Towards an algebraic theory of typed mobile processes

Yuxin Deng [a] and Davide Sangiorgi [b]

[a] *INRIA and Université Paris 7, France*

[b] *Università di Bologna, Italy*

Abstract

The impact of types on the algebraic theory of the π-calculus is studied. The type system has capability types. They allow one to distinguish between the ability to read from a channel, to write to a channel, and both to read and to write. They also give rise to a natural and powerful subtyping relation.

Two variants of typed bisimilarity are considered, both in their late and in their early version. For both of them, proof systems that are sound and complete on the closed finite terms are given. For one of the two variants, a complete axiomatisation for the open finite terms is also presented.

---

1 Introduction

The π-calculus is the best known calculus of mobile processes. Its theory has been studied in depth [8,13]. Relevant parts of it are the algebraic theory and the type systems. Most of the algebraic theory has been developed on the untyped calculus; the results include proof systems or axiomatisations that are sound and complete on finite processes for the main behavioral equivalences: late and early bisimilarity, late and early congruence [10,5,6], open bisimilarity [12], testing equivalence [1]. Much of the research on types has focused on their behavioral effects. For instance, modifications of the standard behavioral equivalences have been proposed so as to take types into account [11,13].

In this paper, we study the impact of types on the algebraic theory of the π-calculus. Precisely, we study axiomatisations of the typed π-calculus. Although algebraic laws for typed calculi of mobile processes have been considered in the literature [13], we are not aware of any axiomatisation or proof system.

---

[1] Work supported by EU project PROFUNDIS

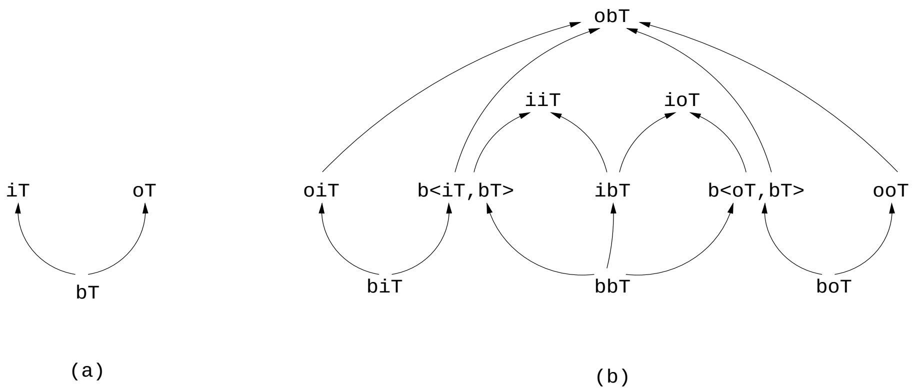

Fig. 1. An example of subtyping relation, with $T = \mathtt{unit}$

The type system that we consider has *capability types* (sometimes called I/O types) [11,4]. These types allow us to distinguish, for instance, the capability of using a channel in input from that of using the channel in output. A capability type shows the capability of a channel and, recursively, of the channels carried by that channel. For instance, a type $a : \mathtt{iob}T$ (for an appropriate type expression T) says that channel a can be used only in input; moreover, any channel received at a may only be used in output — to send channels which can be used both in input and in output. Thus, process $a(x).\bar{x}b.b(y).\bar{b}y.\mathbf{0}$ (sometimes the trailing $\mathbf{0}$ is omitted) is well-typed under the type assignment $a : \mathtt{iob}T, b : \mathtt{b}T$. We recall that $\bar{a}b.P$ is the output at a of channel b with continuation P, and that $a(x).P$ is an input at a with x a placeholder for channels received in the input whose continuation is P.

On calculi for mobility, capability types have emerged as one of the most useful forms of types, and one whose behavioral effects are most prominent. Capabilities are useful for protecting resources; for instance, in a client-server model, they can be used for preventing clients from using the access channel to the server in input and stealing messages to the server; similarly they can be used in distributed programming for expressing security constraints [4]. Capabilities give rise to *subtyping*: the output capability is contravariant, whereas the input capability is covariant. As an example, we show a subtyping relation in Figure 1, where an arrow indicates the subtyping relation between two related types. There are three forms of types for channel names: $\mathtt{i}T, \mathtt{o}S$ and $\mathtt{b}\langle T, S\rangle$, they give names the ability to receive values of type T, send values of type S, or to do both. We use $\mathtt{b}T$ as an abbreviation of $\mathtt{b}\langle T, T\rangle$. The depth of nesting of capabilities is 1 for all types in diagram (a), and 2 for all types in diagram (b). (The formal definitions of types and subtyping relation will be given in Section 2.1.) Subtyping is useful when the π-calculus is used for object-oriented programming, or for giving semantics to object-oriented languages.

To see why the addition of capability types has semantic consequences, consider

$$P \overset{\text{def}}{=} \nu c\, \overline{b}c.a(y).(\overline{y} \mid c) \qquad Q \overset{\text{def}}{=} \nu c\, \overline{b}c.a(y).(\overline{y}.c + c.\overline{y}).$$

These processes are not behaviorally equivalent in the untyped π-calculus. For instance, if the channel received at a is c, then P can terminate after 2 interactions with the external observer. By contrast, Q always terminates after 4 interactions with the observer. However, if we require that only the input capability of channels may be communicated at b, then P and Q are indistinguishable in any (well-typed) context. For instance, since the observer only receives the input capability on c, it cannot resend c along a: channels sent at a require at least the output capability (cf: the occurrence of $\overline{y}$). Therefore, in the typed setting, processes are compared w.r.t. an observer with certain capabilities (i.e., types on channels). Denoting with Δ these capabilities, then typed bisimilarity between P and Q is written $P \sim_\Delta Q$.

In the untyped π-calculus, labelled transition systems (LTS) are defined on processes; the transition $P \xrightarrow{\alpha} P'$ means that P can perform action α and then become P'. In the typed π-calculus, the information about the observer capabilities is relevant because the observer can only test processes on interactions for which the observer has all needed capabilities. Hence typed labelled transition systems (TLTS) are defined on configurations, and a configuration $\Delta \sharp P$ is composed of a process P and the observer capabilities Δ (we sometimes call Δ the external environment). A transition $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ now means that P can evolve into P' after performing an action α allowed by the environment Δ, which in turn evolves into Δ'.

Capability types have been introduced in [11]. A number of variants and extensions have then been proposed. We follow Hennessy and Riely's system [4], in which, in contrast with the system in [11]: (i) there are partial meet and join operations on types; (ii) the typing rule for the *matching* construct (the construct used for testing equality between channels) is very liberal, in that it can be applied to channels of arbitrary types (in [11] only channels that possess both the input and the output capability can be compared). While (i) only simplifies certain technical details, (ii) seems essential. Indeed, the importance of matching for the algebraic theory of the π-calculus is well-known (it is the main reason for the existence of matching in the untyped calculus).

Typed bisimilarity and the use of configurations for defining typed bisimilarity have been introduced in [2]. We follow a variant of them put forward by Hennessy and Rathke [3], because it uses the type system of [4] and includes the matching construct.

The main results in this paper are a proof system and an axiomatisation for typed bisimilarity ($\sim$). The proof system has a simple correctness proof but only works on the closed terms. The axiomatisation is for all finite processes. The bisimilarity $\sim$ is a variant of that in [3]. For the typed bisimilarity in [3] we provide a proof system for the closed terms, and an indirect axiomatisation of all terms that exploits the system of $\sim$. We have not been able to give a direct axiomatisation: the main difficulties are discussed in Section 5.1. All results are given for both the late and the early versions of the bisimilarities.

The axiomatisation and the proof systems are obtained by modifying some of the rules of the systems for the untyped π-calculus, and by adding a few new laws. The proofs of soundness and completeness, although follow the general schema of the proofs of the untyped calculus, have quite different details. An example of this is the treatment of fresh channels in input actions and the closure under injective substitutions, that we comment on below.

In the untyped π-calculus, the following holds:

If $P \sim Q$ and σ is injective on fn(P, Q), then $P\sigma \sim Q\sigma$.

Hence it is sufficient to consider all free channels in P, Q and *one* fresh channel when comparing the input actions of P and Q in the bisimulation game. This result is crucial in the algebraic theory of untyped calculi. For instance, in the proof system for (late) bisimilarity the inference rule for input is:

If $P\{b/x\} = Q\{b/x\}$ for all $b \in$ fn(P, Q, c), where c is a fresh channel, then $a(x).P = a(x).Q$.

For typed bisimilarity the situation is different. Take the processes

$$P \overset{\text{def}}{=} a(x : \texttt{ob}T).\bar{x}c.\bar{c} \qquad Q \overset{\text{def}}{=} a(x : \texttt{ob}T).\bar{x}c$$

and compare them w.r.t. an observer Δ. Consider what happens when the variable x is replaced by a fresh channel b, whose type in Δ is S. By the constraint imposed by types, S must be a subtype of the type $\texttt{ob}T$ for x (see Figure 1 (b)). Now, different choices for S will give different results. For instance, if S is $\texttt{ob}T$ itself, then the observer has no input capability on b, thus can not communicate with P and Q at b. That is, from the observer's point of view the output $\bar{b}c$ is not observable and the two derivative processes are equivalent. Similarly if S is $\texttt{bo}T$ then the output $\bar{c}$ is not observable. However, if S is $\texttt{bb}T$ then $\bar{b}c.\bar{c}$ is not equivalent to $\bar{b}c$, since all outputs become observable. This example illustrates the essential difficulties in formulating proof systems for typed bisimilarities:

(1) Subtyping appears in substitutions and changes the original type of a variable into one of its subtypes.
(2) The choice of this subtype is relevant for behavioral equivalence.
(3) Different subtypes may be incompatible (have no common subtype) with one another (for instance, $\texttt{bo}T$ and $\texttt{bb}T$ in the example above; they are both subtypes of $\texttt{ob}T$).

A consequence of (2) and (3), for instance, is that there is not a "best subtype", that is a single type with the property that equivalence under this type implies equivalence under any other types.

Another example of the consequences brought by types in the algebraic theory is the congruence rule for prefixes: we have to distinguish the cases in which the subject of the prefix is a channel from the case in which the subject is a variable. This is a rather subtle and technical difference, that is discussed in Section 4.

The rest of the paper is structured as follows. Section 2 presents the syntax, semantics and typed bisimilarity for a version of the π-calculus without parallelism. This small language already shows the major obstacles for axiomatisations and hence makes the presentation of our ideas neater. Section 3 sets up a proof system for closed terms. In Section 4 we axiomatize the typed bisimilarity for all finite terms. In Section 5 we examine other equivalences and relate their axiomatisations or proof systems to the results obtained in the previous sections. In Section 6 we show how the operator of parallel composition is admitted in the language. The effect on the axiomatisations is to add an expansion law to eliminate all occurrences of the operator. Finally Section 7 contains concluding remarks and possible directions for further research.

2 A fragment of the typed π-calculus

In this section we review the π-calculus (without parallelism), capability types, the usual operational semantics, typed labelled transition system as well as typed bisimilarity.

2.1 Standard operational semantics

We assume an infinite set of channels, ranged over by $a, b, \ldots$, and an infinite set of variables, ranged over by $x, y, \ldots$. We write $*$ for the unit value (we shall use `unit` as the only base type). Channels, variables and $*$ are the *names*, ranged over by $u, v, \ldots$. Below is the syntax of finite processes (also called terms).

$$\begin{aligned} P, Q &::= \mathbf{0} \bigm| \tau.P \bigm| u(x:T).P \bigm| \bar{u}v.P \bigm| P+Q \bigm| (\nu a:T)P \bigm| \varphi PQ \\ \varphi &::= [u=v] \bigm| \neg\varphi \bigm| \varphi \wedge \psi \end{aligned}$$

It has the usual constructors of finite monadic π-calculus: *inaction, prefix, sum* and *restriction.* The *match* constructor is replaced by a more general *condition*, ranged by φ, ψ etc, and produced by match, *negation* and *conjunction*. *Mismatching* like $[u \neq v]$ abbreviates $\neg[u = v]$. We also use $\vee$, which can be derived from $\wedge$ as usual. Here φPQ is an if-then-else construct on the boolean condition φ. We omit the else branch Q when it is $\mathbf{0}$. We have not included an operator of recursion because our main results in the paper are about proof systems and axiomatisations for finite terms. However, all results and definitions in Section 2 remain valid when recursion is added.

There is a channel-binding and a variable-binding operator. In $(\nu a : S)P$ the displayed occurrence of channel a is *binding* with *scope* P. In $u(x : T).P$ the occurrence of variable x is binding with scope P. An occurrence of a channel (resp. variable) in a process is *bound* if it lies within the scope of a binding occurrence of the channel (resp. variable). An occurrence of a channel or a variable in a process is *free* if it is not bound. We write $fn(P)$ and $fv(P)$ for the set of free names and the set of free variables, respectively, in P. We use $n(\varphi)$ for all names appearing in φ. When φ has no variables, $[\![\varphi]\!]$ denotes the boolean value of φ.

When $fv(P) \neq \emptyset$, P is an open term. We can make open terms closed by the use of *closing substitutions*, ranged over by $\sigma, \sigma', \sigma_i, \cdots$, which are substitutions mapping variables to channels and acting as identity on channels (thus similar to the concept of ground substitution used in term rewriting systems [14]). In the calculus, the distinction between channels and variables simplifies certain technical details; see for instance the discussion on the rules for substitutivity of prefixes in Section 4: the rules are different depending on whether the prefixes use channels or variables. (This is not the case in the untyped case: for instance, [10] does not distinguish between variables and channels, but it is quite straightforward to adapt the work to the case where there is such a distinction.)

The standard operational semantics is presented in the so-called late style in Table 1. The symmetric rule of sum is omitted. In a transition $P \xrightarrow{\alpha} P'$, the closed term P may become open in P' after performing the action α. As usual there are four forms of actions: τ (interaction), $a(x : T)$ (input), $\bar{a}b$ (free output), $\bar{a}(b : T)$ (bound output). We also use α to range over the set of extended prefixes, which contains the tau, the input prefixes, the output prefixes and the bound output prefixes. The bound output $\bar{u}(a : T).P$ is an abbreviation of $(\nu a : T)\bar{u}a.P$. We use $subj(\alpha)$, $bn(\alpha)$ and $n(\alpha)$ to stand for the subject, bound name and names of α. As usual we identify terms up to alpha-conversion.

We recall the capability types, as from [3,4]. The subtyping relation $<:$ and the typing rules for processes are displayed in Table 2. We write $T :: \texttt{TYPE}$ to mean that T is a well-defined type. There are three forms of types for channel names: $\texttt{i}T$, $\texttt{o}S$ and $\texttt{b}\langle T, S\rangle$, they give names the ability to receive values of type

Table 1
Transition rules

$$
\begin{array}{ll}
\mathsf{in}\ \dfrac{}{a(x:T).P \xrightarrow{a(x:T)} P} & \mathsf{out}\ \dfrac{}{\bar{a}b.P \xrightarrow{\bar{a}b} P} \\[2ex]
\mathsf{tau}\ \dfrac{}{\tau.P \xrightarrow{\tau} P} & \mathsf{sum}\ \dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \\[2ex]
\mathsf{true}\ \dfrac{[\![\varphi]\!] = \mathit{True} \quad P \xrightarrow{\alpha} P'}{\varphi\ P\ Q \xrightarrow{\alpha} P'} & \mathsf{false}\ \dfrac{[\![\varphi]\!] = \mathit{False} \quad Q \xrightarrow{\alpha} Q'}{\varphi\ P\ Q \xrightarrow{\alpha} Q'} \\[2ex]
\mathsf{open}\ \dfrac{P \xrightarrow{\bar{a}b} P' \quad a \neq b}{(\nu b:T)P \xrightarrow{\bar{a}(b:T)} P'} & \mathsf{res}\ \dfrac{P \xrightarrow{\alpha} P' \quad b \notin n(\alpha)}{(\nu b:T)P \xrightarrow{\alpha} (\nu b:T)P'}
\end{array}
$$

T, send values of type S, or to do both. For simplicity we often abbreviate $\mathtt{b}\langle T, T\rangle$ to $\mathtt{b}T$. As shown in [4], this extension to the original I/O types makes it possible to define two partial operators meet ($\sqcap$) and join ($\sqcup$). But the definitions of the two operators are rather long, so we do not repeat it and recommend the reader to consult Section 6 of [4].[2] Intuitively, the meet (resp. join) of T and S is the union (resp. intersection) of their capabilities.

Proposition 1 *Given types T_1, T_2 and S with $T_1 <: T_2$.*

(1) If $T_i \sqcap S$ are defined, for $i = 1, 2$, then $T_1 \sqcap S <: T_2 \sqcap S$;
(2) If $T_i \sqcup S$ are defined, for $i = 1, 2$, then $T_1 \sqcup S <: T_2 \sqcup S$;
(3) $T_1 \sqcap T_2 = T_1$;
(4) $T_1 \sqcup T_2 = T_2$.

Proof. Following the definitions of meet and join, the result is straightforward by structural induction on types. $\square$

We use Δ and Γ for type environments. A type environment Δ is a partial function from channels and variables to types; we write Δ_{c} and $\Delta_{\mathtt{v}}$ for the channel and variable parts of Δ, respectively. A type environment is undefined on infinitely many channels and variables (to make sure it can always be extended). We will often view, and talk about, Δ_{c} as a set of assignments of the form $a : T$, describing the value of Δ_{c} on all the channels on which Δ_{c} is defined. Similarly for $\Delta_{\mathtt{v}}$.

We use $dom(\Delta)$ to stand for the channels and variables on which Δ is well defined ($dom(\Delta)$ can be infinite). When $dom(\Delta) \cap dom(\Delta') = \emptyset$, we use Δ, Δ' to represent the union of Δ and Δ'. If $\Delta(u)$ is defined and takes the form $\mathtt{i}T$ or $\mathtt{b}\langle T, S\rangle$, then the predicate $\Delta(u)\downarrow_{\mathtt{i}}$ holds and we write $\Delta(u)_{\mathtt{i}}$ for T, otherwise

---

[2] The only modification we have made is as follows. If two channel types T and S have no common capability, then in our setting $T \sqcup S$ is undefined, while in [4] $T \sqcup S$ is defined to be a maximal type, which is a supertype of every channel type.

the predicate $\Delta(u)\not\downarrow_{\mathtt{i}}$ holds, indicating that Δ has no input capability on u. Similarly for $\Delta(u)_{\mathtt{o}}$ and $\Delta(u)\downarrow_{\mathtt{o}}$ (output capability). Notice that $\Delta(u)\downarrow_{\mathtt{i}}$ is covariant and $\Delta(u)\downarrow_{\mathtt{o}}$ is contravariant.

Proposition 2 *Suppose that* $u, v \in \mathit{dom}(\Delta)$ *and* $\Delta(u) <: \Delta(v)$*.*

(1) If $\Delta(v)\downarrow_{\mathtt{i}}$ *then* $\Delta(u)_{\mathtt{i}} <: \Delta(v)_{\mathtt{i}}$*;*
(2) If $\Delta(v)\downarrow_{\mathtt{o}}$ *then* $\Delta(v)_{\mathtt{o}} <: \Delta(u)_{\mathtt{o}}$*.*

The typing rules for processes are standard except for conditions. We impose no constraint for names appearing in conditions. The reason is discussed in the introduction. This mild modification does not affect the proofs of the following two results [11,4,3].

Lemma 3 (Substitution) *If* $\Gamma \vdash a : T$ *and* $\Gamma, x : T \vdash P$*, then* $\Gamma \vdash P\{a/x\}$*.*

Theorem 4 (LTS subject reduction) *Suppose* $\Gamma \vdash P$ *and* $P \xrightarrow{\alpha} P'$*.*

(1) if $\alpha = \tau$ *then* $\Gamma \vdash P'$*.*
(2) if $\alpha = a(x : T)$ *then* $\Gamma(a) \downarrow_{\mathtt{i}}$ *and* $\Gamma, x : T \vdash P'$*.*
(3) if $\alpha = \bar{a}b$ *then* $\Gamma(a) \downarrow_{\mathtt{o}}$*,* $\Gamma \vdash b : \Gamma(a)_{\mathtt{o}}$ *and* $\Gamma \vdash P'$*.*
(4) if $\alpha = \bar{a}(b : T)$ *then* $\Gamma(a) \downarrow_{\mathtt{o}}$*,* $\Gamma, b : T \vdash b : \Gamma(a)_{\mathtt{o}}$ *and* $\Gamma, b : T \vdash P'$*.*

2.2 Typed labelled transition system

Two known TLTS were presented in [2,3], both of them were given in early style. We prefer to write a TLTS in late style, so as to define the "late" version of bisimilarity in a concise way.

First we extend the subtyping relation to type environments, but only considering the types of channels. So $\Gamma <: \Delta$ means that $\Gamma_{\mathtt{v}} = \Delta_{\mathtt{v}}$, $\mathit{dom}(\Delta_{\mathtt{c}}) \subseteq \mathit{dom}(\Gamma_{\mathtt{c}})$ and $\Gamma_{\mathtt{c}}(a) <: \Delta_{\mathtt{c}}(a)$ for all $a \in \mathit{dom}(\Delta_{\mathtt{c}})$.

Definition 5 *A configuration is a pair* $\Delta \sharp P$ *which respects some type environment* Γ*, i.e.,* $\Gamma <: \Delta$ *and* $\Gamma \vdash P$*.*

The above definition implies the condition $\mathit{fv}(P) \subseteq \mathit{dom}(\Delta_{\mathtt{v}})$, because we have $\mathit{fv}(P) \subseteq \mathit{dom}(\Gamma_{\mathtt{v}})$ by $\Gamma \vdash P$ and $\mathit{dom}(\Gamma_{\mathtt{v}}) = \mathit{dom}(\Delta_{\mathtt{v}})$ by $\Gamma <: \Delta$. Since alpha-conversion is implicitly used throughout the paper, we may assume $\mathit{bn}(P) \cap \mathit{dom}(\Delta) = \emptyset$. Here there exists a mild difference from the definitions of configuration given in [2,3]. We do not require the environment to have knowledge of all the free channels used by P. The less knowledge it grasps, the weaker testing power it owns when observing the behaviour of P. In Table 3, we present a transition system built on this definition. In the premise of rule

Table 2
Types and typing rules

Types:

$$\frac{}{\mathtt{unit} :: \mathtt{TYPE}} \qquad \frac{T :: \mathtt{TYPE}}{\mathtt{i}T, \mathtt{o}T :: \mathtt{TYPE}} \qquad \frac{T, S :: \mathtt{TYPE} \quad S <: T}{\mathtt{b}\langle T, S\rangle :: \mathtt{TYPE}}$$

Subtyping:

$$\frac{}{T <: T} \qquad \frac{T <: T'}{\mathtt{i}T <: \mathtt{i}T'} \qquad \frac{T <: T'}{\mathtt{o}T' <: \mathtt{o}T}$$

$$\frac{T <: T'}{\mathtt{b}\langle T, S\rangle <: \mathtt{i}T'} \qquad \frac{T <: T'}{\mathtt{b}\langle S, T'\rangle <: \mathtt{o}T} \qquad \frac{T <: T' \quad S <: S'}{\mathtt{b}\langle T, S'\rangle <: \mathtt{b}\langle T', S\rangle}$$

Typing rules:

$$\frac{\Gamma(u) <: T}{\Gamma \vdash u : T} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P + Q} \qquad \frac{\Gamma, x : T \vdash P \quad \Gamma \vdash u : \mathtt{i}T}{\Gamma \vdash u(x : T).P}$$

$$\frac{}{\Gamma \vdash \mathbf{0}} \qquad \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a : T)P} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash v : T \quad \Gamma \vdash u : \mathtt{o}T}{\Gamma \vdash \bar{u}v.P}$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash \tau.P} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q \quad n(\varphi) \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash \varphi\; P\; Q}$$

Red, $P \xrightarrow{\tau} P'$ stands for the standard reduction relation of the π-calculus, as given in Table 1.

Using the partial meet operation, we can extend a type environment Δ to $\Delta \sqcap u : T$, which is just $\Delta, u : T$ if $u \notin \mathit{dom}(\Delta)$, otherwise it differs from Δ at name u because the capability of this name is extended to be $\Delta(u) \sqcap T$ (if $\Delta(u) \sqcap T$ is undefined, then so is $\Delta \sqcap u : T$). In this way we can define $\Delta_1 \sqcap \Delta_2$ as the meet of two environments Δ_1 and Δ_2. In rule Out, the process sends channel b to the environment, so the latter should be dynamically extended with the capability on b thus received. For this, we use the meet operator, and exploit the following property on types:

$$R <: T \textit{ and } R <: S \textit{ imply } T \sqcap S \textit{ defined and } R <: T \sqcap S$$

for any type T, S and R. (This property does not hold for the capability types as in [11].)

The next three fundamental lemmas describe various properties of the TLTS. They underpin many later results. The well-definedness of our TLTS is based

Table 3
Typed LTS

$$\textsf{Red}\ \frac{P \xrightarrow{\tau} P'}{\Delta \sharp P \xrightarrow{\tau} \Delta \sharp P'} \qquad \textsf{Out}\ \frac{\Delta(a) \downarrow_{\mathtt{i}}}{\Delta \sharp \bar{a}b.P \xrightarrow{\bar{a}b} \Delta \sqcap b : \Delta(a)_{\mathtt{i}} \sharp P}$$

$$\textsf{In}\ \frac{\Delta(a) \downarrow_{\mathtt{o}}}{\Delta \sharp a(x:T).P \xrightarrow{a(x:T)} \Delta, x : T \sharp P} \qquad \textsf{Open}\ \frac{\Delta \sharp P \xrightarrow{\bar{a}b} \Delta' \sharp P' \quad a \neq b}{\Delta \sharp (\nu b : T)P \xrightarrow{\bar{a}(b:T)} \Delta' \sharp P'}$$

$$\textsf{Res}\ \frac{\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P' \quad a \notin n(\alpha)}{\Delta \sharp (\nu a : T)P \xrightarrow{\alpha} \Delta' \sharp (\nu a : T)P'} \qquad \textsf{Sum}\ \frac{\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'}{\Delta \sharp P + Q \xrightarrow{\alpha} \Delta' \sharp P'}$$

$$\textsf{True}\ \frac{[\![\varphi]\!] = \mathit{True} \quad \Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'}{\Delta \sharp \varphi PQ \xrightarrow{\alpha} \Delta' \sharp P'} \qquad \textsf{False}\ \frac{[\![\varphi]\!] = \mathit{False} \quad \Delta \sharp Q \xrightarrow{\alpha} \Delta' \sharp Q'}{\Delta \sharp \varphi PQ \xrightarrow{\alpha} \Delta' \sharp Q'}$$

---

on Lemma 6. The close relationship between processes and configurations is reflected by their corresponding transitions, as can be seen in Lemma 7. Finally Lemma 8 says that the more capabilities an environment owns, the more behaviours it can observe on a process.

Lemma 6 (TLTS subjection reduction) *If* $\Delta\sharp P$ *is a configuration which respects* Γ *and* $\Delta\sharp P \xrightarrow{\alpha} \Delta'\sharp P'$, *then* $\Delta'\sharp P'$ *is also a configuration, respecting* Γ', *where*

(1) if $\alpha = \tau$ *then* $\Delta' = \Delta$ *and* $\Gamma' = \Gamma$.
(2) if $\alpha = a(x : T)$ *then* $\Delta' = \Delta, x : T$ *and* $\Gamma' = \Gamma, x : T$.
(3) if $\alpha = \bar{a}b$ *then* $\Delta' = \Delta \sqcap b : \Delta(a)_{\mathtt{i}}$ *and* $\Gamma' = \Gamma$.
(4) if $\alpha = \bar{a}(b : T)$ *then* $\Delta' = \Delta, b : \Delta(a)_{\mathtt{i}}$ *and* $\Gamma' = \Gamma, b : T$.

Proof. By induction on depth of inference. LTS subject reduction theorem is needed. □

Lemma 7 *Suppose that* $\Delta\sharp P$ *is a configuration.*

(1) $\Delta\sharp P \xrightarrow{\tau} \Delta\sharp P'$ *iff* $P \xrightarrow{\tau} P'$.
(2) $\Delta\sharp P \xrightarrow{a(x:T)} \Delta, x : T\sharp P'$ *iff* $\Delta(a) \downarrow_{\mathtt{o}}$ *and* $P \xrightarrow{a(x:T)} P'$.
(3) $\Delta\sharp P \xrightarrow{\bar{a}b} \Delta \sqcap b : \Delta(a)_{\mathtt{i}}\sharp P'$ *iff* $\Delta(a) \downarrow_{\mathtt{i}}$ *and* $P \xrightarrow{\bar{a}b} P'$.
(4) $\Delta\sharp P \xrightarrow{\bar{a}(b:T)} \Delta, b : \Delta(a)_{\mathtt{i}}\sharp P'$ *iff* $\Delta(a) \downarrow_{\mathtt{i}}$ *and* $P \xrightarrow{\bar{a}(b:T)} P'$.

Proof. By induction on depth of inference. □

Lemma 8 *Suppose that* $\Delta\sharp P \xrightarrow{\alpha} \Delta'\sharp P'$, $\Gamma <: \Delta$ *and* $\Gamma\sharp P$ *is a configuration. Then* $\Gamma\sharp P \xrightarrow{\alpha} \Gamma'\sharp P'$ *and* $\Gamma' <: \Delta'$.

Proof. Straightforward by using the preceding lemma. □

When comparing two typed actions, to require them to be syntactically the same is too restrictive. For example one would not be able to say $(\nu a : T_1)\bar{u}a$ is bisimilar to $(\nu a : T_2)\bar{u}a$ under the environment $\Delta = u : \mathtt{bob}T$, where $T_1 = \mathtt{bo}T, T_2 = \mathtt{bb}T$. Therefore we do not check types in the bisimulation game. We shall write $|\,\alpha\,|$ for the action α where its type annotations have been stripped off.

$P \sim_\Delta Q$ reads "P and Q are bisimilar under type environment Δ". The type environment Δ is used as follows: Δ_{c} shows the channels that are known to the external observer testing the processes in the bisimulation game, and the types with which the observer is allowed to use such channels. By contrast, Δ_{v} shows the set of variables that may appear free in the processes and the types for these variables show how the observer can instantiate such variables (in closing substitutions). Therefore: the channels of Δ_{c} are to be used by the observer, with the types indicated in Δ_{c}; the variables in Δ_{v} are to be used by the processes, but the observer can instantiate them following the types indicated in Δ_{v}.

A process is *closed* if it does not have free variables; similarly a type environment is closed if it is only defined on channels. Otherwise, processes and type environments are *open*. We first define $\sim_\Delta$ on the closed terms, then on the open terms. Bisimilarity is given in the "late" style [13]; we consider the "early" style in Section 5.2.

Definition 9 *A family of symmetric binary relations over closed terms, indexed by type environments, and written $\{\mathcal{R}_\Delta\}_\Delta$, is a* typed bisimulation *whenever $P\ \mathcal{R}_\Delta\ Q$ implies that, for two configurations $\Delta \sharp P$ and $\Delta \sharp Q$,*

(1) if $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ and α is not an input action, then for some Q', $\Delta \sharp Q \xrightarrow{\beta} \Delta' \sharp Q'$, $|\alpha| = |\beta|$ and $P'\ \mathcal{R}_{\Delta'}\ Q'$.

(2) if $\Delta \sharp P \xrightarrow{a(x:T)} \Delta' \sharp P'$, then for some Q', $\Delta \sharp Q \xrightarrow{a(x:S)} \Delta'' \sharp Q'$ and for all b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$ it holds that $P'\{b/x\}\ \mathcal{R}_\Delta\ Q'\{b/x\}$.

Two processes P and Q are typed Δ-bisimilar, written $P \sim_\Delta Q$, if there exists a typed bisimulation $\{\mathcal{R}_\Delta\}_\Delta$ such that $P\ \mathcal{R}_\Delta\ Q$.

The difference w.r.t. typed bisimilarity as in [2,3] is that, in the input clause, the type environment Δ is not extended. In other words, the knowledge of the external observer does not change through interactions with the process in which the value transmitted is supplied by the observer itself (by contrast, the knowledge does change when the value is supplied by the process; cf: rule Out in Table 3). Therefore $\sim_\Delta$ is optimised for reasoning on finite systems. To deal with infinite systems, it is more suitable to use the alternative equivalence where the environment can be extended. We shall turn to this topic in Section 5.1.

Definition 10 *Two processes P and Q are bisimilar under the environment $\Delta = \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$, written $P \sim_{\Delta} Q$, if $\Delta \sharp P$, $\Delta \sharp Q$ are configurations and, for all $\widetilde{b}$ with $\Delta_{\mathsf{c}} \vdash \widetilde{b} : \widetilde{T}$, it holds that $P\{\widetilde{b}/\widetilde{x}\} \sim_{\Delta_{\mathsf{c}}} Q\{\widetilde{b}/\widetilde{x}\}$.*

The intuition behind the above definition is that channels are capabilities while variables are obligations of the environment. The environment is obliged to fill in the variables at the specified types. Once the obligations are determined, they cannot be strengthened or weakened. That's why variables are invariant in the subtyping relation on type environments given before.

Below we report three basic properties of typed bisimilarity.

Lemma 11 *If $P \sim_{\Delta} Q$ and $\Delta <: \Delta'$, then $P \sim_{\Delta'} Q$.*

Proof. By Lemma 7, 8 and the definition of typed bisimilarity. $\Box$

The intuition behind this lemma is quite clear. When two processes exhibit similar behaviours under an environment with stronger discriminating power, they are also indistinguishable by a weaker environment. In the presence of distinction between channels and variables, we have the following interesting property for typed bisimilarity.

Lemma 12 *If $P \sim_{\Delta,x:T} Q$ and $S <: T$ then $P \sim_{\Delta,x:S} Q$.*

Proof. It follows easily from the definition of typed bisimilarity on open terms. $\Box$

As we said before in the introduction, generally speaking, typed behavioural equivalences are not closed under injective substitutions. Nevertheless, if a substitution only maps channels and variables to other channels and variables of the same types respectively (called *type-preserving substitution*), we do have the property seen in untyped π-calculus, as expressed by the lemma below. (With a slight abuse of notation, here we use σ to stand for type-preserving substitutions.)

Lemma 13 *If $P \sim_{\Delta} Q$ then $P\sigma \sim_{\Delta\sigma} Q\sigma$ for σ injective on $fn(P, Q) \cup dom(\Delta)$ and $\Delta\sigma$ is the type environment which maps $\sigma(a)$ to $\Delta(a)$ for all $a \in dom(\Delta)$.*

Proof. Similar to the proof in untyped setting. It follows from the fact that $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ implies $\Delta\sigma \sharp P\sigma \xrightarrow{\alpha\sigma} \Delta'\sigma \sharp P'\sigma$, for injective type-preserving substitution σ. $\Box$

Since all processes are finite, and we do not use recursive types, in $P \sim_{\Delta} Q$, the environment Δ can always be taken to be *finite* (i.e., defined only on a finite number of channels and variables): it is sufficient that Δ has enough names fresh w.r.t. P and Q, for all relevant types. This can be proved with a construction similar to that in Lemma 34. In the remainder of the paper all type environments are assumed to be finite. (If Δ is infinite, our proof systems in Section 3 and 5.1.1 remain sound and complete; the axiom system in Section 4 is still sound, but its completeness proof relies on the finiteness of Δ.) We should stress, however, that all results and definitions presented up to this section are also valid for non-finite processes (i.e., processes extended with recursion) and for infinite type environment.

3 Proof system for the closed terms

In this section we present a proof system for the closed terms.

The proof system $\mathcal{P}$ for typed bisimilarity is composed of all inference rules and axioms in Table 4. Whenever we write $P =_\Delta Q$ it is intended that both $\Delta\sharp P$ and $\Delta\sharp Q$ are configurations (see Definition 5 and the explanations immediately follow the definition), and in this section P, Q are deemed to be closed terms. The rules are divided into six groups, namely those for: substitutivity, sums, looking up the type environment, conditions, restrictions and alpha-conversion. The rules that are new or different w.r.t. those of the untyped π-calculus are marked with an asterisk.

Tin* shows that an input prefix is not observable if the observer has no output capability on the subject of the input. This comes as no surprise because the only means that the observer uses for testing a process is to communicate with it. When no communication happens, he/she simply regards the process being tested as **0**. **Tout*** is the symmetric rule, for output. **Twea*** gives us weakening for type environments, corresponding to Lemma 11. In **Ires***, the side condition $a \notin \mathit{dom}(\Delta)$ is added for the sake of clarity, but formally it is not needed because of the definition of configurations and our convention on bound names. Note that different types T_1, T_2 are used for the processes in the conclusion. We cannot replace **Ires*** with two simpler rules such as

- If $P =_\Delta Q$ then $(\nu a : T)P =_\Delta (\nu a : T)Q$
- $(\nu a : T_1)P =_\Delta (\nu a : T_2)P$,

for equalities like $(\nu b : \mathtt{bi}T)\bar{a}b.b(x : \mathtt{i}T).\mathbf{0} =_{a:\mathtt{iob}T} (\nu b : \mathtt{bo}T)\bar{a}b.b(x : \mathtt{o}T).\mathbf{0}$ could not be derived (due to the constraints given by the well-typedness of processes). Similarly for rule **Iinc***.

Iinc* and **Iout*** are the rules for substitutivity for input and output prefixes.

Table 4
The proof system $\mathcal{P}$ for the closed terms

Iinc*	If $P\{b/x\} =_\Delta Q\{b/x\}$ for all b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$ then $a(x : T_1).P =_\Delta a(x : T_2).Q$.
Iout*	If $P =_{\Delta \sqcap b:\Delta(a)_{\mathtt{i}}} Q$ then $\bar{a}b.P =_\Delta \bar{a}b.Q$
Itau	If $P =_\Delta Q$ then $\tau.P =_\Delta \tau.Q$
Isum	If $P =_\Delta Q$ then $P + R =_\Delta Q + R$
Ires*	If $P =_\Delta Q$ then $(\nu a : T_1)P =_\Delta (\nu a : T_2)Q \quad a \notin dom(\Delta)$
S1	$P + \mathbf{0} =_\Delta P$
S2	$P + P =_\Delta P$
S3	$P + Q =_\Delta Q + P$
S4	$P + (Q + R) =_\Delta (P + Q) + R$
Tin*	If $\Delta(a)\not\downarrow_{\mathsf{o}}$ then $a(x : T).P =_\Delta \mathbf{0}$
Tout*	If $\Delta(a)\not\downarrow_{\mathtt{i}}$ then $\bar{a}u.P =_\Delta \mathbf{0}$
Twea*	If $P =_\Delta Q$ and $\Delta <: \Delta'$ then $P =_{\Delta'} Q$
Ca	$\varphi\ P\ Q =_\Delta P \quad \text{if } [\![\varphi]\!] = \mathit{True}$
Cb	$\varphi\ P\ Q =_\Delta Q \quad \text{if } [\![\varphi]\!] = \mathit{False}$
R1	$(\nu a : T)\mathbf{0} =_\Delta \mathbf{0}$
R2	$(\nu a : T)\alpha.P =_\Delta \mathbf{0} \quad \text{if } subj(\alpha) = a$
R3	$(\nu a : T)(\nu b : S)P =_\Delta (\nu b : S)(\nu a : T)P$
R4	$(\nu a : T)(P + Q) =_\Delta (\nu a : T)P + (\nu a : T)Q$
R5	$(\nu a : T)\alpha.P =_\Delta \alpha.(\nu a : T)P \quad \text{if } a \notin n(\alpha)$
A	$P =_\Delta Q$ if P alpha-equivalent to Q

In **Iinc***, the well-definedness of the two configurations $\Delta \sharp a(x : T_1).P$ and $\Delta \sharp a(x : T_2).Q$ implies the condition: $\Delta(a)_{\mathsf{o}} <: T_i$ for $i = 1, 2$. In **Iout***, the observer knowledge of the type of b may increase when the processes emit b themselves (for the type under which b is emitted is composed with the possible type of b in Δ).

Compared with the proof system for untyped π-calculus [10], **Tin*** and **Tout*** are the main differences.

Theorem 14 (Soundness of $\mathcal{P}$) *If $\mathcal{P} \vdash P =_\Delta Q$ then $P \sim_\Delta Q$.*

Proof. By constructing appropriate bisimulations. □

The completeness proof uses a standard strategy. By using the axioms **S1-4**, **R1-5** and **Ca-b**, we can transform each closed term into a canonical form $\sum_i \alpha_i.P_i$. If P and Q are bisimilar, their canonical forms P' and Q' are provably equal by induction on the depth of $P' + Q'$.

Theorem 15 (Completeness of $\mathcal{P}$) *If $P \sim_\Delta Q$ then $\mathcal{P} \vdash P =_\Delta Q$, where P and Q be closed terms.*

Proof. This proof differs from the completeness proof of untyped π-calculus [9] in one place: instead of showing that each summand of P is provably equivalent to a summand in Q, we only require that each *active summand* of P is matched by an active summand of Q, and vice versa. By active summand, we mean that the prefix can perform actions allowed by the environment Δ. More precisely, if $a_i(x_i : T_i).P_i$ is a summand of P and $\Delta(a_i)\downarrow_{\mathsf{o}}$ then this is an active input prefix. Similarly for output prefixes. Inactive summand is provably equivalent to **0** by **Tin*** and **Tout***, thus can be consumed by **S1**. After finite steps of transformation, we have $\mathcal{P} \vdash P =_\Delta \sum_{i=1}^{n} \alpha_i.P_i$ and $\mathcal{P} \vdash Q =_\Delta \sum_{j=1}^{m} \beta_j.Q_j$, where all summands in P and Q are active.

Suppose that $\alpha_i = \bar{a}(b : T_1)$. Then $\Delta \sharp P \xrightarrow{\bar{a}(b:T_1)} \Delta, b : \Delta(a)_{\mathtt{i}} \sharp P_i$. Hence there is some $\beta_j = \bar{a}(b : T_2)$ such that $P_i \sim_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. Since the depth of $P_i + Q_j$ is less than the depth of $P + Q$, we can use induction hypothesis to derive $\mathcal{P} \vdash P_i =_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. By **A** we assume that the bound name $b \notin dom(\Delta)$, so $\Delta, b : \Delta(a)_{\mathtt{i}} = \Delta \sqcap b : \Delta(a)_{\mathtt{i}}$. Therefore we have $\mathcal{P} \vdash \bar{a}b.P_i =_\Delta \bar{a}b.Q_j$ by **Iout***, and furthermore $\mathcal{P} \vdash \bar{a}(b : T_1).P_i =_\Delta \bar{a}(b : T_2).Q_j$ by **Ires***.

Suppose that $\alpha_i = a(x : T_1)$. Then $\Delta \sharp P \xrightarrow{a(x:T_1)} \Delta' \sharp P_i$. There must exist a $\beta_j = a(x : T_2)$ such that $P_i\{b/x\} \sim_\Delta Q_j\{b/x\}$, for all b s.t. $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$. Now observe that the depth of $P_i\{b/x\} + Q_j\{b/x\}$ is less than the depth of $P + Q$, thus it follows from induction hypothesis that $\mathcal{P} \vdash P_i\{b/x\} =_\Delta Q_j\{b/x\}$. Using **Iinc*** we infer that $\mathcal{P} \vdash a(x : T_1).P_i =_\Delta a(x : T_2).Q_j$.

Other cases can be analyzed similarly. As a result, each active summand of P is provably equivalent to some active summand of Q. Symmetric arguments also hold. □

4 Axioms for typed bisimilarity

In this section we give an axiom system for typed bisimilarity and prove its soundness and completeness. This axiomatisation is for all finite terms of the language given in Section 2, including both open and closed terms.

4.1 The axiom system

The axiom system $\mathcal{A}$ for typed bisimilarity is presented in Table 5. Roughly speaking, it is obtained from $\mathcal{P}$ by adding some axioms for dealing with conditions. In open terms usually the conditions cannot be simply eliminated by **Ca-b**, so we need the axioms **C1-7** and **R6-7** to manipulate them. We use the notation $\varphi \Rightarrow \psi$ to mean that φ logically implies ψ; in **C1** the condition $\varphi \iff \psi$ means that φ and ψ are logically equivalent. In view of **C3** and **R6**, axiom **R1** is redundant. The rule **Iinc*** of $\mathcal{P}$ now becomes the concise axiom **Iin*** in $\mathcal{A}$. **Tvar*** shows that a variable can only be instantiated with channels that in the type environment have types compatible with that of the variable. **Tpre*** is used to replace names underneath a match. It implies, in the presence of other axioms of $\mathcal{A}$, a more powerful axiom: $[x = a]P =_{\Delta} [x = a]P\{a/x\}$ if $\Delta(a) <: \Delta(x)$, which substitutes through P. In the untyped setting, **Tpre*** has no side condition. Here we need one to ensure well-typedness of the process resulting from the substitution, since the names in the match can have arbitrary — and possibly unrelated — types.

The following axioms and rules are derivable from **{S1-S4, C1-C6, Tvar*}**. More derived rules are given in Appendix A.

C8	$P =_{\Delta} \varphi P + \neg\varphi P$	**C9**	$\varphi PQ =_{\Delta} \varphi P + \neg\varphi Q$
C10	$[\varphi \vee \psi]P =_{\Delta} \varphi P + \psi P$	**C11**	$\varphi(P + Q) =_{\Delta} \varphi P + \varphi Q$
Cnn1	$[a = b]P =_{\Delta} \mathbf{0}$ if $a \neq b$	**Tvn1**	$[x = a]P =_{\Delta} \mathbf{0}$ if $a \notin dom(\Delta)$
Cnn2	$[a \neq b]P =_{\Delta} P$ if $a \neq b$	**Tvn2**	$[x \neq a]P =_{\Delta} P$ if $a \notin dom(\Delta)$
Tv1	$P =_{\Delta,x:T} \mathbf{0}$ if there exists no $a \in dom(\Delta)$ s.t. $\Delta(a) <: T$		

Note that in **Iin*** and **Iout***, the free names of the input and output prefixes are channels rather than variables. Below we discuss:

(1) the unsoundness of the rules in which (some or all) the channels are replaced by variables;
(2) other rules, that are valid for variables;
(3) why these other rules are not needed in the axiom system.

Intuitively the reason for (1) is the different usage of channels and variables that appear in a type environment: the information on channels tells us how these channels are to be used by the *external environment*, while the information on variables tells us how these variables are to be instantiated inside the *tested processes*.

To see that **Iin*** is unsound when the subject of the prefix is a variable, take

Table 5
The axiom system $\mathcal{A}$

Iin*	If $P =_{\Delta,x:\Delta(a)_{\mathtt{o}}} Q$ then $a(x:T_1).P =_\Delta a(x:T_2).Q$
Icon	If $P =_\Delta Q$ then $\varphi P =_\Delta \varphi Q$
Tvar*	$[x \neq a_1]\cdots[x \neq a_m]P =_\Delta \mathbf{0}$ if $\{b \in \mathit{dom}(\Delta_{\mathtt{c}}) \mid \Delta(b) <: \Delta(x)\} \subseteq \{a_1,\cdots,a_m\}$
Tpre*	$[x=a]\alpha.P =_\Delta [x=a](\alpha\{a/x\}).P \quad \text{if } \Delta(a) <: \Delta(x)$
C1	$\varphi\ P =_\Delta \psi\ P \quad \text{if } \varphi \Longleftrightarrow \psi$
C2	$[a=b]P =_\Delta [a=b]Q \quad \text{if } a \neq b$
C3	$\varphi\ P\ P =_\Delta P$
C4	$\varphi\ P\ Q =_\Delta \neg\varphi\ Q\ P$
C5	$\varphi(\psi P) =_\Delta [\varphi \wedge \psi]P$
C6	$\varphi\ (P_1+P_2)\ (Q_1+Q_2) =_\Delta \varphi\ P_1\ Q_1 + \varphi\ P_2\ Q_2$
C7	$\varphi\ (\alpha.P) =_\Delta \varphi\ (\alpha.\varphi P) \quad \text{if } \mathit{bn}(\alpha) \cap n(\varphi) = \emptyset$
R6	$(\nu a:T)[a=u]P =_\Delta \mathbf{0} \quad \text{if } a \neq u$
R7	$(\nu a:T)[u=v]P =_\Delta [u=v](\nu a:T)P \quad \text{if } a \neq u,v$
$\mathcal{P}\backslash\{$**Iinc*, Ca-b, R1**$\}$	

$\Delta_{\mathtt{c}} \overset{\text{def}}{=} a:\mathtt{bo}T, b:\mathtt{o}T$ and $\Delta \overset{\text{def}}{=} \Delta_{\mathtt{c}}, x:\mathtt{b}\langle \mathtt{o}T, \mathtt{b}T\rangle$. Then we have

$$[y=b]\tau \sim_{\Delta,y:\Delta(x)_{\mathtt{o}}} \mathbf{0}$$

because $\Delta(x)_{\mathtt{o}} = \mathtt{b}T$ and no c in Δ satisfies the condition $\Delta_{\mathtt{c}} \vdash c:\mathtt{b}T$ and can therefore instantiate y. However,

$$x(y:\mathtt{o}T).[y=b]\tau \not\sim_\Delta x(y:\mathtt{o}T).\mathbf{0}.$$

To see this, let us look at the possible closing substitutions. In $\mathit{dom}(\Delta_{\mathtt{c}})$, a is the only channel satisfying $\Delta_{\mathtt{c}} \vdash a:\Delta(x)$, and so the only substitution we need to consider is $\{a/x\}$. After applying this substitution, the resulting closed terms are not bisimilar:

$$a(y:\mathtt{o}T).[y=b]\tau \not\sim_\Delta a(y:\mathtt{o}T).\mathbf{0}$$

This holds because the observer can send b along a and, after the communication, y is instantiated to be b, thus validating the condition $y=b$ and liberating the prefix τ. When the subject of the prefix is a variable, the following rule is needed in place of **Iin***:

$$\textbf{Iv1} \quad \text{If } P =_{\Delta,y:\Delta(x)_{\mathtt{i}}} Q \text{ then } x(y:T_1).P =_\Delta x(y:T_2).Q$$

In rule **Iout***, both the subject and object of the output prefix are channels. The rule is also valid when the object is a variable. However, it is not valid if the subject is a variable. As a counterexample, let $\Delta_{\mathtt{c}} \overset{\text{def}}{=} a:\mathtt{i}T, b:\mathtt{bb}T$ and $\Delta \overset{\text{def}}{=} \Delta_{\mathtt{c}}, x:\mathtt{b}\langle \mathtt{i}T, \mathtt{b}T\rangle$. Then we have $a \sim_{\Delta \sqcap a:\mathtt{i}T} \mathbf{0}$ but $\bar{x}a.a \not\sim_\Delta \bar{x}a.\mathbf{0}$ because, under the substitution $\{b/x\}$, it holds that $\bar{b}a.a \not\sim_\Delta \bar{b}a.\mathbf{0}$. When the subject of the prefix is a variable, we need the following rule:

$$\textbf{Iv2}\quad \text{If } P =_{\Delta\sqcap v:\Delta(x)_\circ} Q \text{ then } \bar{x}v.P =_\Delta \bar{x}v.Q$$

We show, by means of an example, why rules **Iin*** and **Iout*** are sufficient in the axiom system (rules **Iv1** and **Iv2** are derivable, see Appendix A). Consider the equality

$$x(y : \mathtt{ii}T).y \sim_\Delta x(y : \mathtt{io}T).\mathbf{0}$$

where $\Delta \overset{\text{def}}{=} a : \mathtt{bib}T, b : \mathtt{ib}T, x : \mathtt{bib}T$. First, we infer

$$y =_{\Delta'} \mathbf{0} \quad \text{for } \Delta' = \Delta, y : \mathtt{ib}T \qquad (1)$$

proceeding as follows:

$$\begin{aligned}
y \quad &=_{\Delta'} \quad [y = b]y + [y \neq b]y && \text{by } \mathbf{C8}\\
&=_{\Delta'} \quad [y = b]y && \text{by } \mathbf{Tvar*}\\
&=_{\Delta'} \quad [y = b]b && \text{by } \mathbf{Tpre*}\\
&=_{\Delta'} \quad [y = b]\mathbf{0} && \text{by } \mathbf{Tin*}\\
&=_{\Delta'} \quad \mathbf{0} && \text{by } \mathbf{C3}
\end{aligned}$$

Then we derive $x(y : \mathtt{ii}T).y =_\Delta x(y : \mathtt{io}T).\mathbf{0}$ in a similar way:

$$\begin{aligned}
& x(y : \mathtt{ii}T).y \\
=_\Delta \quad & [x = a]x(y : \mathtt{ii}T).y + [x \neq a]x(y : \mathtt{ii}T).y && \text{by } \mathbf{C8}\\
=_\Delta \quad & [x = a]x(y : \mathtt{ii}T).y && \text{by } \mathbf{Tvar*}\\
=_\Delta \quad & [x = a]a(y : \mathtt{ii}T).y && \text{by } \mathbf{Tpre*}\\
=_\Delta \quad & [x = a]a(y : \mathtt{io}T).\mathbf{0} && \text{by } \mathbf{(1),\ Iin*,\ Icon}\\
=_\Delta \quad & x(y : \mathtt{io}T).\mathbf{0} && \text{by } \mathbf{Tpre*,\ Tvar*,\ C8}
\end{aligned}$$

4.2 Soundness and completeness

The soundness of the axioms displayed in Table 5, and therefore of $\mathcal{A}$, is easy to be verified.

Theorem 16 (Soundness of $\mathcal{A}$) *If $\mathcal{A} \vdash P =_\Delta Q$ then $P \sim_\Delta Q$.*

The remainder of the section is devoted to proving the completeness of $\mathcal{A}$. The schema of the proof is similar to that for the untyped π-calculus [10]. The details, however, are quite different. An example of this is the manipulation of terms underneath input and output prefixes mentioned above. We discuss below another example, related to the issue of invariance of bisimilarity under injective substitutions. In the untyped case, the process $x \mid \overline{a}$ (the operational semantics of parallel composition is standard and will be given in Section 6) is equal to $x.\overline{a} + \overline{a}.x + \tau$ when x is instantiated to a, to $x.\overline{a} + \overline{a}.x$ otherwise. This can be expressed by expanding the process by means of conditions: that is, using conditions to make a case analysis on the possible values that the variable may take. Thus, $x \mid \overline{a}$ is expanded to $[x=a](x \mid \overline{a}) + [x \neq a](x \mid \overline{a})$. Now, underneath $[x=a]$ we know that x will be a, and therefore $x \mid \overline{a}$ can be rewritten as $x.\overline{a}+\overline{a}.x+\tau$, whereas underneath $[x \neq a]$ we know that x will not be a and therefore $x \mid \overline{a}$ can be rewritten as $x.\overline{a}+\overline{a}.x$. In general, the expansion of a process with a free variable x produces a summand $[x \neq a_1] \cdots [x \neq a_n]P$ where $a_1, \cdots, a_n$ are all channels (different from x) that appear free in P. The mismatch $[x \neq a_1] \cdots [x \neq a_n]$ tells us that x in P will be instantiated to a fresh channel, which is sufficient for all manipulations of P involving x, since bisimulation is invariant under injective substitutions. In the typed calculus, by contrast, knowing that x is fresh may not be sufficient: we may also need the information on the type with which x will be instantiated. This type may be different from the type T of x in the type environment: x could be instantiated to a fresh channel whose type is a *subtype* of T (the behavioral consequences of this type information can be seen in the example at the end of Section 5.1). We have therefore adopted a strategy different from that in the proof for untyped calculi: rather than manipulating processes that begin with "complete" sequences of mismatches — as in the untyped case — we try to cancel them, using rule **Tvar***; further, the conditional expansion of a process takes into account also the names that appear in the type environment.

Definition 17 *A condition φ is satisfiable if $[\![\varphi\sigma]\!] = True$ for some closing substitution σ. Given a set of names V, a condition φ is complete on V if for some equivalence relation $\mathcal{E}$ on V, called the equivalence relation corresponding to φ, it holds that $\varphi \Rightarrow [u = v]$ iff $u\mathcal{E}v$ and $\varphi \Rightarrow [u \neq v]$ iff $\neg(u\mathcal{E}v)$, for any $u, v \in V$.*

In the untyped setting which does not distinguish channels from variables, like in [10], every complete condition is satisfiable, and two substitutions satisfying the same complete condition relate to each other by some injective substitution. In this work, however, due to the distinction between variables and channels and the concept of closing substitution, there exist some conditions which are complete but not satisfiable. For instance, $\varphi = [x = a] \wedge [a = b] \wedge [b \neq c]$ is complete on $V = \{x, a, b, c\}$, with the equivalence classes $\{\{x, a, b\}, \{c\}\}$. This condition is not satisfiable because closing substitutions do not map channels to other channels, then $\sigma(a) = a \neq b = \sigma(b)$ for any closing substitution σ, i.e., $[\![\varphi\sigma]\!] = False$. In a typed setting, there are even fewer conditions which are satisfiable. For a given type environment $\Delta = \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$ we are only interested in closing substitutions of the form (called *legal substitution on* Δ): $\sigma = \{\widetilde{b}/\widetilde{x}\}$ where $\Delta_{\mathsf{c}} \vdash \widetilde{b} : \widetilde{T}$. As to the simple condition $[x_i = a]$, with $x_i, a \in dom(\Delta)$, if $\Delta(a) \not<: T_i$, the substitution $\{a/x_i\}$ is illegal and not considered. So no legal substitution can satisfy $[x_i = a]$, i.e., the condition is not satisfiable.

Lemma 18 *If φ is complete on $dom(\Delta)$ and $\emptyset \subset dom(\Delta_{\mathsf{v}}) \subset dom(\Delta)$, there is at most one legal substitution which satisfies φ.*

Proof. Since φ is complete, there is a corresponding equivalence relation $\mathcal{E}$. For φ to be satisfiable by a closing substitution σ on $dom(\Delta)$, each equivalence class of $\mathcal{E}$, say $\{u_1, \cdots, u_n\}$, must meet the following two conditions.

- Not all u_i are variables. Otherwise, for any $a \in dom(\Delta_{\mathsf{c}})$, $\varphi \Rightarrow [u_i \neq a]$. Then $\varphi\sigma \Rightarrow [\sigma(u_i) \neq a]$ for all $a \in dom(\Delta_{\mathsf{c}})$, contradicting the definition of closing substitution, which maps variables to channels, i.e., $\sigma(u_i) \in dom(\Delta_{\mathsf{c}})$.
- There is no more than one channel in any equivalence class. Otherwise, let a, b be two channels and $\varphi \Rightarrow [a = b]$, then $\varphi\sigma \Rightarrow [a = b]$, i.e., $[\![\varphi\sigma]\!] = \mathit{False}$.

As a result, in each equivalence class there is one and only one channel, possibly with some variables. So the class looks like $\{a, x_1, \cdots, x_{n-1}\}$ where $n \geq 1$. The substitution which satisfies φ must map all the variables in the equivalence class into its unique channel. Moreover, to ensure that φ is satisfied by a legal substitution, there is a third constraint imposed on the equivalence class:

- $\Delta(a) <: \Delta(x_i)$ for all $i \leq n - 1$.

All these conditions determine the uniqueness of the legal substitution, if it exists. $\square$

Lemma 19 *If φ and ψ are complete conditions on $dom(\Delta)$ and are satisfied by the same legal substitution on Δ, then $\varphi \Longleftrightarrow \psi$.*

Proof. $\varphi \wedge \psi$ is also satisfiable by the same legal substitution. Then $\varphi \Longleftrightarrow \varphi \wedge \psi \Longleftrightarrow \psi$ because φ and ψ are complete conditions. $\square$

The following lemma shows that in the presence of complete conditions, it is sufficient to test one substitution for typed bisimilarity of open terms.

Lemma 20 *Let $P \equiv \varphi P'$ and $Q \equiv \varphi Q'$, with φ complete on $dom(\Delta)$. If σ is a legal substitution on Δ, σ satisfies φ and $P\sigma \sim_{\Delta_{\mathsf{c}}} Q\sigma$, then $P \sim_{\Delta} Q$.*

Proof. By Lemma 18, besides σ there is no other substitution $\rho = \{\widetilde{c}/\widetilde{x}\}$ with $\Delta_{\mathsf{c}} \vdash \widetilde{c} : \widetilde{T}$ which can satisfy φ. In other words, $(\varphi P')\rho \sim_{\Delta_{\mathsf{c}}} \mathbf{0} \sim_{\Delta_{\mathsf{c}}} (\varphi Q')\rho$. Therefore we have $P \sim_{\Delta} Q$ by the definitions of typed bisimilarity. $\square$

As in [10], the definition of head normal form exploits complete conditions. Here the difference is that we only consider those conditions which can be satisfied by some legal substitutions, while in [10] all complete conditions are involved because all of them are satisfiable.

Definition 21 (head normal form) *We say that P is in head normal form w.r.t. Δ if P is of the form*

$$\sum_i \varphi_i \alpha_i.\varphi'_i P_i$$

where for all i,

(1) $bn(\alpha_i) \notin dom(\Delta)$;
(2) φ_i is complete on $dom(\Delta)$ and satisfiable by some legal substitution on Δ;
(3) $\varphi'_i = \varphi_i$ if α_i is an input or free action;
(4) $\varphi'_i = \varphi_i \wedge (\bigwedge_{v \in dom(\Delta)}[a \neq v])$ if $\alpha_i = \bar{u}(a:T)$.

The proof of completeness is established by induction on the depth, $d(P)$, of a head norm form (hnf) P. Its depth is defined as:

$$\begin{aligned} d(\mathbf{0}) &= 0 \\ d(\textstyle\sum_{i=1}^{n} \varphi_i \alpha_i.\varphi'_i P_i) &= 1 + max\{d(P_i) \mid 1 \leq i \leq n\} \end{aligned}$$

Lemma 22 *For each process P and environment Δ, with $fv(P) \subseteq dom(\Delta_{\mathsf{v}})$, there is some H of no greater depth than P and in hnf w.r.t. Δ, such that $\mathcal{A} \vdash P =_\Delta H$.*

Proof. By structural induction on processes. Let $V = dom(\Delta)$. We consider two interesting cases.

The first is when $P \equiv \alpha.P'$. Let x be any variable in V. If for each channel $a \in V$, $\Delta(a) \not<: \Delta(x)$, then we use **Tv1** to derive that $\mathcal{A} \vdash P =_\Delta \mathbf{0}$. Otherwise, suppose $V_x = \{a_1, \cdots, a_n\}$ collects all channels in V such that $\Delta(a_i) <: \Delta(x)$. As in the untyped setting [10] we can infer that $\mathcal{A} \vdash P =_\Delta \sum_{i=1}^{m} \psi_i \alpha.\psi_i P'$, where each ψ_i is complete on V, but not necessarily satisfiable by some legal substitution on Δ. There are two occasions where ψ_i is not satisfiable.

(1) If $\psi_i \Rightarrow [a = b]$ for $a, b \in dom(\Delta_{\mathsf{c}})$ and $a \neq b$, we use **Cnn1** to get $\mathcal{A} \vdash \psi_i \alpha.\psi_i P' =_\Delta \mathbf{0}$.
(2) If $\psi_i \Rightarrow [x \neq a_1] \cdots [x \neq a_n]$ we can use **Tvar*** to derive that $\mathcal{A} \vdash \psi_i \alpha.\psi_i P' =_\Delta \mathbf{0}$.

So we can remove the summand $\psi_i \alpha.\psi_i P'$ if ψ_i is not satisfiable. All other summands are satisfiable by some legal substitutions because $\psi_i \Rightarrow [x = a_i]$ for one $a_i \in V_x$ and $\psi_i \Rightarrow [x \neq b]$ for any other $b \in dom(\Delta_{\mathsf{c}})$.

The second case is when $P \equiv \psi\ Q\ R$. By induction hypothesis Q and R can be transformed into hnf w.r.t. Δ: $\mathcal{A} \vdash Q =_\Delta \sum_{i=1}^{n} \psi_i \alpha_i.\psi'_i Q_i$ and $\mathcal{A} \vdash R =_\Delta \sum_{j=1}^{m} \psi_j \alpha_j.\psi'_j R_j$. Let us examine the general case that $n, m > 0$. By **C9** and

C11, it is easy to see that

$$\mathcal{A} \vdash P =_\Delta \sum_{i=1}^{n}[\psi \wedge \psi_i]\alpha_i.\psi_i' Q_i + \sum_{j=1}^{m}[\neg\psi \wedge \psi_j]\alpha_j.\psi_j' R_j.$$

Clearly ψ can be reduced to a disjunctive normal form $\bigvee_{k=1}^{o}\bigwedge_{l=1}^{p}\varphi_{kl}$ where $o, p \geq 1$ and φ_{kl} is a match $[u_{kl} = v_{kl}]$ or mismatch $[u_{kl} \neq v_{kl}]$. Let $Q_i' = \alpha_i.\psi_i' Q_i$. We transform each summand $[\psi \wedge \psi_i]Q_i'$ as follows.

$$\begin{aligned} \mathcal{A} \vdash [\psi \wedge \psi_i]Q_i' &=_\Delta [(\textstyle\bigvee_{k=1}^{o}\bigwedge_{l=1}^{p}\varphi_{kl}) \wedge \psi_i]Q_i' && \text{by } \mathbf{C1} \\ &=_\Delta [\textstyle\bigvee_{k=1}^{o}(\psi_i \wedge \bigwedge_{l=1}^{p}\varphi_{kl})]Q_i' && \text{by } \mathbf{C1} \\ &=_\Delta \textstyle\sum_{k=1}^{o}[\psi_i \wedge \bigwedge_{l=1}^{p}\varphi_{kl}]Q_i' && \text{by } \mathbf{C10} \end{aligned}$$

Now we assert that each summand $[\psi_i \wedge \bigwedge_{l=1}^{p}\varphi_{kl}]Q_i'$ is provably equal to $\mathbf{0}$ or $\psi_i Q_i'$.

Let $\phi_k = \bigwedge_{l=2}^{p}\varphi_{kl}$ if $p > 1$, and $\phi_k = \mathit{True}$ if $p = 1$. So by **C1** we have $\mathcal{A} \vdash [\psi_i \wedge \bigwedge_{l=1}^{p}\varphi_{kl}]Q_i' =_\Delta [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i'$. Here φ_{k1} may be a match or mismatch. We look at match first. Let $\varphi_{k1} = [u_{k1} = v_{k1}]$ for some u_{k1}, v_{k1} s.t. $u_{k1} \neq v_{k1}$.

(1) If $u_{k1}, v_{k1} \in V$, then $[\varphi_{k1} \wedge \phi_k \wedge \psi_i]$ is semantically equivalent either to *False* or to $[\phi_k \wedge \psi_i]$ because ψ_i is complete on V. That is, we can infer $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathbf{0}$ or $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta [\phi_k \wedge \psi_i]Q_i'$.

(2) If $u_{k1}, v_{k1} \notin V$, then u_{k1}, v_{k1} are channels because $fv(P) \subseteq \widetilde{x}$. By **Cnn1** we get $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathbf{0}$.

(3) If $u_{k1} \in V$ and $v_{k1} \notin V$, then v_{k1} is a channel but u_{k1} can be either a channel or a variable.
    (a) u_{k1} is also a channel. We infer $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathbf{0}$ by **Cnn1**.
    (b) u_{k1} is a variable, i.e., $u_{k1} \in \widetilde{x}$. We infer $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathbf{0}$ by **Tvn1**.

When φ_{k1} is a mismatch $[u_{k1} \neq v_{k1}]$ we apply similar arguments. In Case 1 the result is the same. In the last two cases, using **Cnn2** or **Tvn2** we infer that $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta [\phi_k \wedge \psi_i]Q_i'$. Since there are only p components in $\bigwedge_{l=1}^{p}\varphi_{kl}$, we can repeat this inference for at most p times and eventually get either $\mathcal{A} \vdash [\psi_i \wedge \bigwedge_{l=1}^{p}\varphi_{kl}]Q_i' =_\Delta \mathbf{0}$ or $\mathcal{A} \vdash [\psi_i \wedge \bigwedge_{l=1}^{p}\varphi_{kl}]Q_i' =_\Delta \psi_i Q_i'$.

Similar result can be got for $[\neg\psi \wedge \psi_j]\alpha_j.\psi_j' R_j$ as well.

In summary we have shown that each summand of P can either be removed or put into the form of the summands of a hnf. □

Theorem 23 (Completeness of $\mathcal{A}$) *If $P \sim_\Delta Q$ then $\mathcal{A} \vdash P =_\Delta Q$.*

Proof. Let $\Delta = \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$. If there is no legal substitution on Δ, i.e., no $\widetilde{a}$ with $\Delta_{\mathsf{c}} \vdash \widetilde{a} : \widetilde{T}$, then by **Tv1** we have that $\mathcal{A} \vdash P =_{\Delta} \mathbf{0} =_{\Delta} Q$.

Below we suppose that there exist legal substitutions on Δ. By Lemma 22 we assume that P and Q are in hnf w.r.t. Δ. Let

$$\mathcal{A} \vdash P =_{\Delta} \sum_i \varphi_i \alpha_i . P_i \qquad \text{and} \qquad \mathcal{A} \vdash Q =_{\Delta} \sum_j \psi_j \beta_j . Q_j .$$

For any summand $\varphi_i \alpha_i . P_i$ of P, let σ_i be a legal substitution on Δ which satisfies φ_i (actually σ_i is the only legal substitution satisfying φ_i, according to Lemma 18). So if $\varphi_i \Rightarrow [x = a]$ then $\Delta(a) <: \Delta(x)$ and $x\sigma_i = a$. By using **Tpre*** we can transform the action α_i into $\alpha_i\sigma_i$ which contains no free variable. For example, if $\alpha_i = \bar{x}y$ and $\varphi_i \Rightarrow [x = a] \wedge [y = b]$, then $\varphi_i \bar{x}y.P_i =_{\Delta} \varphi_i \overline{x\sigma_i} y\sigma_i . P_i \equiv \varphi \bar{a}b.P_i$. Furthermore, if the action $\alpha_i\sigma_i$ is disallowed by the environment (e.g., $\alpha_i\sigma_i = \bar{a}b$ and $\Delta(a)\not\downarrow_{\mathtt{i}}$, similar for input actions), then by **Tin*** and **Tout*** the summand $\varphi_i \alpha_i . P_i$ is provably equal to $\mathbf{0}$ and thus can be consumed by **S1**. After finite steps of transformation, all remaining summands are active, i.e., can perform some actions allowed by Δ. We do similar transformation for Q.

Now we prove by induction on the depth of $P + Q$ that each active summand of P is provably equal to some active summand of Q. An active summand $\varphi_i \alpha_i . P_i$ of P gives rise to a transition $\Delta_{\mathsf{c}} \sharp P\sigma_i \xrightarrow{\alpha_i\sigma_i} \Delta'_{\mathsf{c}} \sharp P_i\sigma_i$. Since $P \sim_{\Delta} Q$, we have $P\sigma_i \sim_{\Delta_{\mathsf{c}}} Q\sigma_i$. So there is a matching transition $\Delta_{\mathsf{c}} \sharp Q\sigma_i \xrightarrow{\beta_j\sigma_i} \Delta''_{\mathsf{c}} \sharp Q_j\sigma_i$ contributed by some active summand $\psi_j \beta_j . Q_j$ of Q, with ψ_j satisfied by σ_i. By Lemma 19 we know that $\varphi_i \Longleftrightarrow \psi_j$. From the definition of $\sim_{\Delta_{\mathsf{c}}}$ we have:

(1) if $\alpha_i\sigma_i = \beta_j\sigma_i = \tau$ then $P_i\sigma_i \sim_{\Delta_{\mathsf{c}}} Q_j\sigma_i$;
(2) if $\alpha_i\sigma_i = \beta_j\sigma_i = \bar{a}b$, for some channels a, b, then $P_i\sigma_i \sim_{\Delta_{\mathsf{c}} \sqcap b:\Delta(a)_{\mathtt{i}}} Q_j\sigma_i$;
(3) if $\alpha_i\sigma_i = \bar{a}(b : T_1)$ and $\beta_j\sigma_i = \bar{a}(b : T_2)$ for some channels a, b then $P_i\sigma_i \sim_{\Delta_{\mathsf{c}}, b:\Delta(a)_{\mathtt{i}}} Q_j\sigma_i$;
(4) if $\alpha_i\sigma_i = a(x : T_1)$ and $\beta_j\sigma_i = a(x : T_2)$, for some a and x, then for all c with $\Delta_{\mathsf{c}} \vdash c : \Delta(a)_{\mathsf{o}}$ it holds that $P_i\sigma_i\{c/x\} \sim_{\Delta_{\mathsf{c}}} Q_j\sigma_i\{c/x\}$.

Let us analyze the last two cases in details. In Case 3, σ_i is also a legal substitution on $\Delta, b : \Delta(a)_{\mathtt{i}}$. By Lemma 20 one can infer that $P_i \sim_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. By induction hypothesis $\mathcal{A} \vdash P_i =_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. By **Iout*, Ires*, Icon** and **C1** it can be inferred that $\mathcal{A} \vdash \varphi_i \bar{a}(b : T_1).P_i =_{\Delta} \psi_j \bar{a}(b : T_2)Q_j$. The required result is got by using **Tpre***.

In Case 4, we have that $P_i\sigma_i\{c/x\} \sim_{\Delta_{\mathsf{c}}} Q_j\sigma_i\{c/x\}$ for all c satisfying the condition $\Delta_{\mathsf{c}} \vdash c : \Delta(a)_{\mathsf{o}}$. Note that $P_i = \varphi_i P'_i$ and $Q_j = \psi_j Q'_j$. By Lemma 18, any substitution $\rho = \{\widetilde{c}/\widetilde{x}, d/x\}$, with $\Delta_{\mathsf{c}} \vdash \widetilde{c} : \widetilde{T}, d : \Delta(a)_{\mathsf{o}}$, which can satisfy φ_i and ψ_j, must coincide with σ on variables $\widetilde{x}$. That is, $\rho = \sigma\{d/x\}$. Therefore $P_i\rho \sim_{\Delta_{\mathsf{c}}} Q_j\rho$. For any other substitution, say ρ', $[\![\varphi_i\rho']\!] = [\![\psi_j\rho']\!] = \mathit{False}$, and so $P_i\rho' \sim_{\Delta_c} \mathbf{0} \sim_{\Delta_c} Q_j\rho'$. Consequently for all ρ we have $P_i\rho \sim_{\Delta_c} Q_j\rho$, i.e., $P_i \sim_{\Delta,x:\Delta(a)_\circ} Q_j$. Now applying induction hypothesis, $\mathcal{A} \vdash P_i =_{\Delta,x:\Delta(a)_\circ} Q_j$. It follows that $\mathcal{A} \vdash a(x : T_1).P_i =_\Delta a(x : T_2).Q_j$ by **Iin***. Then we can infer $\mathcal{A} \vdash \varphi_i\alpha_i.P_i =_\Delta \psi_j\beta_j.Q_j$ by using **Icon, C1** and **Tpre***, in the listed order. □

5 Other equivalences

In this section we study a variant bisimilarity proposed in [3], which allows extension of environment and enjoys a nice contextual property. Proof systems for closed terms are given. An indirect axiomatisation is got by resorting to the system $\mathcal{A}$ of Section 4. We also show that the difference between late and early style of typed bisimilarity is characterized by one axiom.

5.1 Hennessy and Rathke's typed bisimilarity

5.1.1 Proof system for closed terms

In the input clause of $\sim$ (Definition 9), the type environment Δ is not extended. By contrast, extensions are allowed in the bisimilarity used in [3]. We denote with $\asymp_\Delta$ the variant of $\sim_\Delta$ which allows extension; its definition is obtained from that of $\sim_\Delta$ by using the following input clause:

- if $\Delta\sharp P \xrightarrow{a(x:T)} \Delta'\sharp P'$, then for some Q', $\Delta\sharp Q \xrightarrow{a(x:S)} \Delta''\sharp Q'$ and $\Delta, \Delta''' \vdash b : \Delta(a)_\circ$ implies $P'\{b/x\}\ \mathcal{R}_{\Delta,\Delta'''}\ Q'\{b/x\}$, for any channel b and closed type environment Δ''' with $dom(\Delta''') \cap (fn(P, Q) \cup dom(\Delta)) = \emptyset$.

Similarly, Δ can be extended in the definition on open terms.

Lemma 24 *If $P \asymp_\Delta Q$ then $P \sim_\Delta Q$.*

In $\asymp_\Delta$, the environment collects the knowledge of the observer *relative* to the tested processes, in the sense that the environment only tells us what the observer knows of the free channels of the processes. In contrast, in $\sim_\Delta$, the environment collects the *absolute* knowledge of the observer, including information on channels that at present do not appear in the tested processes, but that might appear later — if the observer decides to send them to the processes. The main advantage of $\asymp_\Delta$ is that the environment is allowed to invent an unbounded number of distinct names, so it is more suitable for infinite systems. On the other hand, $\sim_\Delta$ allows us to express more refined interrogations on the equivalence of processes, for it gives us more flexibility in setting the observer knowledge. Indeed, while $\asymp$-equivalences can be expressed using $\sim$ (Lemma 24), the converse is false. For instance, the processes

$$P \overset{\text{def}}{=} a(x : \mathtt{bo}T).[x = y]\tau \qquad Q \overset{\text{def}}{=} a(x : \mathtt{bo}T).\mathbf{0}$$

are in the relation $\sim_\Delta$, for $\Delta \overset{\text{def}}{=} a : \mathtt{obo}T, b : \mathtt{bb}T, y : \mathtt{ob}T$. However, they are not in a relation $\asymp_\Gamma$, for any Γ: the observer can always create a new channel of type $\mathtt{bo}T$, and use it to instantiate both x and y, thus validating the condition $[x = y]$.

In the following lemma we give two properties of $\asymp_\Delta$. They are analogous to Lemma 11 and 13 respectively, and can be proved as their counterparts.

Lemma 25 *(1) If $P \asymp_\Delta Q$ and $\Delta <: \Delta'$, then $P \asymp_{\Delta'} Q$.*
(2) If $P \asymp_\Delta Q$ then $P\sigma \asymp_{\Delta\sigma} Q\sigma$ for σ injective on $fn(P, Q) \cup dom(\Delta)$ and $\Delta\sigma$ is the type environment which maps $\sigma(a)$ to $\Delta(a)$ for all $a \in dom(\Delta)$.

An important property which is enjoyed by $\asymp_\Delta$ but not by $\sim_\Delta$ is as follows.

Lemma 26 *If $P \asymp_\Delta Q$ and $a \notin fn(P, Q) \cup dom(\Delta)$, then $P \asymp_{\Delta,a:T} Q$.*

This lemma says that increasing capabilities on irrelevant channels does not raise an observer's discriminating power. The reason is that the observer already has the ability to create new channels, since in the definitions of bisimulations we test all channels with appropriate types for the case of input.

Lemma 27 *It holds that $a(x : T_1).P \asymp_\Delta a(x : T_2).Q$, if the following two conditions are satisfied.*
(i) $P\{b/x\} \asymp_\Delta Q\{b/x\}$ for all b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$;
(ii) given $c \notin fn(P, Q) \cup dom(\Delta)$, $P\{c/x\} \asymp_{\Delta,c:T} Q\{c/x\}$ for all $T <: \Delta(a)_{\mathsf{o}}$.

Proof. The action of the configuration $\Delta \sharp a(x : T_1).P$ can be matched by that of $\Delta \sharp a(x : T_2).Q$. So we only show that $P\{b/x\} \asymp_{\Delta,\Delta'} Q\{b/x\}$ for any b and Δ' with $dom(\Delta') \cap fn(P, Q) = \emptyset$ and $\Delta, \Delta' \vdash b : \Delta(a)_{\mathsf{o}}$. There are two possibilities:

(1) $b \in dom(\Delta)$. When $\Delta' = \emptyset$, the result follows from the hypothesis (i). For other Δ', we get the result indirectly by using Lemma 26.

(2) $b \notin dom(\Delta)$. We consider the case that $\Delta' = b : T$ with $T <: \Delta(a)_{\mathsf{o}}$. Base on this case, the result for other Δ' with $\Delta' = b : T, \Delta''$ can be inferred from Lemma 26. From (ii) we know that $P\{c/x\} \asymp_{\Delta,c:T} Q\{c/x\}$. Since bisimulation is insensitive to injective type-preserving substitutions by Lemma 25 (2), we have $P\{c/x\}\{b/c\} \asymp_{\Delta,b:T} Q\{c/x\}\{b/c\}$. That is, $P\{b/x\} \asymp_{\Delta,\Delta'} Q\{b/x\}$, which is the required result. $\square$

We can derive a proof system for $\asymp$ with a simple modification of that for $\sim$ in Section 3. Let $\mathcal{P}'$ be the system obtained from $\mathcal{P}$ by replacing rule **Iinc*** with **Iinc′**:

Iinc′ If
- $P\{b/x\} =_\Delta Q\{b/x\}$ for all b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$, and
- given $c \notin fn(P, Q) \cup dom(\Delta)$,
  $P\{c/x\} =_{\Delta,c:T} Q\{c/x\}$ for all $T <: \Delta(a)_{\mathsf{o}}$,

then $a(x : T_1).P =_\Delta a(x : T_2).Q$.

The quantification on T in the premises is finite: any type has only finitely many subtypes.

Theorem 28 *$\mathcal{P}' \vdash P =_\Delta Q$ iff $P \asymp_\Delta Q$, where P and Q are closed.*

Proof. According to Lemma 27, rule **Iinc′** is sound. The soundness of other rules is easy to show. The completeness proof is similar to that of $\mathcal{P}$ (Theorem 15). $\square$

5.1.2 Indirect axiomatisation

The previous definition of $\asymp$ involves infinitely many substitutions. Nevertheless we show in the following lemma that there exists an efficient characterisation of the equivalence which employs only finitely many substitutions. This characterisation result relies on the assumption that the set of subtypes of any type is finite and the environment contains finitely many variables (the terms could even be extended with non-finite operators such as recursion, as long as they contain finitely many free variables). First, we introduce a notation. Let $\widetilde{T} = T_1, \cdots, T_n$. There are only finitely many different types, say $S_1, \ldots, S_m$, each of which is a subtype of some T_i for $i \leq n$. Then we pick n fresh names (which do not appear in Δ, P and Q) $a_{i1}, \cdots, a_{in}$ for each type S_i and extend Δ in the following way.

$$Env(\Delta, \widetilde{T}, P, Q) \stackrel{\text{def}}{=} \Delta \cup \{a_{ik} : S_i \mid 0 < i \leq m, 0 < k \leq n, a_{ik} \notin fn(\Delta, P, Q)\}$$

Lemma 29 *Suppose $\Delta \stackrel{\text{def}}{=} \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$. If for each legal substitution σ on $Env(\Delta, \widetilde{T}, P, Q)$ it holds that $P\sigma \asymp_{Env(\Delta_{\mathsf{c}}, \widetilde{T}, P, Q)} Q\sigma$, then $P \asymp_\Delta Q$.*

Proof. Let $\Delta_1 = Env(\Delta_{\mathsf{c}}, \widetilde{T}, P, Q)$, and the length of the tuple $\widetilde{T}$ be n with $n > 0$. We prove a stronger result $P \asymp_{\Delta_1, \widetilde{x}:\widetilde{T}} Q$ and then conclude by Lemma 25 (1). We shall show that $P\{\widetilde{b}/\widetilde{x}\} \asymp_{\Delta_1, \Delta'} Q\{\widetilde{b}/\widetilde{x}\}$ for any $\widetilde{b}$ and closed environment Δ' s.t. $dom(\Delta') \cap fn(P, Q) = \emptyset$ and $\Delta_1, \Delta' \vdash \widetilde{b} : \widetilde{T}$. We proceed by induction on the number of names appearing in $\widetilde{b}$ but not in $dom(\Delta_1)$, which is defined as follows.

$$
\begin{aligned}
num(\emptyset) &= 0 \\
num(\widetilde{b}) &= \begin{cases} num(b_1 \cdots b_{n-1}) + 1 & \text{if } b_n \notin dom(\Delta_1) \\ num(b_1 \cdots b_{n-1}) & \text{otherwise} \end{cases}
\end{aligned}
$$

- Base step. Suppose $num(\widetilde{b}) = 0$. When $\Delta' = \emptyset$, the result follows from the hypothesis. For other Δ', the result is got indirectly by using Lemma 26.
- Inductive step. Suppose that the result holds for all $\widetilde{b}$ which satisfy the conditions in the hypothesis and $num(\widetilde{b}) \leq k$. Given another $\widetilde{b}$ with $num(\widetilde{b}) = k+1$. Without loss of generality we assume that there exists a $c \notin dom(\Delta_1)$ and $l \leq n$ such that $b_1 = b_2 = \cdots = b_l = c$ and $b_i \neq c$ for all $i > l$. Then Δ_1, Δ' can be rewritten as $\Delta_2, c : S_i$ for some Δ_2 and S_i s.t. $S_i \leq T_\iota$ for all $\iota \leq l$. Choose one name from $\{a_{i1}, \cdots, a_{in}\}$, say a_{ij}, which is different from any names in $b_{l+1}, \cdots, b_n$, and construct a substitution

$$\sigma = \{a_{ij}/x_1, \cdots, a_{ij}/x_l, b_{l+1}/x_{l+1}, \cdots, b_n/x_n\}$$

Obviously $\Delta_2 \vdash a_{ij} : T_1, \cdots, a_{ij} : T_l, b_{l+1} : T_{l+1}, \cdots, b_n : T_n$ and $num(a_{ij}, \cdots, a_{ij}, b_{l+1}, \cdots, b_n) \leq k$. By induction hypothesis $P\sigma \asymp_{\Delta_2} Q\sigma$. From Lemma 25 (2) we have

$$P\sigma\{c/a_{ij}\} \ \asymp_{\Delta_2\{c/a_{ij}\}} \ Q\sigma\{c/a_{ij}\}$$

i.e., $P\{\widetilde{b}/\widetilde{x}\} \ \asymp_{\Delta_2\{c/a_{ij}\}} \ Q\{\widetilde{b}/\widetilde{x}\}$. As $a_{ij} \notin dom(\Delta_2\{c/a_{ij}\})$, by Lemma 26 we get $P\{\widetilde{b}/\widetilde{x}\} \asymp_{\Delta_3} \ Q\{\widetilde{b}/\widetilde{x}\}$ for $\Delta_3 = \Delta_2\{c/a_{ij}\}, a_{ij} : S_i = \Delta_1, \Delta'$, which is just the required result. $\square$

Below we establish a property of $\sim_\Delta$, corresponding to Lemma 26 for $\asymp_\Delta$. It allows the extension of Δ in a limited way. The proof employs the concept of *size* of a process P, written $size(P)$, which we define as follows.

$$
\begin{aligned}
size(\mathbf{0}) &= 0 & size(P+Q) &= max\{size(P), size(Q)\} \\
size(\alpha.P) &= 1 + size(P) & size(\varphi PQ) &= max\{size(P), size(Q)\} \\
size((\nu a : S)P) &= size(P) & size(P \mid Q) &= size(P) + size(Q)
\end{aligned}
$$

One can verify that if $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ then $size(P) > size(P')$ and $fn(P') \subseteq fn(P) \cup bn(\alpha)$.

Lemma 30 *Given two closed terms P and Q, let $\Delta = \Delta_0, c_1 : T, ..., c_n : T$ with $n \geq size(P+Q)$ and $c_i \notin fn(P,Q)$ for all $i \in 1..n$. If $P \sim_\Delta Q$ then $P \sim_{\Delta, a:T} Q$ for $a \notin fn(P,Q) \cup dom(\Delta)$.*

Proof. By induction on the size of $P+Q$. If $size(P+Q)=0$ then it is obvious that $P \sim_{\Delta,a:T} \mathbf{0} \sim_{\Delta,a:T} Q$. Below we suppose $size(P+Q)>0$. If $\Delta, a:T \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ there must exist some Δ'' s.t. $\Delta' = \Delta'', a:T$ because a does not affect the transition. In other words, we have $\Delta \sharp P \xrightarrow{\alpha} \Delta'' \sharp P'$. Since $P \sim_\Delta Q$, we have a matching transition $\Delta \sharp Q \xrightarrow{\beta} \Delta''' \sharp Q'$, where $|\alpha|=|\beta|$. It follows that $\Delta, a:T \sharp Q \xrightarrow{\beta} \Delta''', a:T \sharp Q'$. There are two cases:

1. (1) α is not an input action. In this case $\Delta'' = \Delta'''$ and $P' \sim_{\Delta''} Q'$. By induction hypothesis we have $P' \sim_{\Delta'',a:T} Q'$.
2. (2) α is an input action $b(x:S)$. Then for each d with $\Delta \vdash d : \Delta(b)_\circ$ it holds that $P'\{d/x\} \sim_\Delta Q'\{d/x\}$.
   - (a) If $d \in dom(\Delta_0)$ with $\Delta_0 \vdash d : \Delta(b)_\circ$, then $n \geq size(P+Q) > size(P'\{d/x\}+Q'\{d/x\})$ and $c_i \notin fn(P'\{d/x\}, Q'\{d/x\})$ for $i \in 1..n$. By induction hypothesis we have $P'\{d/x\} \sim_{\Delta,a:T} Q'\{d/x\}$.
   - (b) If $c_1:T, ..., c_n:T \vdash d : \Delta(b)_\circ$, then without loss of generality we may assume that $d = c_1$. It can be checked that $n-1 \geq size(P+Q)-1 \geq size(P'\{d/x\}+Q'\{d/x\})$ and $c_i \notin fn(P'\{d/x\}, Q'\{d/x\})$ for $i \in 2..n$. We can now appeal to induction hypothesis and get the result that $P'\{d/x\} \sim_{\Delta,a:T} Q'\{d/x\}$.
   - (c) If $a:T \vdash a : \Delta(b)_\circ$, then $T <: \Delta(b)_\circ$ and thus $\Delta \vdash c_1 : \Delta(b)_\circ$, which implies $P'\{c_1/x\} \sim_\Delta Q'\{c_1/x\}$. As $\{a/c_1\}$ is an injective type-preserving substitution, we have

     $$P'\{c_1/x\}\{a/c_1\} \sim_{\Delta\{a/c_1\}} Q'\{c_1/x\}\{a/c_1\}$$

     i.e., $P'\{a/x\} \sim_{\Delta\{a/c_1\}} Q'\{a/x\}$. Now observe that
     - (i) $n-1 \geq size(P+Q)-1 \geq size(P'\{a/x\}+Q'\{a/x\})$,
     - (ii) $c_i \notin fn(P'\{a/x\}, Q'\{a/x\})$ for $i \in 2..n$,
     - (iii) $c_1 \notin fn(P'\{a/x\}, Q'\{a/x\}) \cup dom(\Delta\{a/c_1\})$.

     By induction hypothesis we have $P'\{a/x\} \sim_{\Delta\{a/c_1\},c_1:T} Q'\{a/x\}$. Note that $\Delta\{a/c_1\}, c_1:T = \Delta, a:T$.

   In summary, for each d with $\Delta, a:T \vdash d : \Delta(b)_o$, it always holds that $P'\{a/x\} \sim_{\Delta,a:T} Q'\{a/x\}$, which is the required result. □

We know from Lemma 24 that $\sim_\Delta$ is weaker than $\asymp_\Delta$. This gives rise to an interesting question: whether there exists some Δ^* such that under the extended environment Δ, Δ^* we have that $P \sim_{\Delta,\Delta^*} Q$ iff $P \asymp_\Delta Q$. We shall give a positive answer to this question, though we did not succeed in obtaining the counterpart of Theorem 23 for $\asymp$. The encountered problem is discussed at the end of this subsection.

We define the depth, $d(T)$, of a type T, indicating the maximum number of nesting of capabilities in it.

$$d(\texttt{unit}) = 0 \qquad d(\texttt{i}T) = d(\texttt{o}T) = 1 + d(T)$$
$$d(\texttt{b}\langle T, S\rangle) = 1 + max\{d(T), d(S)\}$$

Let $\Gamma \vdash P$. Each name in P has a type, either recorded in the syntax of P or in Γ. If $T_1, \ldots, T_n$ are all such types, $d(\Gamma, P)$ is $max\{d(T_i) \mid 1 \leq i \leq n\}$. Now, if $\Delta\sharp P_i$ is a configuration, for $i = 1, 2$, then there are type environments Γ_i such that $\Gamma_i <: \Delta$ and $\Gamma_i \vdash P_i$. In this case, we set $d(P_1, P_2, \Gamma_1, \Gamma_2)$ as $max\{d(\Gamma_1, P_1), d(\Gamma_2, P_2)\}$. There are only finitely many different types with depth less than or equal to $d(P_1, P_2, \Gamma_1, \Gamma_2)$, say $S_1, \ldots, S_m$, and $\Delta_{\texttt{v}}$ is defined on finitely many variables, say $x_1, \ldots, x_k$. We can pick up n fresh (hitherto unused) channels $a_{i1}, \ldots, a_{in}$ for each S_i, where $n = max\{k, size(P + Q)\}$, and construct a type environment

$$Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2) = \{a_{ij} : S_{ij} \mid 0 < i \leq m, 0 < j \leq n\}.$$

We say that $P_1 \asymp_\Delta P_2$ *under* Γ_1, Γ_2 if $\Gamma_i <: \Delta$ and $\Gamma_i \vdash P_i$ $(i = 1, 2)$.

Lemma 31 *If* $P_1 \asymp_\Delta P_2$ *under* Γ_1, Γ_2 *then* $P_1 \ \sim_{\Delta, Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} \ P_2$.

Proof. By Lemma 26 we have $P_1 \ \asymp_{\Delta, Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} \ P_2$. Then the result follows from Lemma 24. $\Box$

In this lemma, P_1, P_2 can be either closed or open. For the opposite direction, we consider closed terms first.

Lemma 32 *If* $\Delta\sharp P_i$ *respects* Γ_i, P_i *is closed, for* $i = 1, 2$, *and it holds that* $P_1 \ \sim_{\Delta, Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} \ P_2$, *then* $P_1 \asymp_\Delta P_2$.

Proof. By induction on the size of $P_1 + P_2$. In the case $size(P_1 + P_2) = 0$, it is immediate that $P_1 \asymp_\Delta \mathbf{0} \asymp_\Delta P_2$. Below we suppose $size(P_1 + P_2) > 0$. Let $\Delta^* = Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$. Since $dom(\Delta^*) \cap fn(P_1, P_2) = \emptyset$, all actions of the configuration $\Delta, \Delta^*\sharp P_1$ can be performed by $\Delta\sharp P_1$, and vice versa. Suppose that $\Delta\sharp P_1 \xrightarrow{\alpha} \Delta'\sharp P_1'$. It is easy to see that there is a matching transition $\Delta\sharp P_2 \xrightarrow{\beta} \Delta''\sharp P_2'$.

(1) If α is not an input action, then $\mid \alpha \mid = \mid \beta \mid$, $\Delta' = \Delta''$ and $P_1' \sim_{\Delta', \Delta^*} P_2'$. Suppose that $\Delta'\sharp P_i'$ respects Γ_i' for $i = 1, 2$. Clearly $d(P_1', P_2', \Gamma_1', \Gamma_2') \leq d(P_1, P_2, \Gamma_1, \Gamma_2)$ by Lemma 6. From Lemma 30 we have $P_1' \sim_{\Delta_1} P_2'$ where $\Delta_1 = \Delta', \Delta^*, Env(\Delta', P_1', P_2', \Gamma_1', \Gamma_2')$. Now it follows from Lemma 11 that $P_1' \ \sim_{\Delta', Env(\Delta', P_1', P_2', \Gamma_1', \Gamma_2')} P_2'$. By induction hypothesis we get $P_1' \asymp_{\Delta'} P_2'$.

(2) If α is an input action $a(x : T)$, then $P_1'\{b/x\} \ \sim_{\Delta, \Delta^*} \ P_2'\{b/x\}$ for all b with $\Delta, \Delta^* \vdash b : \Delta(a)_{\texttt{o}}$. Note that $\Delta, \Delta^* \supseteq \Delta_1$ for some $\Delta_1 = Env(\Delta, \Delta(a)_{\texttt{o}}, P_1', P_2')$ by the definition of $Env(\Delta, \widetilde{T}, P_1, P_2)$ given in the beginning of this subsection. So for all c with $\Delta_1 \vdash c : \Delta(a)_{\texttt{o}}$ we have $P_1'\{c/x\} \ \sim_{\Delta, \Delta^*} \ P_2'\{c/x\}$. It can be checked that $\Delta_1\sharp P_i'\{c/x\}$ is a config-

uration respecting $\Gamma_i' \stackrel{\text{def}}{=} \Gamma_i, \Delta^*$ for $i = 1, 2$. As

$$d(\Delta_1, P_1'\{c/x\}, P_2'\{c/x\}, \Gamma_1', \Gamma_2') \leq d(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$$

we have $P_1'\{c/x\} \sim_{\Delta_2} P_2'\{c/x\}$, where

$$\Delta_2 = \Delta, \Delta^*, \mathit{Env}(\Delta_1, P_1'\{c/x\}, P_2'\{c/x\}, \Gamma_1', \Gamma_2')$$

by Lemma 30. It follows from Lemma 11 that $P_1'\{c/x\} \sim_{\Delta_3} P_2'\{c/x\}$ where $\Delta_3 = \Delta_1$, $\mathit{Env}(\Delta_1, P_1'\{c/x\}, P_2'\{c/x\}, \Gamma_1', \Gamma_2')$. By induction hypothesis we get $P_1'\{c/x\} \asymp_{\Delta_1} P_2'\{c/x\}$. By Lemma 29 it follows that $P_1' \asymp_{\Delta, x:\Delta(a)_\circ} P_2'$, which is the required result. $\Box$

Lemma 33 *If $\Delta \sharp P_i$ respects Γ_i, for $i = 1, 2$, and $P_1 \sim_{\Delta, \mathit{Env}(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} P_2$ then $P_1 \asymp_\Delta P_2$.*

Proof. Similar to the second case of the proof in Lemma 32. Let $\Delta = \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$ and $\Delta^* = \mathit{Env}(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$. Then for any legal substitution σ on Δ, Δ^* we have that $P_1\sigma \sim_{\Delta_{\mathsf{c}}, \Delta^*} P_2\sigma$. We also have $\Delta_{\mathsf{c}}, \Delta^* \supseteq \Delta_1$ for some $\Delta_1 = \mathit{Env}(\Delta_{\mathsf{c}}, \widetilde{T}, P_1, P_2)$. So for all $\rho = \{\widetilde{c}/\widetilde{x}\}$ with $\Delta_1 \vdash \widetilde{c} : \widetilde{T}$ we have $P_1\rho \sim_{\Delta_{\mathsf{c}}, \Delta^*} P_2\rho$. One can prove that $\Delta_1 \sharp P_i\rho$ is a configuration respecting $\Gamma_i' \stackrel{\text{def}}{=} \Gamma_i, \Delta^*$. Obviously $d(\Delta_1, P_1\rho, P_2\rho, \Gamma_1', \Gamma_2') = d(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$, so $P_1\rho \sim_{\Delta_2} P_2\rho$ for some environment $\Delta_2 = \Delta_{\mathsf{c}}, \Delta^*$, $\mathit{Env}(\Delta_1, P_1\rho, P_2\rho, \Gamma_1', \Gamma_2')$. It follows that $P_1\rho \sim_{\Delta_1, \mathit{Env}(\Delta_1, P_1\rho, P_2\rho, \Gamma_1', \Gamma_2')} P_2\rho$. By Lemma 32 we have $P_1\rho \asymp_{\Delta_1} P_2\rho$, which implies $P_1 \asymp_\Delta P_2$ by Lemma 29. $\Box$

Combining Lemma 31 and 33 we have the result below.

Lemma 34 *$P_1 \asymp_\Delta P_2$ under Γ_1, Γ_2 iff $P_1 \sim_{\Delta, \mathit{Env}(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} P_2$.*

As a consequence of this lemma, we obtain the following theorem.

Theorem 35 *$P_1 \asymp_\Delta P_2$ under Γ_1, Γ_2 iff $\mathcal{A} \vdash P_1 =_{\Delta, \mathit{Env}(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} P_2$.*

Directly axiomatizing $\asymp$ appears far from straightforward due to complications entailed by subtyping. We consider an example. Let $T \stackrel{\text{def}}{=} \mathtt{unit}$ and

$$\begin{aligned}
\Delta &\stackrel{\text{def}}{=} a : \mathsf{obo}T, y : \mathsf{ob}T \\
R &\stackrel{\text{def}}{=} \tau.((\nu c : \mathsf{b}T)\bar{y}c.\bar{c} + a(x : \mathsf{bo}T).[x = y]\tau) \\
R_1 &\stackrel{\text{def}}{=} \tau.((\nu c : \mathsf{b}T)\bar{y}c.\mathbf{0} + a(x : \mathsf{bo}T).[x = y]\tau) \\
R_2 &\stackrel{\text{def}}{=} \tau.((\nu c : \mathsf{b}T)\bar{y}c.\bar{c} + a(x : \mathsf{bo}T).\mathbf{0}).
\end{aligned}$$

It holds that

$$R + R_1 + R_2 \asymp_\Delta R_1 + R_2.$$

Here y can be instantiated by channels with subtypes of $\mathtt{ob}T$, which can be seen in Figure 1 (b). When y is instantiated by a channel with type $\mathtt{bo}T$, we can simulate R with R_1. For other subtypes of $\mathtt{ob}T$, we can simulate R with R_2. That is, we have two equivalent processes, say P and Q, with a free variable y, and the actions from a summand of P have to be matched by different summands of Q, depending on the types used to instantiate y. It appears hard to capture this relationship among terms using axioms involving only the standard operators of the π-calculus.

5.2 Early bisimilarity

All bisimilarities considered so far in the paper are in the "late" style [13]. As usual, the "early" versions are obtained by commuting the quantifiers in the input clause of bisimilarity. As in the untyped case, the difference between late and early equivalences is captured by the axiom **SP** [10]:

$$\begin{aligned}\mathbf{SP}\qquad & a(x:T_1).P + a(x:T_2).Q\\ =_{\Delta}\ & a(x:T_1).P + a(x:T_2).Q + a(x:T_3).([x=u]PQ)\end{aligned}$$

All results in the paper also hold for the early versions of the equivalences, when rule **SP** is added. For example, by letting the early version of $\sim$ be $\sim^e$, $\mathcal{A}_e$ be $\mathcal{A} \cup \{\mathbf{SP}\}$ and $\mathcal{P}_e$ be $\mathcal{P} \cup \{\mathbf{SP}\}$, we can establish the counterparts of Theorem 15 and 23.

Theorem 36 *(1) $P \sim^e_\Delta Q$ iff $\mathcal{P}_e \vdash P =_\Delta Q$, where P and Q are closed; (2) $P \sim^e_\Delta Q$ iff $\mathcal{A}_e \vdash P =_\Delta Q$.*

Proof. See Appendix B. □

6 Adding parallelism

So far the only π-calculus operator that we have not considered is parallel composition. When it is admitted, Table 1 should be extended with the following three transition rules (their symmetric rules are omitted).

$$\mathsf{par}\ \frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad\qquad \mathsf{com}\ \frac{P \xrightarrow{\bar{a}b} P' \quad Q \xrightarrow{a(x:S)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{b/x\}}$$

$$\mathsf{close}\ \frac{P \xrightarrow{\bar{a}(b:T)} P' \quad Q \xrightarrow{a(x:S)} Q'}{P \mid Q \xrightarrow{\tau} (\nu b : T)(P' \mid Q'\{b/x\})}$$

Table 6

Two rules for parallel composition

Ipar* Assume $\Delta_0\sharp P$ respects Γ_1, $\Delta_0\sharp Q$ respects Γ_2, and $\Delta = \Delta_0, Env(\Delta_0, P, Q, \Gamma_1, \Gamma_2)$.

If $P =_\Delta Q$ and $\Delta \vdash R$ then $P \mid R =_\Delta Q \mid R$

E* Assume $P \equiv \Sigma_i \varphi_i \alpha_i.P_i$ and $Q \equiv \Sigma_j \psi_j \beta_j.Q_j$ where no α_i (resp. β_j) binds a name free in Q (resp. P). Let $\Delta\sharp P \mid Q$ respect Γ. Then infer:

$$P \mid Q =_\Delta \textstyle\sum_i \varphi_i\alpha_i.(P_i \mid Q) + \sum_j \psi_j\beta_j.(P \mid Q_j) + \sum_{\alpha_i\ opp\ \beta_j} [\varphi_i \wedge \psi_j \wedge (u_i = v_j)]\tau.R_{ij}$$

where $\alpha_i\ opp\ \beta_j, u_i, v_j$ and R_{ij} are defined as follows:
1. α_i is $\bar{u}_i w$, β_j is $v_j(x : T)$ and $\Gamma(w) <: T$; then R_{ij} is $P_i \mid Q_j\{w/x\}$;
2. α_i is $\bar{u}_i(w : S)$, β_j is $v_j(x : T)$ and $S <: T$; then R_{ij} is $(\nu w : S)(P_i \mid Q_j\{w/x\})$;
3. the converse of (1) or (2).

In the typed setting, we incorporate the standard typing rule

$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

into Table 2. The TLTS shown in Table 3 is now extended with one rule:

$$\mathsf{Par}\ \frac{\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{\Delta \sharp P \mid Q \xrightarrow{\alpha} \Delta' \sharp P' \mid Q}$$

After the above modifications, all definitions and results in Section 2 are still valid.

To lift the results in Section 3, 4 and 5 to the full π-calculus, it suffices to enrich Table 4 with the two rules in Table 6. As in untyped π-calculus, the expansion law **E*** is used to reduce the parallel composition of two terms into the sum of parallel-free terms. In the typed setting we add conditions on types in order to check the typability of the resulting process R_{ij}. Rule **Ipar*** says that if Δ cannot distinguish P from Q, then it cannot distinguish $P \mid R$ from $Q \mid R$ either, provided that: (i) Δ contains enough fresh channels; (ii) R requires no capabilities beyond the knowledge of Δ. Note that we cannot do without the first condition, i.e., the rule cannot be simplified as:

For any Δ, if $P =_\Delta Q$ and $\Delta \vdash R$ then $P \mid R =_\Delta Q \mid R$

which is unsound for $\sim$ (though it is sound for $\asymp$). The point is that when comparing $P \mid R$ and $Q \mid R$, the observer may first increase his knowledge by interacting with R, then distinguish P from Q by the new knowledge. For example, let $\Delta \stackrel{\text{def}}{=} a : \mathsf{b}T, e : \mathsf{b}T, b : T$ and

$$P \stackrel{\text{def}}{=} a(x : T).[x \neq b]\tau \qquad Q \stackrel{\text{def}}{=} a(x : T).\mathbf{0} \qquad R \stackrel{\text{def}}{=} (\nu c : T)\bar{e}c.$$

It is easy to see that $P \sim_\Delta Q$ and $\Delta \vdash R$ but $P \mid R \not\sim_\Delta Q \mid R$. After the interaction with R, the environment evolves into $\Delta, c : T$. Later the new channel c may be used to instantiate x, thus validating the condition $x \neq b$ and liberating the prefix τ.

The soundness of **E*** is easy to show. To prove that **Ipar*** is sound, we define a family of relations $\mathcal{R} = \{\mathcal{R}_\Delta\}_\Delta$ where

$$\begin{aligned}\mathcal{R}_\Delta = \{((\nu\widetilde{a} : \widetilde{T}_1)(P \mid R), (\nu\widetilde{a} : \widetilde{T}_2)(Q \mid R)) \bigm| & P \sim_{\Delta\sqcap\Delta'} Q,\ \Delta \sqcap \Delta' \vdash R, \\ & \Delta = \Delta_0, Env(\Delta_0, P, Q, \Gamma_1, \Gamma_2),\ \Delta_0 \sqcap \Delta' \sharp P \text{ respects } \Gamma_1, \widetilde{a} : \widetilde{T}_1, \\ & \text{and } \Delta_0 \sqcap \Delta' \sharp Q \text{ respects } \Gamma_2, \widetilde{a} : \widetilde{T}_2,\ \text{for some } \Delta_0, \Delta',\ \Gamma_1, \Gamma_2\}.\end{aligned}$$

Then it can be proved that $\mathcal{R}$ is a typed bisimulation.

In general, if $P \sim_\Delta Q$ then the equality $P =_\Delta Q$ can be inferred in two steps:

(1) By **E***, **Ipar*** and **Twea*** we infer $P =_\Delta P'$ and $Q =_\Delta Q'$, where both P' and Q' are parallel-free terms.
(2) After the above preprocessing job, we infer $P' =_\Delta Q'$ by the proof systems and axiomatisations presented in previous sections.

7 Conclusions and future work

In this work we have constructed a proof system and an axiom system for typed bisimilarity ($\sim$). For the variant bisimilarity proposed in [3], we have provided a proof system for closed terms, and an indirect axiomatisation of all terms that depends on the system of $\sim$. Early versions of the systems are obtained by adding one axiom **SP**. All the systems are proved to be sound and complete.

As partial meet and join operators do not exist in the original capability types [11], we adopt in this work one of their extensions, Hennessy and Rathke's types [3]. An alternative path to take is to go the opposite direction and add some syntactic constraints to capability types, thus only certain shapes of types are legal and partial meet and join operators exist upon the legal types. For instance, in synchronous localised π-calculus there are two forms of legal types: $\mathtt{oo}\cdots\mathtt{o}B$ and $\mathtt{bo}\cdots\mathtt{o}B$ where B is a basic type. It is easy to see that the two operators exist because whenever $T <: S$ holds, then either $T \equiv S$ or $T \equiv \mathtt{b}T', S \equiv \mathtt{o}T'$ for some T', which means:

(1) if $T <: T_1, T_2$ and $T_1 \not\equiv T_2$ then $T_1 \sqcap T_2 = T$;
(2) if $T_1, T_2 <: T$ and $T_1 \not\equiv T_2$ then $T_1 \sqcup T_2 = T$.

Therefore axiomatisation in synchronous localised π-calculus is a special case of the problem addressed in this work.

Below we indicate some further directions of possible future work.

- Due to the difficulty discussed at the end of Section 5.1.2 we are only able to give an indirect axiomatisation of $\asymp$. We are not clear whether it is possible to directly axiomatize the equivalence in the language considered in the current paper.
- In our type system we allow matching names to have arbitrary types. It is not clear how to relax our use of matching. Limiting matching to names of compatible types might pose a problem for subject reduction. On the other hand, allowing matching only on names with types of the form $\mathtt{b}T$, as in [11], would seem difficult, for matching plays an important role in axiomatisations. For example, one would not be able to rewrite $x \mid \bar{y}$ as $x.\bar{y} + \bar{y}.x + [x = y]\tau$ under the type environment $\Delta = x : \mathtt{i}T, y : \mathtt{o}T$. In [3], a particular typing rule for matching was presented, which allowed meet of types on successful matches. It might be interesting to know whether the presence of this typing rule would affect the validity of our proof systems.
- For the variant bisimilarity $\asymp$, as well as the typed bisimilarity defined in [13], there are results that relate them to contextual equivalences such as barbed equivalence. It would be interesting to see what kind of contextual equivalence (if any) corresponds to $\sim$.
- We do not know at present how to adapt our results to the language in [2]. We recall that the main differences are: (i) no distinction between channels and variables, (ii) no matching construct, (iii) the use of Pierce and Sangiorgi's types. Because of (i), some care is needed in a proof system, for instance in defining the appropriate rules for manipulating names that will later be bound in an input. Because of (ii), the expansion law cannot be used without appropriate modification.
- Another issue is axiomatisations of typed *weak* bisimilarities. In this case, however, types may not be so central, in that the addition of the usual tau laws [7] might be sufficient.

Acknowledgements

We are very grateful to the anonymous referees whose comments allowed us to improve the presentation of the paper and to amend some errors in an earlier version.

References

[1] M. Boreale and R. De Nicola. Testing equivalences for mobile processes. *Journal of Information and Computation*, 120:279–303, 1995.

[2] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *Proceedings of LICS '98*. IEEE, Computer Society Press, 1998.

[3] M. Hennessy and J. Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14:651–684, 2004.

[4] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Journal of Information and Computation*, 173:82–120, 2002.

[5] H. Lin. Symbolic bisimulation and proof systems for the π-calculus. Technical Report 7/94, School of Cognitive and Computing Sciences, University of Sussex, UK, 1994.

[6] H. Lin. Complete inference systems for weak bisimulation equivalences in the π-calculus. *Journal of Information and Computation*, 180(1):1–29, 2003.

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

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

[9] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, 1992.

[10] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Journal of Information and Computation*, 120(2):174–197, 1995.

[11] B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[12] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.

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

[14] H. Zantema. Termination. In M. Bezem, J. Klop, and R. de Vrijer, editors, *Term Rewriting Systems*, pages 181–259. Cambridge University Press, 2003.

Appendix

A Some more derived rules:

Cvn $[x = a]P =_\Delta [x = a][x \neq a_1] \cdots [x \neq a_n]P \quad \text{if } a \notin \{a_i \mid 1 \leq i \leq n\}$

Tv2 $P =_\Delta [x = a_1]P + [x = a_2]P + \cdots + [x = a_n]P$

$\text{if } \{b \in \mathit{dom}(\Delta_{\mathtt{c}}) \mid \Delta(b) <: \Delta(x)\} = \{a_1, \cdots, a_n\}$

Tv3 If $P =_{\Delta,x:T} Q$ then $P =_{\Delta,x:S} Q$ for $S <: T$

Iv1 If $P =_{\Delta,y:\Delta(x)_{\mathtt{i}}} Q$ then $x(y : T_1).P =_\Delta x(y : T_2).Q$

Iv2 If $P =_{\Delta \sqcap v:\Delta(x)_{\mathtt{o}}} Q$ then $\bar{x}v.P =_\Delta \bar{x}v.Q$

Proof. Among all the rules, the proof of **Iv2** is the hardest, so we report it below in details and omit the others.

Let $\{b \in \mathit{dom}(\Delta_{\mathtt{c}}) \mid \Delta(b) <: \Delta(x)\} = \{a_1, \cdots, a_n\}$. When $n = 0$, the result is immediate by using **Tv1**. Suppose $n > 0$. For each $i \leq n$, $\Delta(a_i) <: \Delta(x)$, there are two possibilities: (i) if $\Delta(a_i)\not\downarrow_{\mathtt{i}}$ then $\bar{a}_i b.P =_\Delta \mathbf{0} =_\Delta \bar{a}_i b.Q$ by **Tout***; (ii) if $\Delta(a_i)\downarrow_{\mathtt{i}}$, then we have $\Delta(x)_{\mathtt{o}} <: \Delta(a_i)_{\mathtt{o}} <: \Delta(a_i)_{\mathtt{i}}$ by Proposition 2. There are two cases, depending on name v.

- v is a channel, say b. It follows from $P =_{\Delta \sqcap b:\Delta(x)_{\mathtt{o}}} Q$ that $P =_{\Delta \sqcap b:\Delta(a_i)_{\mathtt{i}}} Q$ by **Twea***. Using **Iout***, we have

  $$\bar{a}_i b.P =_\Delta \bar{a}_i b.Q \tag{A.1}$$

  Finally,

  $$\begin{aligned} \bar{x}b.P &=_\Delta [x = a_1]\bar{x}b.P + \cdots + [x = a_n]\bar{x}b.P && \text{by } \textbf{Tv2} \\ &=_\Delta [x = a_1]\bar{a}_1 b.P + \cdots + [x = a_n]\bar{a}_n b.P && \text{by } \textbf{Tpre*} \\ &=_\Delta [x = a_1]\bar{a}_1 b.Q + \cdots + [x = a_n]\bar{a}_n b.Q && \text{by } \textbf{(A.1)} \\ &=_\Delta \bar{x}b.Q && \text{by } \textbf{Tpre*, Tv2} \end{aligned}$$

- v is a variable, say y. By hypothesis, $\Delta\sharp\bar{x}y.P$ and $\Delta\sharp\bar{x}y.Q$ are configurations, then $\Delta(y) <: \Delta(x)_{\mathtt{o}}$. By Proposition 1, we have $\Delta \sqcap y : \Delta(x)_{\mathtt{o}} = \Delta$. Let $\{b \in \mathit{dom}(\Delta_{\mathtt{c}}) \mid \Delta(b) <: \Delta(y)\} = \{b_1, \cdots, b_m\}$. We consider the non-trivial case that $m > 0$. For each $i \leq n, j \leq m$, by Proposition 2 we have

  $$\Delta(b_j) <: \Delta(y) <: \Delta(x)_{\mathtt{o}} <: \Delta(a_i)_{\mathtt{o}} <: \Delta(a_i)_{\mathtt{i}}.$$

  So $\Delta \sqcap b_j : \Delta(a_i)_{\mathtt{i}} = \Delta = \Delta \sqcap y : \Delta(x)_{\mathtt{o}}$. Therefore we can rewrite the hypothesis $P =_{\Delta \sqcap y:\Delta(x)_{\mathtt{o}}} Q$ as $P =_{\Delta \sqcap b_j:\Delta(a_i)_{\mathtt{i}}} Q$. Using **Iout***, we get the result

$$\bar{a}_i b_j.P =_\Delta \bar{a}_i b_j.Q \tag{A.2}$$

At last we can do the inference.

$$\begin{aligned}
& \bar{x}y.P \\
=_\Delta\ & [x = a_1]\bar{x}y.P + \cdots + [x = a_n]\bar{x}y.P && \text{by } \mathbf{Tv2} \\
=_\Delta\ & [x = a_1][y = b_1]\bar{x}y.P + \cdots + [x = a_1][y = b_m]\bar{x}y.P + \\
& \cdots + [x = a_n][y = b_1]\bar{x}y.P + \cdots + [x = a_n][y = b_m]\bar{x}y.P && \text{by } \mathbf{Tv2} \\
=_\Delta\ & [x = a_1][y = b_1]\bar{a}_1 b_1.P + \cdots + [x = a_1][y = b_m]\bar{a}_1 b_m.P + \\
& \cdots + [x = a_n][y = b_1]\bar{a}_n b_1.P + \cdots + [x = a_n][y = b_m]\bar{a}_n b_m.P && \text{by } \mathbf{Tpre*} \\
=_\Delta\ & [x = a_1][y = b_1]\bar{a}_1 b_1.Q + \cdots + [x = a_1][y = b_m]\bar{a}_1 b_m.Q + \\
& \cdots + [x = a_n][y = b_1]\bar{a}_n b_1.Q + \cdots + [x = a_n][y = b_m]\bar{a}_n b_m.Q && \text{by } \mathbf{(A.2)} \\
=_\Delta\ & \bar{x}y.Q && \text{by } \mathbf{Tpre*,\ Tv2}
\end{aligned}$$

□

B The proof of Theorem 36

Proof. We sketch the completeness proof of clause (ii), which is carried out by induction on the depth of $P + Q$; clause (i) can be shown in a similar way. Assume that P, Q are in hnf w.r.t. Δ and $\Delta = \Delta_{\mathtt{c}}, \widetilde{x} : \widetilde{T}$. Let $\Delta\sharp Q$ be a configuration respecting Γ. For some complete condition φ which are satisfiable by some legal substitution on Δ, let $P_{\varphi,a}$ be the sum of all active summands $\varphi_i\alpha_i.P_i$ of P such that $\{\mathbf{C1}, \mathbf{Tpre*}\} \vdash \varphi_i\alpha_i.P_i =_\Delta \varphi a(x : T_i).P_i$. We write

$$P_{\varphi,a} = \sum_{i=1}^{n} \varphi a(x : T_i).P_i \quad \text{and} \quad Q_{\varphi,a} = \sum_{j=1}^{m} \varphi a(x : S_j).Q_j$$

The key of the proof is to find, for each $1 \leq i \leq n$, a term R_i satisfying the following two properties.

$$\mathcal{A}_e \vdash \varphi a(x : T_i).P_i =_\Delta \varphi a(x : \Gamma(a)_{\mathtt{i}}).R_i \tag{B.1}$$

$$\mathcal{A}_e \vdash Q_{\varphi,a} =_\Delta Q_{\varphi,a} + \varphi a(x : \Gamma(a)_{\mathtt{i}}).R_i \tag{B.2}$$

Let $\sigma = \{\widetilde{b}/\widetilde{x}\}$ be a substitution which satisfies φ and $\Delta_{\mathtt{c}} \vdash \widetilde{b} : \widetilde{T}$. From $P\sigma \sim^e_{\Delta_{\mathtt{c}}} Q\sigma$ we derive that $P_{\varphi,a}\sigma \sim^e_{\Delta_{\mathtt{c}}} Q_{\varphi,a}\sigma$. Given $\Delta_{\mathtt{c}}\sharp P_{\varphi,a}\sigma \xrightarrow{a(x:T_i)} \Delta'\sharp P_i\sigma$, for each $b \in \{b \in dom(\Delta_{\mathtt{c}}) \mid \Delta_{\mathtt{c}}(b) <: \Delta_{\mathtt{c}}(a)_{\mathtt{o}}\} = \{c_1, \cdots, c_k\}$ we have a matching transition $\Delta_{\mathsf{c}}\sharp Q_{\varphi,a}\sigma \xrightarrow{a(x:S_{J(i,b)})} \Delta''\sharp Q_{J(i,b)}\sigma$ for some function J from $[1,n]$ and $\{c_i \mid 1 \leq i \leq k\}$ to $[1,m]$. By the definition of hnf, P_i and $Q_{J(i,b)}$ are of the form $\varphi P_i'$ and $\varphi Q'_{J(i,b)}$ respectively. Here φ is complete on $dom(\Delta)$, but not on $dom(\Delta) \cup \{x\}$. We can complete it by adding conditions on the top which respects $\{b/x\}$. Let $\varphi_b = [x = b] \wedge \bigwedge_{u \in dom(\Delta)\backslash b}[x \neq u]$. It is easy to see that

$$([\varphi_b \wedge \varphi]P_i')\sigma\{b/x\} \sim^e_{\Delta_{\mathsf{c}}} ([\varphi_b \wedge \varphi]Q'_{J(i,b)})\sigma\{b/x\}.$$

By Lemma 20 we have $[\varphi_b \wedge \varphi]P_i' \sim^e_{\Delta,x:\Delta(a)_\circ} [\varphi_b \wedge \varphi]Q'_{J(i,b)}$. By induction hypothesis

$$\mathcal{A}_e \vdash \varphi_b P_i =_{\Delta,x:\Delta(a)_\circ} \varphi_b Q_{J(i,b)}. \tag{B.3}$$

Now define $S_{i,l}$ for $l \leq k$ by

$$\begin{aligned} S_{i,1} &= Q_{J(i,c_1)} \\ S_{i,l} &= [x = c_l]\ Q_{J(i,c_l)}\ S_{i,l-1} \quad \text{for } 1 < l \leq k \end{aligned}$$

Let R_i be defined as $S_{i,k}$. Using **C9** and **Cvn**, we decompose binary conditions in R_i into unary conditions.

$$\mathcal{A}_e \vdash R_i =_{\Delta,x:\Delta(a)_\circ} \varphi_{c_k} Q_{J(i,c_k)} + \varphi_{c_{k-1}} Q_{J(i,c_{k-1})} + \cdots + \varphi_{c_1} Q_{J(i,c_1)}$$

On the other hand by **Tv2** and **Cvn** we have

$$\mathcal{A}_e \vdash P_i =_{\Delta,x:\Delta(a)_\circ} \varphi_{c_k} P_i + \cdots + \varphi_{c_1} P_i.$$

By using (B.3) we have $\mathcal{A}_e \vdash P_i =_{\Delta,x:\Delta(a)_\circ} R_i$, from which we infer that $\mathcal{A}_e \vdash a(x : T_i).P_i =_\Delta a(x : \Gamma(a)_{\mathtt{i}}).R_i$ and $\mathcal{A}_e \vdash \varphi a(x : T_i).P_i =_\Delta \varphi a(x : \Gamma(a)_{\mathtt{i}}).R_i$ by **Iin*** and **Icon**. So we get the property in (B.1).

Finally with axiom **SP** we can show by induction on $0 < l \leq k$ that

$$\mathcal{A}_e \vdash Q_{\varphi,a} =_\Delta Q_{\varphi,a} + \varphi a(x : \Gamma(a)_{\mathtt{i}}).S_{i,l}. \tag{B.4}$$

Therefore (B.2) follows because it is the special case of (B.4) when $l = k$. □