variable contexts are defined as:

$$C_{var}^{x,n} = \llbracket x[\cdot]_1 \cdots [\cdot]_n \rrbracket$$
$$= (p)\,\boldsymbol{\nu} r\,\Big(\llbracket x[\cdot]_1 \cdots [\cdot]_{n-1} \rrbracket \langle r \rangle \mid r(v).\, \boldsymbol{\nu} y\,(\overline{v}\langle y, p \rangle \mid !y(q).\, \llbracket [\cdot]_n \rrbracket \langle q \rangle)\Big), \quad y \text{ fresh.}$$

Suppose $n \geqslant 1$, since there is nothing to prove if $n = 0$. By an inductive analysis similar to that in Lemma B.1, we have

$$C_{var}^{x,n}[\llbracket M_1 \rrbracket, \cdots, \llbracket M_n \rrbracket]\langle r_n \rangle = \llbracket x M_1 \cdots M_n \rrbracket \langle r_n \rangle$$
$$\sim \boldsymbol{\nu} r_0\,(\overline{x}\langle r_0 \rangle \mid \mathcal{O}\langle r_0, r_n, \llbracket M_1 \rrbracket, \ldots, \llbracket M_n \rrbracket \rangle)$$

where $r_1, \ldots, r_{n-1}, v_0, \ldots, v_{n-1}, x_1, \ldots, x_n, q$ are fresh and

$$\mathcal{O}\langle r_0, r_n, F_1, \ldots, F_n \rangle \stackrel{\text{def}}{=} \boldsymbol{\nu} r_1, \ldots, r_{n-1}, x_1, \ldots, x_n$$
$$\Big(r_0(v_0).\, \overline{v_0}\langle x_1, r_1 \rangle \mid \cdots \mid$$
$$r_{n-1}(v_{n-1}).\, \overline{v_{n-1}}\langle x_n, r_n \rangle \mid$$
$$!x_1(q).\, F_1\langle q \rangle \mid \cdots \mid !x_n(q).\, F_n\langle q \rangle \Big)$$

We need to design the contexts $\{D_i^{x,n} \mid 1 \leqslant i \leqslant n\}$ in which

$$D_i^{x,n}[C_{var}^{x,n}[\llbracket M_1 \rrbracket, \ldots, \llbracket M_n \rrbracket]] \Uparrow\!\!\geqslant (\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle \widetilde{c} \rangle \mid b(z).\, \llbracket M_i \rrbracket \langle z \rangle) \quad (b \in \widetilde{b} \subseteq \widetilde{c})$$

The context $D_i^{x,n}$ is defined thus, for $0 < j < i-1$, and a, b, z fresh.

$$D_i^{x,n} \stackrel{\text{def}}{=} (\boldsymbol{\nu} r_n, x, v_0, \ldots, v_{i-1}, b\,)$$
$$(\ [\cdot]\langle r_n\rangle \mid x(r_0).\overline{r_0}\langle v_0\rangle \mid v_0(x_1, r_1).\overline{r_1}\langle v_1\rangle \mid$$
$$\ldots \mid v_{j-1}(x_j, r_j).\overline{r_j}\langle v_j\rangle \mid$$
$$\ldots \mid v_{i-2}(x_{i-1}, r_{i-1}).\overline{r_{i-1}}\langle v_{i-1}\rangle \mid v_{i-1}(x_i, r_i').(\overline{a}\langle x, b\rangle \mid b(z).\overline{x_i}\langle z\rangle))$$

It holds that

$$D_i^{x,n}[C_{var}^{x,n}[\llbracket M_1\rrbracket, \ldots, \llbracket M_n\rrbracket]]$$
$$\sim \quad (\boldsymbol{\nu} r_n, x, v_0, \ldots, v_{i-1}, b\,, r_0)$$
$$(\ \overline{x}\langle r_0\rangle \mid \mathcal{O}\langle r_0, r_n, \llbracket M_1\rrbracket, \ldots, \llbracket M_n\rrbracket\rangle) \mid$$
$$x(r_0).\overline{r_0}\langle v_0\rangle \mid v_0(x_1, r_1).\overline{r_1}\langle v_1\rangle \mid$$
$$\ldots \mid v_{j-1}(x_j, r_j).\overline{r_j}\langle v_j\rangle \mid$$
$$\ldots \mid v_{i-2}(x_{i-1}, r_{i-1}).\overline{r_{i-1}}\langle v_{i-1}\rangle \mid v_{i-1}(x_i, r_i').(\overline{a}\langle x, b\rangle \mid b(z).\overline{x_i}\langle z\rangle)$$
$$\Lstronger \quad (\boldsymbol{\nu} r_n, x, x_i, r_i, b\,)(\mathcal{O}\langle r_i, r_n, \llbracket M_{i+1}\rrbracket, \ldots, \llbracket M_n\rrbracket\rangle \mid$$
$$!x_i(q).\llbracket M_i\rrbracket\langle q\rangle \mid \overline{a}\langle x, b\rangle \mid b(z).\overline{x_i}\langle z\rangle)$$
$$\sim \quad (\boldsymbol{\nu} x, x_i, b\,)(!x_i(q).\llbracket M_i\rrbracket\langle q\rangle \mid \overline{a}\langle x, b\rangle \mid b(z).\overline{x_i}\langle z\rangle)$$
$$\sim \quad (\boldsymbol{\nu} x, x_i, b\,)(\overline{a}\langle x, b\rangle \mid b(z).(!x_i(q).\llbracket M_i\rrbracket\langle q\rangle \mid \overline{x_i}\langle z\rangle))$$
$$\Lstronger \quad (\boldsymbol{\nu} x, b\,)(\overline{a}\langle x, b\rangle \mid b(z).\llbracket M_i\rrbracket\langle z\rangle)$$

$\square$

Lemma B.3 (On the strong call-by-name encoding, Figure 3)**.**

   (1) *The abstraction contexts of $\llbracket\,\rrbracket$ have inverse with respect to $\Lstronger$;*
   (2) *The variable contexts of $\llbracket\,\rrbracket$ have inverse with respect to $\Lstronger$.*

Proof. As noted, the following property (which admits a routine reasoning) is used in the proof of this lemma.

$$\boldsymbol{\nu} r\,(r \triangleright p \mid \llbracket M\rrbracket\langle r\rangle)\Lstronger \llbracket M\rrbracket\langle p\rangle \tag{B.1}$$

Below we cope with each clause of the lemma.
1. The abstraction context is

$$C_\lambda^x = (p)\,\boldsymbol{\nu} x, q\,(\overline{p}\langle x, q\rangle \mid [\cdot]\langle q\rangle)$$

We define

$$D \stackrel{\text{def}}{=} (\boldsymbol{\nu} r, b\,)([\cdot]\langle r\rangle \mid r(x, q).(\overline{a}\langle x, b\rangle \mid b(r_1).(q \triangleright r_1)))$$

We then have

$$D[C_\lambda^x[\llbracket M\rrbracket]]$$
$$= \quad (\boldsymbol{\nu} r, b\,)((\boldsymbol{\nu} x, q\,)(\overline{r}\langle x, q\rangle \mid \llbracket M\rrbracket\langle q\rangle) \mid r(x, q).(\overline{a}\langle x, b\rangle \mid b(r_1).(q \triangleright r_1)))$$
$$\Lstronger \quad (\boldsymbol{\nu} r, b\,)((\boldsymbol{\nu} x, q\,)(\llbracket M\rrbracket\langle q\rangle \mid \overline{a}\langle x, b\rangle \mid b(r_1).(q \triangleright r_1)))$$
$$\sim \quad (\boldsymbol{\nu} b, x, q\,)(\overline{a}\langle x, b\rangle \mid \llbracket M\rrbracket\langle q\rangle \mid b(r_1).(q \triangleright r_1))$$
$$\sim \quad (\boldsymbol{\nu} b, x, q\,)(\overline{a}\langle x, b\rangle \mid b(r_1).(\llbracket M\rrbracket\langle q\rangle \mid q \triangleright r_1))$$
$$\Lstronger \quad (\boldsymbol{\nu} b, x\,)(\overline{a}\langle x, b\rangle \mid b(r_1).\llbracket M\rrbracket\langle r_1\rangle)$$

where property (B.1) is used.
2. The variable context in the encoding is defined as

$$C_{var}^{x,n} = \llbracket x[\cdot]_1 \cdots [\cdot]_n\rrbracket$$
$$= (p)\,\boldsymbol{\nu} q, r\,(\llbracket x[\cdot]_1 \cdots [\cdot]_{n-1}\rrbracket\langle q\rangle \mid q(y, p').(p' \triangleright p \mid !\overline{y}\langle r\rangle.\llbracket[\cdot]_n\rrbracket\langle r\rangle)) \ (y\text{ fresh})$$

Suppose $n \geqslant 1$ (nothing to prove if $n = 0$). By an inductive analysis similar to that in Lemma B.1, we have

$$C_{var}^{x,n}[\![M_1]\!], \cdots, [\![M_n]\!]_{]}\langle r_n \rangle =$$
$$[\![xM_1 \cdots M_n]\!]\langle r_n \rangle \sim \boldsymbol{\nu} q_0 \, (x(p_0').\, (p_0' \rhd q_0) \mid \mathcal{O}\langle q_0, q_n, [\![M_1]\!], \ldots, [\![M_n]\!]\rangle)$$

where $q_1, \ldots, q_{n-1}, r_1, \ldots, r_n, x_1, \ldots, x_n$ are fresh and

$$\mathcal{O}\langle q_0, q_n, F_1, \ldots, F_n \rangle \overset{\text{def}}{=}$$
$$\boldsymbol{\nu} q_1, \ldots, q_{n-1}, r_1, \ldots, r_n, x_1, \ldots, x_n$$
$$\Big(\; q_0(x_1, p_1').\, (p_1' \rhd q_1 \mid !\overline{x_1}\langle r_1 \rangle.\, [\![M_1]\!]\langle r_1 \rangle) \mid$$
$$\cdots \mid q_{i-1}(x_i, p_i').\, (p_i' \rhd q_i \mid !\overline{x_i}\langle r_i \rangle.\, [\![M_i]\!]\langle r_i \rangle)$$
$$\cdots \mid q_{n-1}(x_n, p_n').\, (p_n' \rhd q_n \mid !\overline{x_n}\langle r_n \rangle.\, [\![M_n]\!]\langle r_n \rangle)\Big) \qquad (i = 1, \ldots, n)$$

We need to design the contexts $\{D_i^{x,n} \mid 1 \leqslant i \leqslant n\}$ in which

$$D_i^{x,n}[C_{var}^{x,n}[\![M_1]\!], \ldots, [\![M_n]\!]]] \; \Uparrow_{\succcurlyeq} \; (\boldsymbol{\nu}\widetilde{b})(\overline{a}\langle \widetilde{c} \rangle \mid b(z).\, [\![M_i]\!]\langle z \rangle) \quad (b \in \widetilde{b} \subseteq \widetilde{c})$$

The context $D_i^{x,n}$ is defined as $(0 < j < i - 1; a, b, z \text{ fresh})$.

$$D_i^{x,n} \overset{\text{def}}{=} (\boldsymbol{\nu} r_n, p_0', \ldots, p_i', x, x_1, \ldots, x_i, b)([\cdot]\langle r_n \rangle \mid \overline{a}\langle x, b \rangle \mid$$
$$\overline{x}\langle p_0' \rangle.\, \overline{p_0'}\langle x_1, p_1' \rangle.\cdots.\overline{p_{j-1}'}\langle x_j, p_j' \rangle.\cdots.\overline{p_{i-1}'}\langle x_i, p_i' \rangle \mid$$
$$b(r_1).\, x_i(r_2).\, (r_2 \rhd r_1))$$

Then

$$D_i^{x,n}[C_{var}^{x,n}[\![M_1]\!], \ldots, [\![M_n]\!]]]$$
$$\sim \quad (\boldsymbol{\nu} r_n, p_0', \ldots, p_i', x, x_1, \ldots, x_i, b)(\boldsymbol{\nu} q_0)$$
$$(\; (x(p_0').\, (p_0' \rhd q_0) \mid \mathcal{O}\langle q_0, q_n, [\![M_1]\!], \ldots, [\![M_n]\!]\rangle)$$
$$\mid \overline{a}\langle x, b \rangle \mid \overline{x}\langle p_0' \rangle.\, \overline{p_0'}\langle x_1, p_1' \rangle.\cdots.\overline{p_{j-1}'}\langle x_j, p_j' \rangle.\cdots.\overline{p_{i-1}'}\langle x_i, p_i' \rangle$$
$$\mid b(r_1).\, x_i(r_2).\, (r_2 \rhd r_1))$$
$$\Uparrow_{\succcurlyeq} \quad (\boldsymbol{\nu} r_n, x, x_i, q_i, r_i, b)(\mathcal{O}\langle q_i, q_n, [\![M_{i+1}]\!], \ldots, [\![M_n]\!]\rangle) \mid$$
$$\overline{a}\langle x, b \rangle \mid !\overline{x_i}\langle r_i \rangle.\, [\![M_i]\!]\langle r_i \rangle \mid b(r_1).\, x_i(r_2).\, (r_2 \rhd r_1))$$
$$\sim \quad (\boldsymbol{\nu} x, x_i, r_i, b)(\overline{a}\langle x, b \rangle \mid !\overline{x_i}\langle r_i \rangle.\, [\![M_i]\!]\langle r_i \rangle \mid b(r_1).\, x_i(r_2).\, (r_2 \rhd r_1))$$
$$\sim \quad (\boldsymbol{\nu} x, x_i, r_i, b)(\overline{a}\langle x, b \rangle \mid b(r_1).\, (!\overline{x_i}\langle r_i \rangle.\, [\![M_i]\!]\langle r_i \rangle \mid x_i(r_2).\, (r_2 \rhd r_1)))$$
$$\Uparrow_{\succcurlyeq} \quad (\boldsymbol{\nu} x, r_i, b)(\overline{a}\langle x, b \rangle \mid b(r_1).\, ([\![M_i]\!]\langle r_i \rangle \mid r_i \rhd r_1))$$
$$\Uparrow_{\succcurlyeq} \quad (\boldsymbol{\nu} x, b)(\overline{a}\langle x, b \rangle \mid b(r_1).\, ([\![M_i]\!]\langle r_1 \rangle))$$

where where the first occurrence of $\Uparrow_{\succcurlyeq}$ subsumes $(2i + 1)$ internal τ actions, and the last step uses property (B.1). Thus we are done. $\qquad \square$

APPENDIX C. MORE PROPERTIES OF THE ENCODINGS IN SECTION 5 AND SECTION 6

Proof of Lemma 5.1. For convenience, we recall the content of Lemma 5.1 and Definition 4.1 below.

Lemma 5.1. 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.

Definition 4.1. 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 $Q \xRightarrow{\hat{\mu}} Q'$ 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 two 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.

As mentioned before, the case for bisimulation is proven in [18]. The cases of may, asynchronous may and must equivalences have similar proofs, which follow from their definitions and use the expansion ($\preccurlyeq$ for $\sim_{\mathrm{may}}$ and $\sim_{\mathrm{may}}^{\mathrm{asy}}$; $\preccurlyeq^{\Uparrow}$ for $\sim_{\mathrm{must}}$) in a way similar to the technique for bisimulation in [18]. We focus on $\sim_{\mathrm{may}}$ below. Let $\mathcal{R}$ be a relation as in Definition 4.1, where $\asymp$ is $\sim_{\mathrm{may}}$ and $\leq$ is $\preccurlyeq$. Take the relation

$$\mathcal{S} \stackrel{\mathrm{def}}{=} \asymp \cup \{(P_1, P_2) \mid \text{ there is a context } C \text{ s.t.}$$
$$P_i \succcurlyeq C[\widetilde{P_i}](i=1,2) \text{ and } (\widetilde{P_1}, \widetilde{P_2}) \in \mathcal{R}\cup\asymp\}$$

Notation $(\widetilde{P_1}, \widetilde{P_2}) \in \mathcal{R}$ means $(P_1^k, P_2^k) \in \mathcal{R}$ for every $P_i^k \in \widetilde{P}_i (i = 1, 2)$, $k \leqslant m$ (m is the number of holes in the π-context C).

Obviously we have $\mathcal{R} \subseteq \mathcal{S}$, then it suffices to show that $\mathcal{S} \subseteq \asymp$. Assume $P_1\mathcal{S}P_2$. For any context D, suppose $D[P_1]\Downarrow$. We want to show $D[P_2]\Downarrow$. To do so, we first note that from $P_1 \succcurlyeq C[\widetilde{P_1}]$ we have $D[C[\widetilde{P_1}]]\Downarrow$ (with not more silent moves before the observable action), then we derive $D[C[\widetilde{P_2}]]\Downarrow$ by a case analysis, and finally we have $D[P_2]\Downarrow$ using $\succcurlyeq$ again. We detail the analysis below.

There is a case analysis to be made on the origin of the observable in $D[C[\widetilde{P_1}]]\Downarrow$, according to where the action in the observable comes: (1) from D; (2) from C; (3) from $\widetilde{P_1}$; (4) from an interaction between a component of $\widetilde{P_1}$ and its context. We only show the details for (3), which is the interesting case; cases (1) and (2) are easy, and (4) is easily handled by relying on (1), (2) and (3).

To deal with (3), there are two subcases on $\widetilde{P_1}\Downarrow$: (3-1) the observable is from P_1^k for some k; (3-2) the observable is from interaction between components of $\widetilde{P_1}$. We focus on (3-1) since (3-2) can be tackled similarly to (3-1). For convenience, we set some notations: $\Downarrow^n$ means "observable in n steps" (of internal move), $\Downarrow^{\leqslant n}$ means "observable in no more than n steps", and $\xrightarrow{\tau}_n$ means n consecutive τ actions.

In the subcase of (3-1), we have $P_1^k \Downarrow^n$ for some n, i.e., $P_1^k \xrightarrow{\tau}_n \xrightarrow{\mu}$ for some μ other than τ. We want to show the result $P_2^k\Downarrow$ so that the subcase can be closed. We know $(P_1^k, P_2^k) \in \mathcal{R} \cup \asymp$. The case when $(P_1^k, P_2^k) \in \asymp$ is immediate. For $(P_1^k, P_2^k) \in \mathcal{R}$, we proceed by induction on the n in $P_1^k\Downarrow^n$ to show $P_2^k\Downarrow$; the property of expansion will be needed. We elaborate the arguments below.

- $n{=}0$. This is trivial based on the definition of $\mathcal{R}$.
- Assuming the result holds whenever the number of internal actions before μ is less than n, we show that it also holds for n. We know that for some R_1

$$P_1^k \xrightarrow{\tau} R_1 \xrightarrow{\tau}_{n-1} \xrightarrow{\mu} \qquad \text{i.e., } R_1 \Downarrow^{n-1}$$

Because $(P_1^k, P_2^k) \in \mathcal{R}$, in terms of Definition 4.1, we have for some R_2

$$P_2^k \overset{\tau}{\Longrightarrow} R_2$$

and for some context $C', \widetilde{R_1}, \widetilde{R_2}$ such that $(\widetilde{R_1}, \widetilde{R_2}) \in \mathcal{R} \cup \asymp$, it holds that

$$R_1 \geq C'[\widetilde{R_1}] \quad \text{and} \quad C'[\widetilde{R_2}] \leq R_2$$

Since $R_1 \Downarrow^{n-1}$, we have

$$C'[\widetilde{R_1}] \Downarrow^{\leqslant n-1}$$

Then by (possibly) using induction hypothesis, we derive

$$C'[\widetilde{R_2}] \Downarrow$$

Thus

$$R_2 \Downarrow$$

So in summary

$$R_1(\Downarrow^{n-1}) \geq C'[\widetilde{R_1}](\Downarrow^{\leqslant n-1}) \quad \text{and} \quad C'[\widetilde{R_2}](\Downarrow) \leq R_2(\Downarrow)$$

Hence we finally have

$$P_2^k \Downarrow$$

<div style="text-align:right">□</div>

APPENDIX D. DISCUSSION OF SECTION 7

We briefly introduce the asynchronous π-calculus with linear types, based on the calculus in Section 2. The reader is referred to [18] for more details. After that, we explain how to adapt the conditions to a setting allowing types, and prove Theorem 7.1 from Section 7.

D.1. Linearity: types, typing (rules), barbed congruence, and bisimulation.

D.1.1. *Asynchronous π-calculus with linear types.* The linearly typed asynchronous π-calculus, notation π^l, is defined in Figure 4 (types), Figure 5 (operation on types), Figure 6 (syntax), Figure 7 (typing), and Figure 8 (semantics). They are based on the corresponding part in [18, Chapter 8]. We start with some remark about the notations.

- Notation $\widetilde{b} : \widetilde{T}$ is a shortcut for $b_i : T_i$ $(i = 1, \ldots, n$ where n is the size of $\widetilde{b})$.
- $\sim_{type}$ is type equality, defined as the (smallest) congruence satisfying the rule below.

$$\text{EQ-UNFOLD} \ \frac{}{\mu X. T \sim_{type} T\{\mu X. T/X\}}$$

The figures 4-8 follow the formulation in [18], to which we refer for more details.

The following are standard definitions and notations concerning type environments.

Definition D.1 (Type environment)**.**
- An *assignment* is of the form $a : T$, meaning that a gets type T.
- A *type environment*, represented by Γ, Δ, is a finite set of assignments.
- Metavariable Θ ranges over type environment.
- Given a type environment Γ, $\Gamma(a)$ stands for the type assigned to a by Γ, and $\mathbf{supp}(\Gamma) \overset{\text{def}}{=} \{a \mid \Gamma(a) \text{ is defined}\}$.

Types

$$
\begin{array}{llll}
S, T & ::= & L & \text{link type} \\
& | & \Diamond & \text{behavior type} \\
& | & L \to \Diamond & \text{abstraction type}
\end{array}
$$

$$
\begin{array}{llll}
L & ::= & \textbf{unit} & \text{basic type} \\
& | & \sharp L & \text{connection} \\
& | & iL & \text{input capability} \\
& | & oL & \text{output capability} \\
& | & l_\sharp L & \text{linear connection} \\
& | & l_i L & \text{linear input capability} \\
& | & l_o L & \text{linear output capability} \\
& | & \prod_{i=1}^{n} L_i \ (n \geqslant 2) & \text{product} \\
& & (\text{briefly } \widetilde{L}) & \\
& | & X & \text{type variable} \\
& | & \mu X . L & \text{recursive type}
\end{array}
$$

Type environments

$$
\begin{array}{lll}
\Gamma & ::= & \Gamma, x : L \\
& | & \Gamma, \widetilde{x} : \widetilde{L} \\
& | & \emptyset
\end{array}
$$

FIGURE 4. Types (including linear types)

- $\Gamma \backslash a$ is the type environment excluding only the definition on a in Γ.
- A type environment is *closed* if it does not contain free type variables in its assignments, and $\Gamma(a)$ is a link type for all $a \in \textbf{supp}(\Gamma)$. By default, we consider closed type environments.
- Γ *extends* Δ if $\textbf{supp}(\Delta) \subseteq \textbf{supp}(\Gamma)$ and $\Gamma \vdash x : \Delta(x)$ for every $x \in \textbf{supp}(\Delta)$.

D.1.2. *Asynchronous typed barbed congruence.*

We give the definition of barbed congruence in π^l, and some of its properties. The following notion is used to express the quantification over contexts in the definition of barbed congruence.

Definition D.2 ((Γ/Δ)-context)**.** Context C is a (Γ/Δ)-*context* if, assuming $[\cdot]$ as a process, either the judgement $\Gamma \vdash C : \Diamond$ or $\Gamma \vdash C : T \to \Diamond$ is valid.

Note that if C is a (Γ/Δ)-context and $\Delta \vdash P$, then $\Gamma \vdash C[P]$.

Notation $P \downarrow_\mu$ (respectively $P \Downarrow_\mu$) means $P \xrightarrow{\mu}$ (respectively $P \Longrightarrow \xrightarrow{\mu}$).

Definition D.3 (Asynchronous typed barbed bisimilarity and congruence)**.**

Combination of types

$$l_i T \uplus l_o T \quad \overset{\text{def}}{=} \quad l_\sharp T$$
$$T \uplus T \quad \overset{\text{def}}{=} \quad T \qquad \text{if } T \text{ is not a linear type}$$
$$S \uplus T \quad \overset{\text{def}}{=} \quad \mathbf{error} \qquad \text{otherwise}$$

Combination of type environments

$$(\Gamma_1 \uplus \Gamma_2)(x) \overset{\text{def}}{=} \begin{cases} \Gamma_1(x) \uplus \Gamma_2(x) & \text{if both } \Gamma_1(x) \text{ and } \Gamma_2(x) \text{ are defined} \\ \Gamma_1(x) & \text{if } \Gamma_1(x) \text{ is defined, but not } \Gamma_2(x) \\ \Gamma_2(x) & \text{if } \Gamma_2(x) \text{ is defined, but not } \Gamma_1(x) \\ \text{undefined} & \text{if neither } \Gamma_1(x) \text{ nor } \Gamma_2(x) \text{ is defined,} \\ & \text{or both are defined but } \Gamma_1(x) \uplus \Gamma_2(x) = \mathbf{error} \end{cases}$$

Extraction of linear names

$$Lin(\Gamma) \quad \overset{\text{def}}{=} \quad \{x \mid \Gamma(x) \text{ is a linear type}\}$$
$$Lin_i(\Gamma) \quad \overset{\text{def}}{=} \quad \{x \mid \Gamma(x) = l_i S \text{ or } \Gamma(x) = l_\sharp S, \text{ for some } S\}$$

FIGURE 5. Operations on types

(1) *Asynchronous typed barbed bisimilarity* is the largest symmetric relation, $\approx_{\text{bb}}^{\text{lin,asy}}$, such that whenever $P \approx_{\text{bb}}^{\text{lin,asy}} Q$
    (1) [Barb-preserving] $P\!\downarrow_\mu$ implies $Q\!\Downarrow_\mu$, where μ is an output;
    (2) [Reduction-closed] $P \overset{\tau}{\rightarrow} P'$ implies $Q \Longrightarrow \approx_{\text{bb}}^{\text{lin,asy}} P'$.
(2) Suppose $\Delta \vdash P, Q$ for some Δ. Processes P and Q are *asynchronously typed barbed congruent (w.r.t. Δ)*, written $\Delta \Vdash P \approx_{\text{bc}}^{\text{lin,asy}} Q$, if for every (Γ/Δ)-context C (in which Γ is closed), one has $C[P] \approx_{\text{bb}}^{\text{lin,asy}} C[Q]$.

Some technical concepts and results are given below without comments. They are discussed and proved in [18]. We refer the reader to [18] for more details.

Definition D.4 (Δ-to-Γ substitution). Suppose Δ, Γ are type environments, and σ is a substitution on names. We say that σ is a Δ-*to-*Γ *substitution* if

for every x on which Δ is defined, $\Gamma \vdash \sigma(x) : \Delta(x)$.

Lemma D.5 (Substitution Lemma). *Assume $\Gamma, a : S \vdash A : T$, $\Gamma' \vdash b : S$, and $\Gamma \uplus \Gamma'$ is defined. Then $\Gamma \uplus \Gamma' \vdash A\{b/a\} : T$.*

Lemma D.6 (Subject Reduction Lemma). *Let Γ be closed and $\Gamma \vdash P$. Suppose $P \overset{\alpha}{\rightarrow} P'$.*
  (1) *If α is τ, then either $\Gamma \vdash P'$ or there exist a, T with $\Gamma(a) = l_\sharp T$ such that $\Gamma \backslash a \vdash P'$.*
  (2) *If α is $a(\widetilde{c})$, then there exist $\Gamma_1, \Gamma_1, \widetilde{S}$ such that*
      (a) $\Gamma = \Gamma_1 \uplus \Gamma_2$;
      (b) $\Gamma_1 \vdash a : i\widetilde{S}$ *or* $\Gamma_1 \vdash a : l_i\widetilde{S}$;
      (c) *if $\Gamma_3 \vdash \widetilde{c} : \widetilde{S}$ and $\Gamma_2 \uplus \Gamma_3$ is defined, then $\Gamma_2 \uplus \Gamma_3 \vdash P'$.*
  (3) *If α is $(\boldsymbol{\nu}\widetilde{d} : \widetilde{T})\overline{a}\langle\widetilde{c}\rangle$, then there exist $\Gamma_1, \Gamma_2, \Gamma_3, \widetilde{S}$ such that*
      (a) $\Gamma, \widetilde{d} : \widetilde{T} = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$;
      (b) $\Gamma_1 \vdash a : o\widetilde{S}$ *or* $\Gamma_1 \vdash a : l_o\widetilde{S}$;
      (c) $\Gamma_2 \vdash \widetilde{c} : \widetilde{S}$;

			Values
v	$::=$	x	name
	|	$\widetilde{x}$	name product
	|	$\star$	basic value
			Agents
A	$::=$	P	process
	|	F	abstraction
			Processes
P, Q, R	$::=$	$\mathbf{0}$	null
	|	$a(\widetilde{b}).\,P$	input
	|	$\overline{a}\langle\widetilde{b}\rangle$	output
	|	$P \mid Q$	parallel composition
	|	$(\boldsymbol{\nu}a : L\,)P$	restriction
	|	$!a(\widetilde{b}).\,P$	replication
	|	$F\langle a\rangle$	application
	|	$\mathbf{wrong}$	error
			Abstractions
F	$::=$	$(a)\,P$	

FIGURE 6. Syntax

(d) $\Gamma_3 \vdash P'$.

Lemma D.8 offers a characterization of $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$, as an extension to related results in [18].

Definition D.7. Suppose $\Delta \vdash P, Q$. We write $\Delta \Vdash P \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} Q$ if, for every closed Γ that extends Δ, every Δ-to-Γ substitution σ, and every process R such that $\Gamma \vdash R$, we have $P\sigma \mid R \approx_{\mathrm{bb}}^{\mathrm{lin,asy}} Q\sigma \mid R$.

The following Context Lemma is useful when reasoning about the behaviour of processes that are barbed congruent, so to drastically limit the quantification on contexts by appealing to the above Definition D.7. In this way, the reasoning can become similar to that employed when working with ordinary labeled bisimilarity in the untyped case (e.g., case (4) in the proof of Theorem 7.1 in Section D.2.2; see also [18] for other examples and discussion).

Lemma D.8 (Context Lemma for linearity)**.** *Suppose $\Delta \vdash P, Q$.*
It holds that $\Delta \Vdash P \approx_{\mathrm{bc}}^{\mathrm{lin,asy}} Q$ iff $\Delta \Vdash P \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} Q$.

Proof. The implication that $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$ implies $\approx_{\mathrm{ct}}^{\mathrm{lin,asy}}$ is easy. For the other direction, we prove by induction on the structure of (Γ/Δ)-context C (in which Γ is closed) that

$$\Delta \Vdash P \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} Q \quad \text{implies} \quad \Gamma \Vdash C[P] \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} C[Q]$$

Value typing

$$\text{TV-BASE} \quad \frac{Lin(\Gamma) = \emptyset}{\Gamma \vdash \star : \mathbf{unit}} \qquad \text{TV-NAME} \quad \frac{Lin(\Gamma) = \emptyset}{\Gamma, x : T \vdash x : T} \qquad \text{TV-PRODUCT} \quad \frac{\Gamma_i \vdash x_i : T_i \qquad i = 0, 1, \ldots, n}{\biguplus_i \Gamma_i \vdash \widetilde{x} : \widetilde{T}}$$

$$\text{TV-SUBSUMPTION} \quad \frac{\Gamma \vdash x : S \qquad S \trianglelefteq T}{\Gamma \vdash x : T} \qquad \text{TV-EQ} \quad \frac{\Gamma \vdash x : S \qquad S \sim_{type} T}{\Gamma \vdash x : T}$$

Subtyping

$$\text{SUB-REFL} \ \frac{}{T \trianglelefteq T} \qquad\qquad \text{SUB-TRANS} \ \frac{S \trianglelefteq S' \qquad S' \trianglelefteq T}{S \trianglelefteq T}$$

$$\text{SUB-}\sharp\text{I} \ \frac{}{\sharp T \trianglelefteq iT} \qquad \text{SUB-}\sharp\text{O} \ \frac{}{\sharp T \trianglelefteq oT} \qquad \text{SUB-II} \ \frac{S \trianglelefteq T}{iS \trianglelefteq iT}$$

$$\text{SUB-OO} \ \frac{T \trianglelefteq S}{oS \trianglelefteq oT} \qquad \text{SUB-PRODUCT} \ \frac{S_i \trianglelefteq T_i \qquad i = 0, 1, \ldots, n}{\widetilde{S} \trianglelefteq \widetilde{T}}$$

$$\text{SUB-BB} \ \frac{S \trianglelefteq T \qquad T \trianglelefteq S}{\sharp S \trianglelefteq \sharp T}$$

Process typing

$$\text{T-NIL} \ \frac{Lin(\Gamma) = \emptyset}{\Gamma \vdash \mathbf{0} : \Diamond} \qquad\qquad \text{T-}\Delta\text{-HOLE} \ \frac{\Theta \text{ extends the (fixed) } \Delta}{\Theta \vdash [\cdot] : \Diamond}$$

$$\text{T-INP} \ \frac{\Gamma_1 \vdash a : \varsigma\widetilde{T} \ (\varsigma \in \{i, l_i\}) \qquad \Gamma_2, \widetilde{b} : \widetilde{T} \vdash P : \Diamond}{\Gamma_1 \uplus \Gamma_2 \vdash a(\widetilde{b}). P : \Diamond}$$

$$\text{T-OUT} \ \frac{\Gamma_1 \vdash a : \varsigma\widetilde{T} \ (\varsigma \in \{o, l_o\}) \qquad \Gamma_2 \vdash \widetilde{b} : \widetilde{T}}{\Gamma_1 \uplus \Gamma_2 \vdash \overline{a}\langle\widetilde{b}\rangle : \Diamond}$$

$$\text{T-PAR} \ \frac{\Gamma_1 \vdash P : \Diamond \qquad \Gamma_2 \vdash Q : \Diamond}{\Gamma_1 \uplus \Gamma_2 \vdash P \mid Q : \Diamond} \qquad\qquad \text{T-RES} \ \frac{\Gamma, a : L \vdash P : \Diamond}{\Gamma \vdash (\boldsymbol{\nu} a : L)P : \Diamond}$$

$$\text{T-RES'} \ \frac{\Gamma \vdash P : \Diamond}{\Gamma \vdash (\boldsymbol{\nu} a : L)P : \Diamond}$$

$$\text{T-REP} \ \frac{\Gamma_1 \vdash a : i\widetilde{T} \qquad \Gamma_2, \widetilde{b} : \widetilde{T} \vdash P : \Diamond \qquad Lin(\Gamma_2) = \emptyset}{\Gamma_1 \uplus \Gamma_2 \vdash !a(\widetilde{b}). P : \Diamond}$$

$$\text{T-ABS} \ \frac{\Gamma, a : T \vdash P : \Diamond}{\Gamma \vdash (a) P : T \to \Diamond} \qquad \text{T-APP} \ \frac{\Gamma_1 \vdash F : T \to \Diamond \qquad \Gamma_2 \vdash b : T}{\Gamma_1 \uplus \Gamma_2 \vdash F\langle b\rangle : \Diamond} \qquad .$$

FIGURE 7. Typing rules

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

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

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

$$\texttt{res:}\quad \frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu}a:L\,)P \xrightarrow{\mu} (\boldsymbol{\nu}a:L\,)P'}\ a \text{ does not appear in } \mu$$

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

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

$$\texttt{inpErr:}\ a(\widetilde{b}).\,P \xrightarrow{\tau} \mathbf{wrong}\quad (a \text{ is not a name})$$

$$\texttt{outErr:}\ \overline{a}\langle\widetilde{b}\rangle \xrightarrow{\tau} \mathbf{wrong}\quad (a \text{ is not a name})$$

FIGURE 8. Transition rules

We first give two claims whose proofs are similar to those for the untyped π-calculus. They are used in the analysis of this lemma.

Claim 1. If $\Delta \Vdash P \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} Q$ and Γ extends Δ, then $\Gamma \Vdash P \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} Q$.

Claim 2. If $\Delta \Vdash P \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} Q$ and $\Delta(a){=}S$, then $\Delta\backslash a\Vdash (\boldsymbol{\nu}a:S)P \approx_{\mathrm{ct}}^{\mathrm{lin,asy}} (\boldsymbol{\nu}a:S)Q$.

To proceed, there are a couple of cases to analyze.

- C is $\mathbf{0}$ or $\overline{a}\langle\widetilde{b}\rangle$. This is trivial.
- C is $[\cdot]$. This is by Claim 1.
- C is $R \mid C'$. This is immediate by induction hypothesis and the premise.
- C is $(\boldsymbol{\nu}a:S\,)C'$. This is by Claim 2.
- C is $!a(\widetilde{b}).\,C'$. This case is similar to that for i/o types; see [18].
- C is $a(\widetilde{b}).\,C'$. We focus on the subcase when a is of a linear type. Otherwise the argument is similar to that for i/o types in [18]. Given C as a (Γ/Δ)-context, we have

$$\Gamma \vdash a : l_i\widetilde{S} \quad \text{and} \quad C' \text{ is a } ((\Gamma, \widetilde{b} : \widetilde{S})/\Delta)\text{-context} \tag{D.1}$$

The aim is to prove that for every closed Γ' extending Γ, every R such that $\Gamma' \vdash R$, and every Γ-to-Γ' substitution σ, the relation $\mathcal{R}\cup \approx_{\mathrm{bb}}^{\mathrm{lin,asy}}$ is a barbed bisimulation, where $\mathcal{R}$ is defined as

$$\Big\{ \big((a(\widetilde{b}).\,C'[P])\sigma \mid R, (a(\widetilde{b}).\,C'[Q])\sigma \mid R\big) \ \Big|\ \sigma, R \text{ are as described above}\Big\} \tag{D.2}$$

Then it holds that

$$(a(\widetilde{b}).\,C'[P])\sigma \mid R \approx_{\mathrm{bb}}^{\mathrm{lin,asy}} (a(\widetilde{b}).\,C'[Q])\sigma \mid R$$

Take an element from $\mathcal{R}$, the barb-preserving property should be clear since no immediate output can be made by $(a(\widetilde{b}).\,C'[P])\sigma$ or $(a(\widetilde{b}).\,C'[Q])\sigma$. We thus, in the remainder of the proof, consider the reduction-closed requirement, and show that every reduction of, say, $(a(\widetilde{b}).\,C'[P])\sigma \mid R$ can be matched by $(a(\widetilde{b}).\,C'[Q])\sigma \mid R$. The most interesting case is when the reduction results from the interaction between $(a(\widetilde{b}).\,C'[P])\sigma$ and R. That is,

$$(a(\widetilde{b}).\,C'[P])\sigma \mid R \ \xrightarrow{\tau} \ (\boldsymbol{\nu}\widetilde{d}:\widetilde{T}\,)(C'[P]\sigma\{\widetilde{c}/\widetilde{b}\} \mid R') \tag{D.3}$$

The reduction is shown below to be matched by

$$(a(\widetilde{b}).\,C'[Q])\sigma \mid R \ \xrightarrow{\tau} \ (\boldsymbol{\nu}\widetilde{d}:\widetilde{T}\,)(C'[Q]\sigma\{\widetilde{c}/\widetilde{b}\} \mid R') \tag{D.4}$$

That is,

$$(\boldsymbol{\nu}\widetilde{d}:\widetilde{T}\,)(C'[P]\sigma\{\widetilde{c}/\widetilde{b}\} \mid R') \ \approx_{\mathrm{bb}}^{\mathrm{lin,asy}} \ (\boldsymbol{\nu}\widetilde{d}:\widetilde{T}\,)(C'[Q]\sigma\{\widetilde{c}/\widetilde{b}\} \mid R') \tag{D.5}$$

which is derivable if we can prove the following, because $\approx_{\mathrm{bb}}^{\mathrm{lin,asy}}$ is closed by restriction:

$$C'[P]\sigma\{\widetilde{c}/\widetilde{b}\} \mid R' \ \approx_{\mathrm{bb}}^{\mathrm{lin,asy}} \ C'[Q]\sigma\{\widetilde{c}/\widetilde{b}\} \mid R' \tag{D.6}$$

To this end, (D.6) can be inferred by induction hypothesis on C', which is a $((\Gamma, \widetilde{b}: \widetilde{S})/\Delta)$-context, if one can exhibit that for closed $\Gamma'' \stackrel{\text{def}}{=} \Gamma', \widetilde{d}: \widetilde{T}$ it holds that

$$\sigma\{\widetilde{c}/\widetilde{b}\} \text{ is a } (\Gamma, \widetilde{b}: \widetilde{S})\text{-to-}\Gamma'' \text{ substitution} \tag{D.7}$$

$$\Gamma'' \vdash R' \tag{D.8}$$

In (D.3), the reduction results from an output from R, that is,

$$R \xrightarrow{(\boldsymbol{\nu}\widetilde{d}:\widetilde{T}\,)\overline{a}\langle\widetilde{c}\rangle} R'$$

The most intriguing situation here is when R has the linear output capability on a, i.e., $\Gamma' \vdash a : l_o T$. As $\Gamma' \vdash R$, by the Subject Reduction lemma (Lemma D.6), we have, for some $\Gamma_1, \Gamma_2, \Gamma_3$

$$\Gamma', \widetilde{d}: \widetilde{T} = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3 \tag{D.9}$$

$$\Gamma_1 \vdash a : l_o\widetilde{S} \tag{D.10}$$

$$\Gamma_2 \vdash \widetilde{c} : \widetilde{S} \tag{D.11}$$

$$\Gamma_3 \vdash R' \tag{D.12}$$

From (D.9) and (D.12), we infer (D.8). Then (D.7) can be derived by (D.13) below and (D.10).

$$\Gamma', \widetilde{d}: \widetilde{T} \vdash \sigma(y) : \Gamma(y), \text{ for every } y \text{ defined in } \Gamma \tag{D.13}$$

Moreover, (D.13) is valid because σ is a Γ-to-Γ' substitution. This completes the case and the proof.

$\square$

D.2. **Proof for Section 7.**

D.2.1. *The conditions for full abstraction w.r.t. BT.*

We reuse the conditions for BT in untyped case (Section 4). To adapt to the case for typed π, we assume that types are used 'implicitly' in the conditions (e.g., in $\asymp$), so as to maintain succinctness. For convenience, we reproduce the conditions in Figure 9 for use shortly.

Theorem D.9. *Let $[\![\,]\!]$ be an encoding of the λ-calculus into π-calculus with linear types, $\asymp$ a congruence on π-agents. Suppose there are a precongruence $\leq$ on π-agents and a type T_b assigned to the abstracted names of the encoding such that $\asymp$ is constrained by a type environment respecting T_b (i.e., typing the abstracted names of the encoding with T_b, and every term $[\![M]\!]$ has type $T_b \to \Diamond$). If the conditions in Figure 9 hold, then $[\![\,]\!]$ and $\asymp$ are fully abstract for BTs.*

Proof. Types stipulate the shape of a process (including the encoding of a λ term), and do not play a part in reductions, so the proof is conducted in a way similar to the case without types.

In the completeness proof, the important part is the one about the up-to-$\leq$-and-contexts technique, whereas in the soundness proof the important part is the one about the context-inverse properties. The proofs for these parts do not change with respect to the untyped case, since we use the same expansion relation $\preccurlyeq$ as before.   □

D.2.2. *Proof of Theorem 7.1.* Now we prove Theorem 7.1. Before beginning, we first present the encoding (Figure 2.a) rendered in the typed calculus π^l, as shown in Figure 10, whose design idea is explained in the course of proving the theorem.

Proof of Theorem 7.1. We recall in Figure 9 the soundness and completeness conditions for BT, adapted to typed calculi. As before, $\asymp$ and $\leq$ are relations on agents. Since we are now using typed π, these relations are adapted accordingly. We begin with some explanation and then proceed with the analysis of the conditions.

- To accommodate types, we designate the *basic* type T_b, which is defined as $T_b \stackrel{\text{def}}{=} \mu X. l_i(\sharp X, X)$. When encoding a λ term, T_b is used to type the *address* of its encoding (viz., the parameterized name of the λ term's encoding). That is, T_b is assumed, in the type environment, to be the type of the names used to instantiate the address of the encoding of λ terms.
- For $\asymp$, we use $\approx_{\text{bc}}^{\text{lin,asy}}$; this is parametric on a type environment Δ (assigning the type T_b to the names used as addresses in the encoding of λ terms). ;
- For $\leq$, we reuse the usual expansion $\preccurlyeq$.
- The original encoding (Figure 2.a) is modified using types, as shown in Figure 10. For every λ term M, its encoding $[\![M]\!]$ is of the type $T_b \to \Diamond$;
- We know that if two processes are related by a (finer) untyped behavioral equivalence, then they are also related by the typed one, e.g., $\approx_{\text{bc}}^{\text{lin,asy}}$.

We now analyze the satisfaction of the conditions under (linear) type environment. Those not mentioned can be done as for the untyped case. For instance, in soundness condition (6), variable x corresponds to an observable output, which emits the linear input

Let $\asymp$ and $\leq$ be relations on agents of asynchronous π-calculus with linear types.

We assume, in addition to the untyped version, that $\asymp$ is subject to type environment that respects T_b, that is, the "address" of the encoding of a λ term is assigned this type and every encoding has type $T_b \to \Diamond$.

Completeness conditions for BT

(1) $\asymp$ is a congruence and $\asymp \supseteq \geq$;
(2) $\leq$ is an expansion relation and is a plain precongruence;
(3) $\asymp$ validates the up-to-$\leq$-and-contexts technique;
(4) the variable contexts of $[\![\,]\!]$ are guarded;
(5) $[\![\,]\!]$ and $\geq$ validate rule β;
(6) if M is an unsolvable of order 0 then $[\![M]\!] \asymp [\![\Omega]\!]$;
(7) $[\![M]\!] \asymp [\![\Omega]\!]$ whenever M is unsolvable of order ∞.

Soundness conditions for BT

(1) $\asymp$ is a congruence, $\leq$ a plain precongruence;
(2) $\asymp \supseteq \geq$;
(3) $\asymp$ has the rendez-vous cancellation property;
(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 $\geq$;
(8) $[\![M]\!] \asymp [\![\Omega]\!]$ whenever M is unsolvable of order ∞;
(9) M solvable implies that the term $[\![\lambda x.\, M]\!]$ is unrelated by $\asymp$ to $[\![\Omega]\!]$ and to any term of the form $[\![x\widetilde{M}]\!]$.

FIGURE 9. The conditions for BT under types

$$
\begin{aligned}
[\![\lambda x.\, M]\!] &\stackrel{\text{def}}{=} (p)\, p(x, q).\, [\![M]\!]\langle q \rangle \\
[\![x]\!] &\stackrel{\text{def}}{=} (p)\, \overline{x}\langle p \rangle \\
[\![MN]\!] &\stackrel{\text{def}}{=} (p)\, (\boldsymbol{\nu} r : T_b',\, x : \sharp T_b\,) \Big([\![M]\!]\langle r \rangle \mid \overline{r}\langle x, p \rangle \mid \\
&\qquad\qquad !x(q).\, [\![N]\!]\langle q \rangle \Big) \quad \text{(for x fresh; $T_b' \stackrel{\text{def}}{=} l_\sharp(\sharp T_b, T_b), T_b \stackrel{\text{def}}{=} \mu X.\, l_i(\sharp X, X))$}
\end{aligned}
$$

FIGURE 10. Milner's encoding under linear typing

capability of an 'address' name, so each element in $\widetilde{M}$ can be retrieved. Then the arguments are similar to the untyped case. Below we proceed with a number of claims.

Claim 1. If $\Gamma \Vdash p : T$ in which $T \stackrel{\text{def}}{=} T_b$ (i.e., $T \stackrel{\text{def}}{=} \mu X.\, l_i(\sharp X, X)$), then $\Gamma \Vdash [\![\lambda x.\, \Omega]\!]\langle p \rangle \approx_{\text{bc}}^{\text{lin,asy}} [\![\Omega]\!]\langle p \rangle$.

Claim 2 (completeness condition (7) and soundness condition (8)). *If $\Gamma \Vdash p : T$ in which $T \stackrel{\mathrm{def}}{=} T_b$ (i.e., $T \stackrel{\mathrm{def}}{=} \mu X. l_i(\sharp X, X)$), then $\Gamma \Vdash [\![M]\!]\langle p\rangle \approx_{\mathrm{bc}}^{\mathrm{lin,asy}} [\![\Omega]\!]\langle p\rangle$ for every unsolvable M of order ∞.*

Claim 1 is valid because in Milner's encoding (Figure 2.a), the abstracted name (i.e., the address) of an encoded λ term has essentially the *linear* type in its very first place; that is, each such address is used only in one communication should there be an application of λ. Thus, if we set up a type environment that stipulates precisely the linearity of the address name, say p, an observer from outside will at most be able to obtain the linear output capability of p. Therefore, even if the input prefix in $[\![\lambda x. \Omega]\!]$ would be observed by providing an output like $\overline{p}\langle \widetilde{d}\rangle$, the resulting process on the other side, i.e., $[\![\Omega]\!]\langle p\rangle \mid \overline{p}\langle \widetilde{d}\rangle$, would not be observable at all, because p has been exhausted in its capabilities. Since a λ term may spawn local addresses, the linear type of p has to be a recursive type. Claim 2 can be similarly analyzed.

Claim 3 (completeness condition (6) and soundness condition (5)). *If M is an unsolvable of order 0 and $\Gamma \Vdash p : T_b$, then $\Gamma \Vdash [\![M]\!]\langle p\rangle \approx_{\mathrm{bc}}^{\mathrm{lin,asy}} [\![\Omega]\!]\langle p\rangle$.*

Claim 3 is true because in that case, $[\![M]\!]\langle p\rangle \approx [\![\Omega]\!]\langle p\rangle$, where we recall $\approx$ is the bisimilarity.

Claim 4 (completeness condition (3)). *Assume a type environment Γ. Then (under Γ) $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$ validates the up-to-$\preccurlyeq$-and-contexts technique.*

Claim 4 states somewhat that the up-to technique can be transplanted to the typed case. This result is like that for bisimulation from [18]. The part the expansion plays is similar. We thus sketch the argument. We first recall Definition 4.1 below.

Definition 4.1. 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 two 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.

Let $\mathcal{R}$ be a relation as in Definition 4.1, where $\asymp$ is $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$ and $\leq$ is $\preccurlyeq$. Define relation $\mathcal{S}$ as below.

$$\mathcal{S} \stackrel{\mathrm{def}}{=} \asymp \cup \{(P_1, P_2) \mid P_i \succcurlyeq C[\widetilde{P_i}](i = 1, 2) \text{ and } (\widetilde{P_1}, \widetilde{P_2}) \in \mathcal{R} \cup \asymp\}$$

Note $(\widetilde{P_1}, \widetilde{P_2}) \in \mathcal{R}$ stands for $(P_1^k, P_2^k) \in \mathcal{R}$ for all $P_i^k \in \widetilde{P_i}(i = 1, 2)$, $k \leqslant m$ and m is the number of holes in C. Obviously $\mathcal{R} \subseteq \mathcal{S}$, so we have to show that $\mathcal{S} \subseteq \asymp$ (that is, $\mathcal{S} \subseteq \approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$); one exploits the characterization of $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$ as $\approx_{\mathrm{ct}}^{\mathrm{lin,asy}}$ given in the Context Lemma D.8. The argument is routine analysis, except that the context C and relevant processes in $\mathcal{S}$ should be well-typed, according to the type environment Γ designated upon $\approx_{\mathrm{bc}}^{\mathrm{lin,asy}}$. Yet since type information does not have any effect on reductions, the analysis is similar to the untyped case (e.g. for $\approx$); see also [18].

Claim 5 (soundness condition (3)). *If*

$$\Gamma, a : S_1, (\widetilde{c} - \widetilde{b}) : \widetilde{T} \Vdash (\boldsymbol{\nu}\widetilde{b} : \widetilde{S})(\overline{a}\langle\widetilde{c}\rangle \mid b(r). P) \approx_{\mathrm{bc}}^{\mathrm{lin,asy}} (\boldsymbol{\nu}\widetilde{b} : \widetilde{S})(\overline{a}\langle\widetilde{c}\rangle \mid b(r). Q)$$

where $b \in \widetilde{b} \subseteq \widetilde{c}$, and a, b are fresh and neither of them is of linear type $l_\sharp T_2$ for some T_2, then (for some T_1)

$$\Gamma, r : T_1 \Vdash P \approx^{\mathrm{lin,asy}}_{\mathrm{bc}} Q$$

The cases when a or b is unobservable is not possible, which is why we assume they do not have the linear type $l_\sharp T_2$. Then the argument is similar to the case of $\approx$. That is, feed the two processes a concurrent (well-typed) process $a(\widetilde{x}).\overline{b}\langle q \rangle$ (in which $b \in \widetilde{x}$), and argue as for $\approx$ to obtain the equivalence between P and Q.

□