UNIVERSITY OF SUSSEX

COMPUTER SCIENCE


# Typed behavioural equivalences for processes in the presence of subtyping

Matthew Hennessy
Julian Rathke

Report 02/2001 March 2001

Computer Science
School of Cognitive and Computing Sciences
University of Sussex
Brighton BN1 9QH

ISSN 1350–3170

# Typed behavioural equivalences for processes in the presence of subtyping

Matthew Hennessy and Julian Rathke

Abstract. We study typed behavioural equivalences for the $\pi$-calculus, in which the type system allows a form of subtyping. This enables processes to selectively distribute different capabilities on communication channels.

The equivalences considered include typed versions of testing equivalences and barbed bisimulation equivalences.

We show that these can be characterised via standard techniques applied to a novel labelled transition system of *configurations*. These consist of a process term together with two related type environments; one constraining the process and the other its computing environment.[1]

## 1 Introduction

Type systems are playing an increasingly important role in the theory of distributed systems. They are essentially a form of static analysis which help in the elimination of run-time errors from programs. Within the theory of distributed systems this intuitive notion of run-time error has been extended to include a diverse range of properties. For example in [12, 3] type systems have been designed to detect potential deadlocks while [18] introduced a system of types for the $\pi$-calculus which are used to control the interpretation of the $\lambda$-calculus. This system of types was extended further in [17] and now forms the basis for the powerful type system implemented in the programming language **Pict**, [19]; related type systems for higher-order concurrent languages may be found in [10, 11]. In papers such as [21, 20] types have been used to manage access control to resources, while in [22] notions of trust have been incorporated in order to protect good host sites from bad computing agents.

Sub-typing is an essential part of most of these systems. For example in **Pict** (according to [19], page 9) it is relatively rare for communication channels to be used for both input and output in the same "region" of a program. Typically servers have one form of access while clients require a different form. These access requirements can be implemented and managed using a subtype relation on the set of types. For example a particular channel may be declared with a type which allows both read and write

[1] Research partially funded by EPSRC grant GR/M71169

access; this channel could be passed to one process, say a server, at a subtype which only allows read, or input access, and passed to a client at a different subtype, allowing write, or output access only. Indeed in papers such as [21, 25] types are viewed as sets of capabilities, such as read access and write access, and sending a name to a process at a subtype amounts to sending it with a subset of the declared capabilities.

The subject of this paper is the investigation of behavioural equivalences for typed process languages, particularly in the presence of subtyping. The type environment in which a process runs obviously affects its behaviour, and therefore behavioural identities. Let us informally write

$$\Gamma \vdash P \simeq Q \qquad (\dagger)$$

to denote that $P$ and $Q$ exhibit the same behaviour when run in an environment constrained by some type environment. The type environment dictates the type at which identifiers may be used and also, indirectly, the names actually in existence; if an identifier is not in the domain of $\Gamma$ then intuitively it can not be used by the process or its environment. Then, using the syntax of the $\pi$-calculus, we would expect the identity

$$\Gamma \vdash (a!\langle v\rangle\, R) \mid P \simeq P$$

if the identifier $a$ is not in the domain of the constraining environment $\Gamma$. It should also be true if it were in the domain but $\Gamma$ dictates that it could only be used to output values. In this case neither $P$ nor the process environment would never be able to exercise the component $a!\langle v\rangle\, R$; to do so would require read access to $a$, which is forbidden by $\Gamma$.

In the presence of subtyping the situation gets more complicated. For example consider the two processes, again expressed in $\pi$-calculus syntax,

$$P \equiv (\mathsf{new}\, c : \mathsf{rw}\langle\rangle)\ \ a!\langle c\rangle \mid c!\langle v\rangle\, S$$
$$Q \equiv (\mathsf{new}\, c : \mathsf{rw}\langle\rangle)\ \ a!\langle c\rangle \mid \mathbf{0}$$

Both generate a new channel $c$ at some type $\mathsf{rw}\langle\rangle$ which allows both read and write access. Now suppose $\Gamma$ is a type environment in which the type associated with $a$ is such that it can only be used to write identifiers which themselves can be used for at most write access. Such types are a standard part of many of the type systems for the $\pi$-calculus, [18]. In this situation we would expect

$$\Gamma \vdash P \simeq Q$$

because no observing process can exercise the $c!\langle v\rangle$ component. This follows since the observing process can only gain knowledge of the new chan-

nel $c$ by receiving it on the channel $a$; but this method of transmission ensures that it can never obtain read access on $c$, because of the type of $a$ in $\Gamma$, and therefore can never activate the component $c!\langle v\rangle$.

Intuitively in (†) the type environment $\Gamma$ constrains both the processes being observed, but also the processes with which they are interacting, the observing processes. However this example shows that in general the type environment of the observer diverges from that of the observed processes, in this case $P$ and $Q$. After the communication on the channel $a$ the observed processes, $P$ and $Q$, are now working relative to the environment $\Gamma$ augmented with the new name $c$ at the type $\mathsf{rw}\langle\rangle$, while the observing process is working with respect to a different type environment, $\Gamma$ augmented with $c$ at a different type; in fact a subtype of $\mathsf{rw}\langle\rangle$.

In papers such as [2, 17] behavioural equivalences have been defined for typed versions of the $\pi$-calculus. These are along the lines that for (†) to be true, $C[P]$ and $C[Q]$ must exhibit the same (simple) behaviour for all contexts $C[\ ]$ which are suitably typed respect to $\Gamma$. Moreover interesting identities have been established, [24]. However all of the proofs involve complicated reasoning over possible contexts, essentially establishing a form of *Context Lemma* in each particular case. This is in contrast to the untyped behavioural equivalences, [13, 8], for which there are a range of powerful techniques based on labelled transition systems. These describe processes in terms of the actions they can perform and their consequences, with judgements of the form

$$P \xrightarrow{\mu} Q$$

In this paper we show that similar techniques can be developed for typed equivalences.

The central idea is to replace the untyped actions above with new judgements of the form

$$\mathcal{I};\Delta \vdash P \xrightarrow{\mu} \mathcal{I}';\Delta' \vdash Q$$

where $\Delta$ represents the type environment of the observed process, $P$, and $\mathcal{I}$ the knowledge that the surrounding context, or observing context, knows of $\Delta$. Performing an action may result in the modification of either, or both, of these environments. Triples of the form $\mathcal{I};\Delta \vdash P$, with minor consistency requirements, are called *configurations* and our judgements endow the set of configurations, **Conf**, with the structure of a label transition system (lts). We show that typed versions of *barbed congruence* and *may* and *must* testing equivalences can be characterised by adapting the standard approaches, [13, 8] to this lts.

We now outline the remainder of the paper. In the following section we

| | | |
|---|---|---|
| $T,\ U$ ::= | | **Terms** |
| | $u?(X : \mathrm{A})\,T$ | Input |
| | $u!\langle v\rangle\,T$ | Output |
| | if $u = v$ then $T$ else $U$ | Matching |
| | $(\mathsf{new}\, n : \mathrm{A})\ T$ | Name Creation |
| | $T \mid T$ | Concurrency |
| | $*T$ | Repetition |
| | $\mathbf{0}$ | Termination |
| $X, Y$ ::= | | **Patterns** |
| | $x$ | variable |
| | $(X_1, \ldots, X_n)$ | tuple |
| $u, v, w$ ::= | | **Values** |
| | $bv$ | base value |
| | $n$ | name |
| | $x$ | variable |
| | $(u_1, \ldots, u_n)$ | tuple |

FIGURE 1. The Syntax

review our version of the $\pi$-calculus, which uses a set of types derived from those in [21], although they are only a minor variation of those from [18]; the section contains a standard operational semantics, in terms of an lts, that is a *labelled transition system*, a type inference system and a statement of Subject Reduction. In Section 3 we define the typed behavioural equivalences which are the main concern of the paper. This is followed by the principal section of the paper, Section 4, where we define the set *typed actions* which gives rise to the lts **Conf**; this section also contains an analysis of **Conf** and proofs of the various properties we require of it. This is followed by two technical sections, Section 5 which contains a characterisation of the typed testing equivalences, and Section 6 which contains a co-inductive characterisation of typed barbed congruence. The paper ends with a short example and a conclusion, which contains a comparison with related work.

## 2 The Language

In this section we review the (polyadic) $\pi$-calculus, its standard operational semantics and the type system we use throughout the paper.

The syntax of the language is given in Figure 1. We presuppose a countable supply of both variables, ranged over by $x, y$, and names, ranged

over by $n, m$. Readers familiar with the $\pi$-calculus will find little of surprise here except perhaps the omission of the non-deterministic choice operator. This operator has little impact with respect to typing, and in particular, subtyping, so we disregard it for the purposes of this paper. The input operator $a?(X : \mathrm{A}) -$ acts as a binder for the variables occurring in the pattern $X$ while $(\mathsf{new}\, n : \mathrm{A})\ -$ binds the name $n$. This gives rise to the usual notion of alpha conversion between terms, $\equiv_\alpha$, and we refer to closed terms, those containing no free occurrences of variables, as *programs*; they are ranged over by $P, Q$. More generally we use $\mathsf{fn}(T)$, $\mathsf{fv}(T)$ to denote the set of free names, variables respectively, in the term $T$. We assume a well-defined capture-free substitution operation $T\{\!|v/X|\!\}$ which allows the value $v$ to be substituted for the pattern $X$ throughout the term $T$; this assumes that $v$ has the same structure as $X$, and as usual we require occurrences of variables in patterns to be unique.

For technical reasons we present the reduction relation $\stackrel{\tau}{\longrightarrow}$ between untyped programs using structural induction and untyped labelled transitions. The generating rules are presented in Figure 2 and are entirely standard for the polyadic $\pi$-calculus. The actions, ranged over by $\mu$, take the form

INPUT: $\stackrel{a?v}{\longrightarrow}$

OUTPUT: $\stackrel{(\tilde{c}:\tilde{\mathrm{C}})a!v}{\longrightarrow}$

REDUCTION: $\stackrel{\tau}{\longrightarrow}$

We use $\mathsf{bn}(\mu)$ for the set of bound names in the action $\mu$, that is, $\mathsf{bn}((\tilde{c} : \tilde{\mathrm{C}})a!v)$ is $\tilde{c}$ and empty otherwise, and write $\mathsf{n}(\mu)$ for all of its names. We will use the $\Longrightarrow$ notation for so-called *weak* transitions in which $\tau$ reductions are abstracted. Specifically, $\stackrel{\tau}{\Longrightarrow}$ is the transitive closure of $\stackrel{\tau}{\longrightarrow}$ and $\Longrightarrow$ is the reflexive closure of $\stackrel{\tau}{\Longrightarrow}$. For labelled transitions, $\stackrel{\alpha}{\Longrightarrow}$, denotes $\Longrightarrow\stackrel{\alpha}{\longrightarrow}\Longrightarrow$ and, for a string of actions $\alpha_1, \ldots, \alpha_n$, $\stackrel{s}{\Longrightarrow}$ refers to the relational composition $\stackrel{\alpha_1}{\Longrightarrow} \ldots \stackrel{\alpha_n}{\Longrightarrow}$.

We now present the types used, and the subtyping relation $<:$, in Figure 3. In addition to some primitive types such as $\mathbf{int}, \mathbf{bool}$ we have types of the form $\mathsf{r}\langle \mathrm{A}\rangle, \mathsf{w}\langle \mathrm{B}\rangle$ and $\{\mathsf{r}\langle \mathrm{A}\rangle, \mathsf{w}\langle \mathrm{B}\rangle\}$, where $\mathrm{A}, \mathrm{B}$ are in turn types. Values allocated these types, respectively, are to be thought of as channel names with the capability to read values of type A, write values of type B or both. We will often use the shorthand $\mathsf{rw}\langle A\rangle$ to mean $\{\mathsf{r}\langle \mathrm{A}\rangle, \mathsf{w}\langle \mathrm{A}\rangle\}$. Since we use a polyadic version of the $\pi$-calculus we also allow tuple-types. It will also be convenient to have a *maximal* type $\top$, which dominates all types in the subtyping relation. Intuitively a name $a$ at this type can not be used for either reading or writing; but our version

(L-OUT)

$$\overline{a!\langle v\rangle\, P \xrightarrow{a!v} P}$$

(L-IN)

$$\overline{a?(X:\mathrm{A})\, P \xrightarrow{a?v} P\{\!\{v/X\}\!\}}$$

(L-OPEN)

$$\frac{P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P'}{(\mathsf{new}\, b:\mathrm{B})\ P \xrightarrow{(b:\mathrm{B})(\widetilde{c}:\widetilde{\mathrm{C}})a!v} P'} \quad \begin{array}{l} b \neq a \\ b \in \mathsf{fn}(v) \end{array}$$

(L-COM)

$$\frac{P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} P',\ \ Q \xrightarrow{a?v} Q'}{P \mid Q \xrightarrow{\tau} (\mathsf{new}\, \widetilde{c}:\widetilde{\mathrm{C}})\ (P' \mid Q')} \quad \tilde{c} \cap \mathsf{fn}(Q) = \emptyset$$

(L-COM)

$$\frac{P \xrightarrow{a?v} P',\ \ Q \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q'}{P \mid Q \xrightarrow{\tau} (\mathsf{new}\, \widetilde{c}:\widetilde{\mathrm{C}})\ (P' \mid Q')} \quad \tilde{c} \cap \mathsf{fn}(P) = \emptyset$$

(L-EQ)

$$\frac{}{\mathsf{if}\ u = w\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} Q} \quad u \neq w \qquad \frac{}{\mathsf{if}\ u = u\ \mathsf{then}\ P\ \mathsf{else}\ Q \xrightarrow{\tau} P}$$

(L-CNTX)

$$\frac{P \xrightarrow{\mu} P'}{P|Q \xrightarrow{\mu} P'|Q} \quad \mathsf{bn}(\mu) \notin \mathsf{fn}(Q) \qquad \frac{P \xrightarrow{\mu} P'}{*P \xrightarrow{\mu} *P \mid P'}$$

$$Q \mid P \xrightarrow{\mu} Q \mid P'$$

$$\frac{P \xrightarrow{\mu} P'}{(\mathsf{new}\, a:\mathrm{A})\ P \xrightarrow{\mu} (\mathsf{new}\, a:\mathrm{A})\ P'} \quad a \notin \mathsf{n}(\mu) \qquad \frac{P \xrightarrow{\mu} P' \qquad P \equiv_\alpha Q}{Q \xrightarrow{\mu} P'}$$

FIGURE 2. The Operational Semantics

**Types**:

$$\frac{}{\top, \mathbf{int}, \mathbf{bool} \in \mathbf{Types}} \qquad \frac{\mathrm{A} \in \mathbf{Types}}{\mathsf{r}\langle \mathrm{A} \rangle \in \mathbf{Types} \quad \mathsf{w}\langle \mathrm{A} \rangle \in \mathbf{Types}}$$

$$\frac{\mathrm{A}, \mathrm{B} \in \mathbf{Types},\ \mathrm{B} <: \mathrm{A}}{\{\mathsf{r}\langle \mathrm{A} \rangle,\ \mathsf{w}\langle \mathrm{B} \rangle\} \in \mathbf{Types}} \qquad \frac{\mathrm{A}_i \in \mathbf{Types}}{(\mathrm{A}_1, \ldots, \mathrm{A}_n) \in \mathbf{Types}}$$

**Subtyping**:

$$\frac{\mathrm{A} <: \mathrm{A}'}{\mathsf{r}\langle \mathrm{A} \rangle <: \mathsf{r}\langle \mathrm{A}' \rangle \quad \{\mathsf{r}\langle \mathrm{A} \rangle, \mathsf{w}\langle \mathrm{B} \rangle\} <: \mathsf{r}\langle \mathrm{A}' \rangle} \qquad \frac{\mathrm{A} <: \mathrm{A}'}{\mathsf{w}\langle \mathrm{A}' \rangle <: \mathsf{w}\langle \mathrm{A} \rangle \quad \{\mathsf{r}\langle \mathrm{B} \rangle, \mathsf{w}\langle \mathrm{A}' \rangle\} <: \mathsf{w}\langle \mathrm{A} \rangle}$$

$$\frac{\mathrm{A} <: \mathrm{A}',\ \mathrm{B} <: \mathrm{B}'}{\{\mathsf{r}\langle \mathrm{A} \rangle, \mathsf{w}\langle \mathrm{B}' \rangle\} <: \{\mathsf{r}\langle \mathrm{A}' \rangle, \mathsf{w}\langle \mathrm{B} \rangle\}} \qquad \frac{}{A <: \top}$$

$$\frac{\mathrm{A}_i <: \mathrm{A}'_i}{(\mathrm{A}_1 \ldots \mathrm{A}_n) <: (\mathrm{A}'_1 \ldots \mathrm{A}'_n)}$$

FIGURE 3. Types

$$\frac{\Gamma(u) <: \mathrm{A}}{\Gamma \vdash u : \mathrm{A}}\ (\text{T-ID}) \qquad \frac{bv \in \mathbf{Base}}{\Gamma \vdash bv : \mathbf{Base}}\ (\text{T-BASE}) \qquad \frac{\Gamma \vdash v_i : A_i \quad (\forall i)}{\Gamma \vdash (v_1, \ldots, v_k) : (\mathrm{A}_1, \ldots, \mathrm{A}_k)}\ (\text{T-TUP})$$

$$\frac{\Gamma, X : \mathrm{A} \vdash T \quad \Gamma \vdash u : \mathsf{r}\langle \mathrm{A} \rangle}{\Gamma \vdash u?(X : \mathrm{A})\, T}\ (\text{T-IN}) \qquad \frac{\Gamma \vdash u : \mathsf{w}\langle \mathrm{A} \rangle \quad \Gamma \vdash v : \mathrm{A} \quad \Gamma \vdash T}{\Gamma \vdash u!\langle v \rangle\, T}\ (\text{T-OUT})$$

$$\frac{\Gamma \vdash u : \mathrm{A}, v : \mathrm{B} \quad \Gamma \vdash U \quad \Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \vdash T}{\Gamma \vdash \mathsf{if}\ u = v\ \mathsf{then}\ T\ \mathsf{else}\ U}\ (\text{T-EQ})$$

$$\frac{\Gamma, a : \mathrm{A} \vdash T}{\Gamma \vdash (\mathsf{new}\, a : \mathrm{A})\, T}\ (\text{T-NEW}) \qquad \frac{\Gamma \vdash T,\ U}{\Gamma \vdash T \mid U,\ *T,\ \mathbf{0}}\ (\text{T-STR})$$

FIGURE 4. The Typing Rules

of the $\pi$-calculus has name matching and therefore a name at type $\top$ can be compared to other names.

Thus our types are a generalisation of those introduced in [18]. The subtyping relation $<:$ can also be viewed as the obvious generalisation of their subtyping relation. In fact our types, and our subtyping relation, are a mild variation of those used in [21], to which the reader is referred for more details, particularly with respect to the following result:

**Proposition 2.1** *The set of types* **Types** *is a preorder with respect to* $<:$*, with both a partial meet operation* $\sqcap$ *and a partial join* $\sqcup$*.* $\Box$.

The essential point here is that if two types $\mathrm{A}_1, \mathrm{A}_2$ are bounded below, that is $B <: \mathrm{A}_1,\ B <: \mathrm{A}_1$ for some type $B$ then they have a greatest lower bound, $\mathrm{A}_1 \sqcap \mathrm{A}_2$. Intuitively $\mathrm{A}_1 \sqcap \mathrm{A}_2$ is the "union of the capabilities" in $\mathrm{A}_1$ and $\mathrm{A}_2$. Because the write capability $\mathsf{w}\langle - \rangle$ is contravariant with respect to $<:$ the definition of $\sqcap$ requires the existence of a partial join $\sqcup$.

We now present the type inference rules for process terms in Figure 4. The judgements are of the form $\Gamma \vdash T$ where $\Gamma$ is a *type environment*, that is a finite mapping from *identifiers*, variables and names, to types.

For an identifier $id$ we write $\Gamma, id : \mathrm{A}$ for the type environment obtained by augmenting $\Gamma$ so as to map $id$ to A; this notation is only defined if $id$ is not already in the domain of $\Gamma$. More generally we use $\Gamma \sqcap id : \mathrm{A}$ to mean the type environment $\Gamma, id : \mathrm{A}$ if $id$ is not in the domain of $\Gamma$ and $\Gamma'$ otherwise, where $\Gamma'$ is equal to $\Gamma$ except possibly at $id$, where $\Gamma'$ takes the value $\Gamma(id) \sqcap A$ (if defined). This notation is generalised in the obvious way to values. We will often write $\Delta$ for *closed* type environments whose

domain consists solely of names.

The reader familiar with the input/output capability types of $\pi$-calculus, [18], should find little surprise in the inference rules except perhaps for the type rule for conditionals, taken from [21]:

$$\frac{\begin{array}{l}(\text{T-EQ})\\ \Gamma \vdash u : \mathrm{A}, v : \mathrm{B}\\ \Gamma \vdash U\\ \Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \vdash T\end{array}}{\Gamma \vdash \mathsf{if}\ u = v\ \mathsf{then}\ T\ \mathsf{else}\ U}$$

In order to establish that $\mathsf{if}\ u = v\ \mathsf{then}\ T\ \mathsf{else}\ U$ is well-typed with respect to the type environment $\Gamma$ we would want to at least check that $T$ and $U$ are well-typed with respect to $\Gamma$ and perhaps one might imagine that $u$ and $v$ have the same type. Given that $u$ and $v$ may be different it is perfectly reasonable, particularly in the presence of subtyping, to allow for the fact that $u$ and $v$ may be channels with very different capabilities and we ought not to insist upon them having necessarily the same types. If however, it transpires that $u$ and $v$ are equal, and the conditional branch containing $T$ is taken, then we have gained extra information in the sense that $u$ and $v$ must have the *same* capabilities and the continuing process $T$ may take advantage of this. The inference rule reflects this reasoning in the hypothesis $\Gamma \sqcap \{u : \mathrm{B}, v : \mathrm{A}\} \vdash T$ (note the switch of types for $u$ and $v$). Recall that the $\sqcap$ operator essentially forms a union of capabilities on types.

Our first obligation is to establish that the reduction relation $\xrightarrow{\tau}$ restricts to the subclass of well-typed terms. Moreover, we provide two useful syntactic properties of typed terms with respect to communication actions. We write $\Delta^r(n)\downarrow$ to indicate that the type environment $\Delta$ at $n$ has a type of the form $\mathsf{r}\langle \mathrm{A}\rangle$ or $\{\mathsf{r}\langle \mathrm{A}\rangle, \mathsf{w}\langle \mathrm{B}\rangle\}$ and, in this situation, we will write $\Delta^r(n)$ to refer to the type A , at which $n$ may read values. Similarly for $\Delta^w(n)$ for types with write capability.

**Theorem 2.2 (Subject Reduction)** *Suppose $\Delta \vdash P$. Then*

- *$P \xrightarrow{\tau} Q$ implies $\Delta \vdash Q$*
- *$P \xrightarrow{a?v} Q$ implies $\Delta^r(a)\downarrow$ and if $\Delta \sqcap v : \Delta^r(a)$ is well-defined then $\Delta \sqcap v : \Delta^r(a) \vdash Q$.*
- *$P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$ implies $\Delta^w(a)\downarrow$, and $\Delta, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \Delta^w(a),\ Q$.*

We end this section by briefly defining a *structural congruence* $\equiv$ of the $\pi$-calculus. This is defined to be the least equivalence extending $\equiv_\alpha$, which

is preserved by the operators of the language, and satisfies the axioms:

$$\begin{aligned}
T|\,(\mathsf{new}\,a)\ U &\equiv (\mathsf{new}\,a)\ (T|U) \ \text{ if } a \notin \mathsf{fn}(T)\\
(\mathsf{new}\,a)\ T &\equiv T \ \text{ if } a \notin \mathsf{fn}(T)\\
T|U &\equiv U|T\\
T|\,\mathbf{0} &\equiv T\\
\mathsf{if}\ u = v\ \mathsf{then}\ (\mathsf{new}\,a)\ T\ \mathsf{else}\ U &\equiv (\mathsf{new}\,a)\ (\mathsf{if}\ u = v\ \mathsf{then}\ T\ \mathsf{else}\ U)\\
&\qquad \text{if } a \notin \mathsf{fn}(T), a \neq u, v\\
\mathsf{if}\ u = v\ \mathsf{then}\ T\ \mathsf{else}\ (\mathsf{new}\,a)\ U &\equiv (\mathsf{new}\,a)\ (\mathsf{if}\ u = v\ \mathsf{then}\ T\ \mathsf{else}\ U)\\
&\qquad \text{if } a \notin \mathsf{fn}(U), a \neq u, v\\
u?(x)\,(\mathsf{new}\,a)\ T &\equiv (\mathsf{new}\,a)\ (u?(x)\,t) \ \text{ if } a \neq u
\end{aligned}$$

We state, without proof, the following well-known properties of this structural congruence:

**Proposition 2.3**

- *For every finite term $T$ (i.e. with no occurrence of replication) there is a* new*free term $T'$, that is a term containing no occurrences of* (new) *, such that $T \equiv (\mathsf{new}\,\tilde{c} : \tilde{C})\ T'$*
- *If $P \equiv Q$ and $P \xrightarrow{\tau} P'$ then there exists some $Q \xrightarrow{\tau} Q'$ such that $P' \equiv Q'$.* □

## 3 Typed Behavioural Equivalences

We are interested in developing behavioural equivalences between processes which take into account the type environment in which the processes are operating. We concentrate on two main approaches, the first based on the ability of observers to discern a difference in the run-time behaviour of processes [16, 8], while the second, usually associated with bisimulation theory [13], uses co-inductive methods.

### *3.1 Testing Preorders*

Here we make explicit the computing context in which a process operates. A *test* or *observer* is a *finite* process with an occurrence of a new reserved name $\omega$, used to report the success of the test. The restriction to finite tests is for convenience only; it is well-known [8] that infinite tests do not result in any extra discriminating power. We let $T$ to range over tests, with the typing rule $\mathcal{I} \vdash \omega!\langle\rangle$ for all type environments $\mathcal{I}$. When placed in parallel with a process $P$, a test may interact with $P$, producing an output on $\omega$ if some desired behaviour of $P$ has been observed. We write

$$P\ \mathbf{may}\ T$$

$T \mid P \xrightarrow{\tau}{}^{*} R$ for some $R$ such that $R$ can *report success*, i.e. $R \xrightarrow{\omega!\langle\rangle}$. The stronger relation

$$P \textbf{ must } T$$

holds when in every computation

$$T \mid P \xrightarrow{\tau} R_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} R_n \xrightarrow{\tau} \ldots$$

there is some $R_k$, $k \geq 0$, which can report success. Behavioural equivalences can now be defined by requiring that processes react in the same manner for a given class of tests. Here it is appropriate to choose the class of tests which are well-typed relative to a given environment $\mathcal{I}$.

**Definition 3.1 (Testing Preorders)** *We write $\mathcal{I} \models P \sqsubseteq_{may} Q$ if $P$ **may** $T$ implies $Q$ **may** $T$, for every test $T$ such that $\mathcal{I} \vdash T$.*
*Similarly $\mathcal{I} \models P \sqsubseteq_{must} Q$ means that for every such $T$, $P$ **must** $T$ implies $Q$ **must** $T$.*
*We use $\eqsim_{may}$ and $\eqsim_{must}$ denote the related equivalence relations.*

Note that in $\mathcal{I} \vdash P \eqsim_{must} Q$ (and similar judgements) the type environment $\mathcal{I}$ is a constraint on the observer, or computing context, rather than the processes $P$, $Q$ themselves; indeed in the definition there is no requirement on $P$, $Q$ to be well-typed. However the alternative characterisations of these relations given in Section 5 depend on them being well-typed in a type environment *compatible* with $\mathcal{I}$.

### *3.2 Co-inductive methods*

In this subsection we use type environments as constraints on the processes themselves rather than their computing context.

**Definition 3.2** *A typed relation over processes consists of a family $\mathcal{R}$ of relations over processes, parametrised by closed type environments,*

$$\mathcal{R} = \{ \mathcal{R}_\Delta \mid \Delta \textit{ closed type environment} \}$$

*which satisfies $P \, \mathcal{R}_\Delta \, Q$ implies $\Delta \vdash P, Q$.*
*We normally write $\Delta \models P \, \mathcal{R} \, Q$ in place of $P \, \mathcal{R}_\Delta \, Q$.*

Typed relations over processes are generalised to arbitrary terms by defining

$$\Delta, \; X : \mathrm{A} \models T \; \mathcal{R}^o \; U$$

to be true if for every value $v$, closed type environment $\Delta'$ disjoint from $\Delta$ and type A such that $\Delta, \Delta' \vdash v : \mathrm{A}$, we have $\Delta, \Delta' \vdash T\{\!|v/X|\!\} \; \mathcal{R} \; U\{\!|v/X|\!\}$. Note that this enables us to substitute *new* values, values which are not

necessarily known to the current type environment $\Gamma$, although it does not allow us to extend the types of values which are already in the domain of $\Delta$. However even on closed terms there may be a difference between a relation $\mathcal{R}$ and its open extension $\mathcal{R}^o$; in general for $\Delta \models P\ \mathcal{R}^o\ Q$ to be true we must have $\Delta, \Delta' \models P\ \mathcal{R}\ Q$ for every allowed $\Delta'$. Note that this is a form of *weakening*.

**Definition 3.3** *A typed relation $\mathcal{R}$ is said to* closed with respect to weakening, *or* w-closed, *if $\mathcal{R}^o = \mathcal{R}$.*

All the behavioural equivalences we will consider will be *w-closed.* to define these we need to consider a number of properties of typed relations.

REDUCTION CLOSED: The typed relation $\mathcal{R}$ is reduction closed whenever $\Delta \models P\ \mathcal{R}\ Q$ and $P \xrightarrow{\tau} P'$ implies there exists some $Q'$ such that $Q \Longrightarrow Q'$ and $\Delta \models P'\ \mathcal{R}\ Q'$.

CONTEXTUAL: Contexts are defined by extending the syntax in Figure 1, allowing *typed holes* $[\cdot_\Gamma]$ in terms. The typing system in Figure 4 is extended to contexts in the obvious way, by adding the rule

$$(\text{T-CXT}) \qquad \frac{}{\Gamma, \Gamma' \vdash [\cdot_\Gamma]}$$

We use $C[\ ]$ to denote contexts with at most one hole and $C[T]$ the term which results from substituting the term $T$ into the hole. We leave the reader to establish

**Proposition 3.4** *$\Gamma' \vdash T$ and $\Gamma \vdash C[\cdot_{\Gamma'}]$ implies $\Gamma \vdash C[T]$.* □

Then we say the typed relation $\mathcal{R}$ is contextual whenever $\Gamma' \models T\ \mathcal{R}^o\ U$ and $\Gamma \vdash C[\cdot_{\Gamma'}]$ implies $\Gamma \models C[T]\ \mathcal{R}^o\ C[U]$.

Unravelling this definition gives the following example consequences for contextual relations over closed terms.

- $\Delta \models P\ \mathcal{R}\ P'$ implies $\Delta, \Delta' \models P\ \mathcal{R}\ P'$
- $\Delta \models P\ \mathcal{R}\ P'$ and $\Delta \vdash Q$ implies $\Delta \models P \mid Q\ \mathcal{R}\ P' \mid Q$.
- $\Delta \models P\ \mathcal{R}\ P'$ and $\Delta \vdash a!\langle v\rangle\,\mathbf{0}$ implies $\Delta \models a!\langle v\rangle\ P\ \mathcal{R}\ a!\langle v\rangle\ P'$.
- If $\Delta \vdash a : \mathsf{r}\langle \mathrm{A}\rangle$ and for every $v$, $\Delta'$, such that $\Delta, \Delta' \vdash v : \mathrm{A}$ we have $\Delta, \Delta' \models T\{\!\{v/X\}\!\}\ \mathcal{R}\ U\{\!\{v/X\}\!\}$ then $\Delta, \Delta' \models a?(X:\mathrm{A})\,T\ \mathcal{R}\ a?(X:\mathrm{A})\,U$.
- $\Delta, a : \mathrm{A} \models P\ \mathcal{R}\ P'$ implies $\Delta \models (\mathsf{new}\, a : \mathrm{A})\ P\ \mathcal{R}\ (\mathsf{new}\, a : \mathrm{A})\ P'$.

The condition on inputs is very natural; $a?(X:\mathrm{A})\,T$ and $a?(X:\mathrm{A})\,U$ are only related if $T\{\!\{v/X\}\!\}$ and $U\{\!\{v/X\}\!\}$ are related for every $v$ and $\Delta'$ such

that $\Delta, \Delta' \vdash v : \mathrm{A}$; this includes values $v$ which are not known in the current environment $\Delta$.

BARB PRESERVING: For a given name $a$ such that $\Delta \vdash a : \mathsf{rw}\langle\top\rangle$. we write $\Delta \models P \Downarrow^{\mathrm{barb}} a$ if there exists some $P'$ such that $P \xrightarrow{\tau}{}^{*} P'$ and $P' \xrightarrow{a\{\!\}}$. Then we say the typed relation $\mathcal{R}$ is *barb preserving* if $\Delta \models P \,\mathcal{R}\, Q$ and $\Delta \models P \Downarrow^{\mathrm{barb}} a$ implies $\Delta \models Q \Downarrow^{\mathrm{barb}} a$.

**Definition 3.5 (Contextual observational equivalence)** *Let $\mathrel{\overset{\sim}{\approx}}_{obs}^{cxt}$ be the largest typed relation over processes which is*

- *symmetric, that is each component of the relation is symmetric*
- *contextual*
- *reduction closed*
- *barb preserving.*

We will usually write this relation in the form $\Delta \models P \mathrel{\overset{\sim}{\approx}}_{\mathrm{obs}}^{\mathrm{cxt}} Q$, and we emphasise that here $\Delta$ is a constraint on the processes themselves, that is $\Delta \vdash P,\ Q$, rather than its context.

One significant property of this behavioural relation is:

**Lemma 3.6** *The relation $\mathrel{\overset{\sim}{\approx}}_{obs}^{cxt}$ is* w-closed. *That is $\Delta \models P \mathrel{\overset{\sim}{\approx}}_{obs}^{cxt} Q$ implies $\Delta, \Delta' \models P \mathrel{\overset{\sim}{\approx}}_{obs}^{cxt} Q$.*

**Proof:** Follows immediately from contextuality. □

In Section 6 we will give a co-inductive characterisation of this relation in the lts **Conf**.

## 4 The LTS of Typed Actions

In this section we formally define the *typed actions* discussed in the Introduction and derive their properties. These actions will form the labelled transition system **Conf**, which can be used to provide characterisations of the behavioural equivalences discussed in the previous section.

### *4.1 Typed Actions*

A type environment $\mathcal{I}$ is compatible with $\Delta$ if

- $\mathcal{I} :> \Delta$
- $\mathsf{dom}(\mathcal{I}) = \mathsf{dom}(\Delta)$

**Definition 4.1** *The triple $\mathcal{I}; \Delta \vdash T$ is a* configuration *if $\Delta$ is a closed type environment such that*

(TYLTS-RED)
$$\frac{P \xrightarrow{\tau} P'}{\mathcal{I};\Delta \vdash P \xrightarrow{\tau} \mathcal{I};\Delta \vdash P'}$$

(TYLTS-OUT)
$$\frac{\mathcal{I}^r(a)\downarrow}{\mathcal{I};\Delta \vdash a!\langle v\rangle\, P \xrightarrow{a!v} \mathcal{I}\sqcap v:\mathcal{I}^r(a);\Delta \vdash P}$$

(TYLTS-IN)
$$\frac{\mathcal{I}^w(a)\downarrow \qquad \mathcal{I}\vdash v:\mathcal{I}^w(a)}{\mathcal{I};\Delta \vdash a?(X:\mathrm{A})\,P \xrightarrow{a?v} \mathcal{I};\Delta \vdash P\{\!\{v/X\}\!\}}$$

(TYLTS-OPEN)
$$\frac{\mathcal{I},b:\top;\Delta,b:\mathrm{B}\vdash P \xrightarrow{(\tilde{c})a!v} \mathcal{I}';\Delta'\vdash P' \quad b\neq a \quad b\in \mathsf{fn}(v)}{\mathcal{I};\Delta\vdash (\mathsf{new}\, b:\mathrm{B})\;\; P \xrightarrow{(b\tilde{c})a!v} \mathcal{I}';\Delta'\vdash P'}$$

(TYLTS-CTXT)
$$\frac{\mathcal{I};\Delta\vdash P \xrightarrow{\mu} \mathcal{I}';\Delta'\vdash P'}{\mathcal{I};\Delta\vdash *P \xrightarrow{\mu} \mathcal{I}';\Delta'\vdash *P \mid P'}$$

(TYLTS-EQUIV)
$$\frac{\mathcal{I};\Delta\vdash P \xrightarrow{\mu} \mathcal{I}';\Delta'\vdash P'}{\mathcal{I};\Delta\vdash Q \xrightarrow{\mu} \mathcal{I};\Delta'\vdash P'}\; P\equiv_\alpha Q$$

$$\frac{\mathcal{I};\Delta\vdash P \xrightarrow{\mu} \mathcal{I}';\Delta'\vdash P'}{\begin{array}{l}\mathcal{I};\Delta\vdash P\mid Q \xrightarrow{\mu} \mathcal{I}';\Delta'\vdash P'\mid Q\\ \mathcal{I};\Delta\vdash Q\mid P \xrightarrow{\mu} \mathcal{I}';\Delta'\vdash Q\mid P'\end{array}}\; \mathsf{bn}(\mu)\notin \mathsf{fn}(Q)$$

$$\frac{\mathcal{I},a:\top;\Delta,a:\mathrm{A}\vdash P \xrightarrow{\mu} \mathcal{I}',a:\top;\Delta',a:\mathrm{A}\vdash P'}{\mathcal{I};\Delta\vdash (\mathsf{new}\, a:\mathrm{A})\;\; P \xrightarrow{\mu} \mathcal{I}';\Delta'\vdash (\mathsf{new}\, a:\mathrm{A})\;\; P'}\; a\notin \mathsf{n}(\mu)$$

FIGURE 5. Typed Actions

- $\mathcal{I}$ *is compatible with* $\Delta$
- $\Delta \vdash T$

The environment $\mathcal{I}$ represents the environment's view of the types allocated to the names in the process. For this reason this view must accord with the actual types allocated to these names. This is guaranteed by requiring $\mathcal{I} :> \Delta$ where $\Delta$ is the actual type context for the term under investigation. Essentially this says that the environment cannot know capabilities for a channel which simply do not exist. The requirement that the domains of the environments be the same is a technical means of ensuring uniqueness of fresh names. We use **Conf**, ranged over by $\mathcal{C}, \mathcal{D}$, to denote the set of all configurations.

The generating rules for the transition system of *typed actions* are defined in Figure 5 and are to be understood as acting on configurations. The rules are obtained from those in Figure 2 by taking the type environment of the computing context, $\mathcal{I}$, into account; essentially actions are only possible if they are allowed by $\mathcal{I}$. Note also that the type annotations on the bound names of output actions are dropped; they are only required in Figure 2 for the definition of the untyped reduction relation $\xrightarrow{\tau}$.

Note also that a priori the rule (TYLTS-OUT) is partial in the sense that the conclusion can only be formed if the extended environment $\mathcal{I} \sqcap v : \mathcal{I}^r(a)$ is well defined. However the first part of the next Proposition establishes that this meet always exists. It also proves that the set of configurations is preserved by the transitions.

**Proposition 4.2**

- *If* $\mathcal{I}^r(v)$ *exists and* $\Delta <: \mathcal{I}$ *then* $\mathcal{I} \sqcap v : \mathcal{I}^r(a)$, *is always defined.*
- *If* $\mathcal{I}; \Delta \vdash P \xrightarrow{\mu} \mathcal{I}'; \Delta' \vdash P'$ *and* $\mathcal{I}; \Delta \vdash P$ *is a configuration then so is* $\mathcal{I}'; \Delta' \vdash P'$.

**Proof:** For the first statement it is sufficient to prove the stronger statement, that $\mathcal{I} \sqcap v : \mathcal{I}^r(a) :> \Delta$. This is done by first noting that these can only differ at $v$ so we use the hypothesis $\mathcal{I} :> \Delta$ to reduce our obligation to showing $\mathcal{I}(v) \sqcap \mathcal{I}^r(a) :> \Delta(v)$. This would follow easily if we knew $\mathcal{I}^r(a) :> \Delta(v)$ as $\mathcal{I}(v) :> \Delta(v)$ clearly holds. So, in order to establish $\mathcal{I}^r(a) :> \Delta(v)$ we observe that, because $\Delta \vdash a!v.P$, there must exist some A such that

(i) $\Delta \vdash a : \mathsf{w}\langle \mathrm{A} \rangle$

(ii) $\Delta \vdash v : \mathrm{A}$.

This means that $\Delta(v) <: \mathrm{A}$ (by (ii)), and that $\mathrm{A} <: \Delta^w(a)$ (by (i)). We know that $\mathcal{I}^r(a)\downarrow$ and, as $\mathcal{I} :> \Delta$, thus $\Delta^r(a)\downarrow$ also. By virtue of being a well-formed type it must be the case that $\Delta^w(a) <: \Delta^r(a)$ and, in turn, $\Delta^r(a) <: \mathcal{I}^r(a)$. Collecting these together, transitivity of $<:$ gives the required result.

For the second statement we proceed by rule induction. For the most part this is straightforward, the only involved case arises when $\xrightarrow{\mu}$ has been derived as an instance of the (TYLTS-OUT) rule. We demonstrate this case here.

We are to show that $\mathcal{I} \sqcap v : \mathcal{I}^r(a); \Delta \vdash P$ is a configuration given that $\Delta \vdash a!v.P$, $\mathcal{I} :> \Delta$, and $\mathcal{I}^r(a)\downarrow$. It is easy to see from the type inference rules that $\Delta \vdash P$ and in the first part we have already established the other requirement, namely that $\mathcal{I} \sqcap v : \mathcal{I}^r(a) :> \Delta$. □

It follows that (**Conf**, *Act*) is indeed a labelled transition system, (lts). In the next two sections we show that the various typed behaviour relations over processes can be characterised by adapting the standard definitions, [14, 8] to this lts.

We end this section by establishing various properties which will be required of typed actions. We first determine under what circumstances the untyped actions, in Figure 2, can give rise to typed actions.

**Lemma 4.3** *Suppose $\mathcal{I}; \Delta \vdash P$ is a configuration.*

- $\mathcal{I}; \Delta \vdash P \xrightarrow{\tau} \mathcal{I}; \Delta \vdash Q$ *if and only if* $P \xrightarrow{\tau} Q$
- $\mathcal{I}; \Delta \vdash P \xrightarrow{a?v} \mathcal{I}; \Delta \vdash Q$ *if and only if* $\mathcal{I}^w(a)\downarrow$, $\mathcal{I} \vdash v : \mathcal{I}^w(a)$ *and* $P \xrightarrow{a?v} Q$.
- $\mathcal{I}; \Delta \vdash P \xrightarrow{(\tilde{c})a!v} \mathcal{I} \sqcap v : \mathcal{I}^r(a); \Delta, \tilde{c} : \tilde{\mathrm{C}} \vdash Q$ *if and only if* $\mathcal{I}^r(a)\downarrow$ *and* $P \xrightarrow{(\tilde{c}:\tilde{\mathrm{C}})a!v} Q$.

**Proof:** Straightforward rule induction. □

It turns out that these are the only form of typed actions which can be deduced from the rules. Note that the action determines completely the resulting type environment of the observer, the change from $\mathcal{I}$ to $\mathcal{I}'$, but not necessarily that of the process itself. To make this precise let us define the partial predicate *after* $\mu$ over environments as follows:

$\mu = \tau$: Here $\mathcal{I}$ *after* $\mu$ is always defined to be $\mathcal{I}$.

$\mu = a?v$: Here $\mathcal{I}$ *after* $\mu$ is also defined to be $\mathcal{I}$.

$\mu = (\tilde{c})a!v$: Here it is only defined if $\mathcal{I}^r(a)$ exists and $\mathcal{I} \sqcap v : \mathcal{I}^r(a)$ is well-defined; in which case it is this last updated environment.

We leave the reader to check that

**Lemma 4.4** *If* $\mathcal{I};\Delta \vdash P \xrightarrow{\mu} \mathcal{I}';\Delta' \vdash Q$ *then* $\mathcal{I}$ after $\mu$ *exists and* $\mathcal{I}' = \mathcal{I}$ after $\mu$. □

So to determine the precise form of a typed action it is now sufficient to describe the resulting type environment of the process, the change from $\Delta$ to $\Delta'$. We can now give the precise form of typed actions.

**Lemma 4.5** *Suppose* $\mathcal{I};\Delta \vdash P \xrightarrow{\mu} (\mathcal{I} \text{ after } \mu);\Delta' \vdash Q$. *Then*

$\mu = \tau$: *Here* $\Delta' = \Delta$.

$\mu = a?v$: *Here* $\Delta' = \Delta$.

$\mu = (\tilde{c})a!v$: *Here* $\Delta' = \Delta, \tilde{c} : \tilde{\mathrm{C}}$ *for some* $\tilde{\mathrm{C}}$ *such that* $\Delta, \tilde{c} : \tilde{\mathrm{C}} \vdash v : \mathcal{I}^r(a)$.

**Proof:** Again, a straightforward rule induction suffices. □

*4.2 Contextuality of labelled transitions*

An important feature of our labelled transitions of typed actions is that the observations we make of processes are contextually valid. That is to say, the method of the environment learning information regarding communication capabilities from processes by means of the labelled transition system can actually be realised with well-typed code fragments, or contexts, written in our language. The formalisation of this idea simply identifies each label $\alpha$ of the transition system (other than $\tau$) with a small testing context of the language with the appropriate properties. These properties essentially say that $\alpha$ transitions induce reductions in $\alpha$ contexts and vice-versa. There is some added technical complications which allow the environment's type information to change over time. After each observation we use the testing context to *export* all the information about the current state of knowledge to the environment.

The following abbreviations will be useful: if $\mathcal{I}$ is the type environment $v_1 : \mathrm{A}_1, \ldots, v_n : \mathrm{A}_n$ then we write $(\mathcal{I})$ to represent the tuple type $(\mathrm{A}_1, \ldots, \mathrm{A}_n)$ and we write $v_{\mathcal{I}}$ to mean the tuple of values $(v_1, \ldots, v_n)$.

**Proposition 4.6** *For each label* $\alpha$ *and each environment* $\mathcal{I}$ *compatible with* $\Delta$, *there exists a process* $\mathcal{C}^{\mathcal{I}}_{\alpha}$, *using a fresh name* $\delta$, *such that*

*(i) if* $\mathcal{I};\Delta \vdash P \xrightarrow{\alpha} \mathcal{I}';\Delta,\Delta' \vdash P'$ *then* $\mathcal{I}, \delta : \mathsf{rw}\langle \mathrm{A} \rangle \vdash \mathcal{C}^{\mathcal{I}}_{\alpha}$ *and* $P \mid \mathcal{C}^{\mathcal{I}}_{\alpha} \Longrightarrow (\mathsf{new}\,\Delta')\ P' \mid \delta!\langle v_{\mathcal{I}'} \rangle$, *where* $\mathrm{A} = (\mathcal{I}')$

*(ii) if* $P \mid \mathcal{C}^{\mathcal{I}}_{\alpha} \Longrightarrow (\mathsf{new}\,\Delta')\ P' \mid \delta!\langle v_0 \rangle$ *for some* $v_0$ *and* $\mathcal{I}, \delta : \mathsf{rw}\langle A \rangle \vdash \mathcal{C}^{\mathcal{I}}_{\alpha}$ *with* $\delta \notin \Delta'$ *and* $\mathsf{dom}(\Delta') = \mathsf{bn}(\alpha)$ *then* $\mathcal{I};\Delta \vdash P \overset{\alpha}{\Longrightarrow} (\mathcal{I} \text{ after } \alpha);\Delta,\Delta' \vdash P'$

**Proof:** For ease of presentation we restrict ourselves to terms of the monadic $\pi$-calculus, *i.e.* no tuple types; it is a simple but tedious procedure to extend this to the full polyadic language. Given an environment $\mathcal{I}$ and label $\alpha$ we choose a fresh name $\delta$ of type $\mathsf{rw}\langle(\mathcal{I}')\rangle$ and let B be the type $\mathcal{I}^r(a)$, if defined, and otherwise $\top$. Define the testing processes $\mathcal{C}^{\mathcal{I}}_{\alpha}$ as follows:

- $\mathcal{C}^{\mathcal{I}}_{a!b} = a?(x:\mathrm{B})\,\mathsf{if}\ x = b\ \mathsf{then}\ \delta!\langle v_{\mathcal{I}\sqcap b:\mathrm{B}}\rangle\ \mathsf{else}\ \mathbf{0}$
- $\mathcal{C}^{\mathcal{I}}_{(b)a!b} = a?(x:\mathrm{B})\,\mathsf{if}\ x \notin \mathcal{I}\ \mathsf{then}\ \delta!\langle v_{\mathcal{I},x:\mathrm{B}}\rangle\ \mathsf{else}\ \mathbf{0}$
- $\mathcal{C}^{\mathcal{I}}_{a?b} = a!b.\delta!\langle v_{\mathcal{I}}\rangle$

where $x \notin \mathcal{I}$ is coded using nested conditionals to check that $x$ is not equal to any of the names in $\mathcal{I}$.

As an example of (i) we consider the case in which $\alpha$ is $a!b$;the other cases are similar. Note that the hypothesis, $\mathcal{I};\Delta \vdash P \xrightarrow{\alpha} \mathcal{I}';\Delta,\Delta' \vdash P'$, ensures that $\mathcal{I}^r(a)$ is defined and therefore $\mathcal{C}^{\mathcal{I}}_{\alpha}$ is well typed.

We first show $P \,|\, \mathcal{C}^{\mathcal{I}}_{\alpha} \Longrightarrow (\mathsf{new}\,\Delta')\ P' \,|\, \delta!\langle v_{\mathcal{I}'}\rangle$. Applying the hypothesis to Lemma 4.3 we obtain

$$P \xrightarrow{a!b} P'$$

and, by definition,

$$\mathcal{C}^{\mathcal{I}}_{a!\langle b\rangle} \xrightarrow{a?b} \longrightarrow \delta!\langle v_{\mathcal{I}'}\rangle$$

These can interact to yield:

$$P \,|\, \mathcal{C}^{\mathcal{I}}_{a!\langle b\rangle} \Longrightarrow P' \,|\, \delta!\langle v_{\mathcal{I}'}\rangle$$

as required.

To show that the testing process is well-typed let $\Gamma$ be the type context $x:\mathrm{B}, \mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \sqcap b:\mathrm{B} \sqcap x : \mathcal{I}(b)$ where $\mathcal{I}'$ is $\mathcal{I} \sqcap b:\mathrm{B}$ in this case. It is easy to check that

$$\Gamma \vdash \delta : \mathsf{w}\langle(\mathcal{I}')\rangle \quad \text{and} \quad \Gamma \vdash v_{\mathcal{I}\sqcap b:\mathrm{B}} : (\mathcal{I}')$$

thus, by applying the typing rule (T-OUT) we know $\Gamma \vdash \delta!\langle v_{\mathcal{I}\sqcap b:\mathrm{B}}\rangle$. This can now be used as a hypothesis to rule (T-EQ) to infer

$$x:\mathrm{B}, \mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \vdash \mathsf{if}\ x = b\ \mathsf{then}\ \delta!\langle v_{\mathcal{I}\sqcap b:\mathrm{B}}\rangle\ \mathsf{else}\ \mathbf{0}\,.$$

We know that $\mathrm{B} = \mathcal{I}^r(a)$ so $\mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \vdash a : \mathsf{r}\langle\mathrm{B}\rangle$ and this, combined with the previous judgement, allow us to apply (T-IN) to obtain

$$\mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \vdash a?(x:\mathrm{B})\,\mathsf{if}\ x = b\ \mathsf{then}\ \delta!\langle v_{\mathcal{I}'}\rangle\ \mathsf{else}\ \mathbf{0} \quad (= \mathcal{C}^{\mathcal{I}}_{a!b}).$$

The reader should note the essential use of the accumulated type information in the hypothesis of the type rule (T-EQ).

For the converse, part (ii), we use the case $\alpha = a?b$ as an illustrative example. Suppose then that $\mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I})\rangle \vdash \mathcal{C}^{\mathcal{I}}_{\alpha}$ and

$$P \mid \mathcal{C}^{\mathcal{I}}_{\alpha} \Longrightarrow P' \mid \delta!\langle v_0 \rangle$$

for some $v_0$. It must be the case, as $\delta$ is fresh to $P$, that $v_0$ is $v_{\mathcal{I}}$ and, by analysis of the reduction rules, that $P \Longrightarrow \xrightarrow{a?b} \Longrightarrow P'$. Now, we know that $\mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I})\rangle \vdash \mathcal{C}^{\mathcal{I}}_{\alpha}$ so from this we can deduce that this must have been inferred from $\mathcal{I} \vdash a : \mathsf{w}\langle \mathrm{B} \rangle$ and $\mathcal{I} \vdash b : \mathrm{B}$ for some type B such that $\mathrm{B} <: \mathcal{I}^w(a)$. This ensures $\mathcal{I}^w(a) \downarrow$ and $\mathcal{I} \vdash b : \mathcal{I}^w(a)$, allowing us to apply Lemma 4.3 to obtain the required

$$\mathcal{I}; \Delta \vdash P \Longrightarrow \xrightarrow{a?b} \Longrightarrow \mathcal{I}; \Delta \vdash P'$$

□

## 5 Characterising the Testing Preorders

In this section we give alternative characterisations of the two testing preorders. It should be emphasised that these are obtained by applying the *completely standard* definitions of traces and acceptance sets, [8], not to the lts obtained from the operational semantics given in Figure 2, but to the lts of types actions, **Conf**. However a priori there is a mismatch between formalisation of **Conf** and the definition of the testing preorders in Section 3.1; the latter uses only one type environment, that of the observer, while the former uses two type environments, one for the observer and the other for the process under observation. To rectify this situation we adapt the testing preorders, in a trivial manner, to *typed processes*, that is processes with their typing environment.

**Definition 5.1 (Testing Preorders)** *Let* $\mathcal{I} \models (\Delta \vdash P) \lesssim_{may} (\Delta' \vdash Q)$ *if*

- *both* $\mathcal{I}; \Delta \vdash P$ *and* $\mathcal{I}; \Delta \vdash P$ *are configurations*
- $\mathcal{I} \vdash P \lesssim_{may} Q$

*The preorder* $\lesssim_{must}$ *is generalised in a similar manner.*

To study these relations we need to generalise various properties originally defined for typed relations, that is relations parametrised over single type environments, to relations over configurations, such as $\lesssim_{may}$ and $\lesssim_{must}$ in the above definition. So for example a natural generalisation of the definition of *open extension*, on page 11, is to allow $\mathcal{I} \models (\Delta \vdash P) \ll^{o}_{may} (\Delta \vdash Q)$ whenever $\mathcal{I}, \mathcal{I}' \models (\Delta, \mathcal{I}' \vdash P) \ll_{may} (\Delta, \mathcal{I}' \vdash Q)$, for every $\mathcal{I}'$ disjoint from $\mathcal{I}$. With this definition we have:

**Proposition 5.2** *The relations $\sqsubseteq_{may}$ and $\approx_{must}$ over configurations are* w-closed.

**Proof:** A simple corollary of Lemma 2.3. □

We now give alternative characterisations to these behavioural preorders.

### *5.1 May Testing*

Typed actions are extended to *typed traces* a straightforward manner:

- $\mathcal{I};\Delta \vdash P \stackrel{\varepsilon}{\Longrightarrow} \mathcal{I};\Delta \vdash P$
- $\mathcal{I};\Delta \vdash P \stackrel{\tau}{\longrightarrow} \mathcal{I};\Delta' \vdash P'$ and $\mathcal{I};\Delta' \vdash P' \stackrel{s}{\Longrightarrow} \mathcal{I};\Delta'' \vdash P''$ implies $\mathcal{I};\Delta \vdash P \stackrel{s}{\Longrightarrow} \mathcal{I};\Delta'' \vdash P''$
- $\mathcal{I};\Delta \vdash P \stackrel{\alpha}{\longrightarrow} \mathcal{I};\Delta' \vdash P'$ and $\mathcal{I};\Delta' \vdash P' \stackrel{s}{\Longrightarrow} \mathcal{I};\Delta' \vdash P''$ implies $\mathcal{I};\Delta \vdash P \stackrel{\alpha\cdot s}{\Longrightarrow} \mathcal{I};\Delta'' \vdash P''$

The characterisation depends on a Decomposition and Composition result for these sequences. This requires an asymmetric definition of *complementary action.* For a visible action $\alpha$ we let $\overline{\alpha}$ denote $a!v$ if $\alpha$ is $a?v$ and $a?v$ if $\alpha$ has the form $(\tilde{c}:\tilde{\mathrm{C}})a?v$. Thus $\overline{\alpha}$ transforms an action from the untyped semantics in Figure 2 to one from the typed semantics in Figure 5. It is extended to sequences in the natural way.

**Theorem 5.3 (Trace Decomposition)** *Suppose $T \mid P \stackrel{\tau}{\longrightarrow}^* R$ where $\mathcal{I};\Delta \vdash P$ is a configuration such that $\mathcal{I} \vdash T$, and $T$ is* new*free. Then there exists a typed trace $\mathcal{I};\Delta \vdash P \stackrel{s}{\Longrightarrow} \mathcal{I}';\Delta' \vdash P'$ and a derivation $T \stackrel{\overline{s}}{\Longrightarrow} T'$, where $R \equiv (\tilde{c}:\tilde{\mathrm{C}})(T' \mid P')$, for some $(\tilde{c}:\tilde{\mathrm{C}})$.*

**Proof:** By induction on the derivation $T \mid P \stackrel{\tau}{\longrightarrow}^* R$. The inductive case is when it has the form $T \mid P \stackrel{\tau}{\longrightarrow}\stackrel{\tau}{\longrightarrow}^* R$ and a case analysis is required on the first move. The interesting case is where there is communication between $T$ and $P$.

Output from $T$ to $P$: In this case we have $T \mid P \stackrel{\tau}{\longrightarrow} T' \mid P' \stackrel{\tau}{\longrightarrow}^* R$ where

$$T \stackrel{a!v}{\longrightarrow} T'$$
$$P \stackrel{a?v}{\longrightarrow} P'$$

since $T$ is new*free*. Applying Subject Reduction to the first reduction we obtain $\mathcal{I}^w(a)\downarrow$ and $\mathcal{I} \vdash v : \mathcal{I}^w(a)$, which enables us to apply the second part of Lemma 4.3 to obtain the typed action $\mathcal{I};\Delta \vdash P \stackrel{a?v}{\longrightarrow} \mathcal{I};\Delta \vdash P'$. Subject Reduction also gives $\mathcal{I} \vdash T'$ and therefore induction may be applied to obtain the rest of the typed trace.

OUTPUT FROM $P$ TO $T$: Here we have

$$T \mid P \xrightarrow{\tau} (\mathsf{new}\, \tilde{d} : \tilde{D})\ (T' \mid P') \xrightarrow{\tau}{}^{*} (\mathsf{new}\, \tilde{d} : \tilde{D})\ R'$$

where

$$T \xrightarrow{a?v} T'$$
$$P \xrightarrow{(\tilde{d}:\tilde{D})a!v} P'.$$

Again we can apply Subject Reduction to the first action to obtain $\mathcal{I}^r(a) \downarrow$ and we can apply the third part of Lemma 4.3, this time to obtain the typed action $\mathcal{I}; \Delta \vdash P \xrightarrow{(\tilde{d}:\tilde{D})a!v} \mathcal{I} \sqcap v : \mathcal{I}^r(a); \Delta, \tilde{d} : \tilde{\mathrm{D}} \vdash Q$

Also, since $\mathcal{I} \sqcap v : \mathcal{I}^r(a)$ is defined, Subject Reduction gives $\mathcal{I} \sqcap v : \mathcal{I}^r(a) \vdash T'$ and so we can apply induction to the sequence $T' | P' \xrightarrow{\tau}{}^{*} R'$ to obtain the remainder of the typed trace.

□

Note that this result is not true if $T$ contains any occurrences of $(\mathsf{new}\, n)\ (\ )$.

**Example 5.4** *Suppose $T$, $P$ represent the terms $(\mathsf{new}\, c : \mathrm{C})\ a!\langle c\rangle\ c?()\, T'$ and $a?(x)\ x!\langle\rangle\, P'$ respectively, where $\mathrm{C}$ is the type $\mathsf{rw}\langle\rangle$, and suppose that $\mathcal{I}$ and $\Delta$ are compatible environments such that $\mathcal{I} \vdash T$, $\Delta \vdash P$ and $\mathcal{I}^r(a) = \Delta^r(a) = \mathsf{w}\langle\rangle$; these are easy to construct. Then the derivation $T | P \xrightarrow{\tau} \xrightarrow{\tau}$ $(\mathsf{new}\, c : \mathrm{C})\ T' \mid P'$ can not be decomposed.*

*This is a consequence of the assumption built into our* configurations *$\mathcal{I}; \Delta \vdash P$, that $\mathcal{I} <: \Delta$.*

**Theorem 5.5 (Trace Composition)** *Suppose $\mathcal{I}; \Delta \vdash P \stackrel{s}{\Longrightarrow} \mathcal{I}; \Delta' \vdash P'$ and $T \stackrel{\bar{s}}{\Longrightarrow} T'$. Then there exists a derivation $T \mid P \xrightarrow{\tau}{}^{*} (\tilde{c} : \tilde{\mathrm{C}})(T' \mid P')$, for some $(\tilde{c} : \tilde{\mathrm{C}})$.*

**Proof:** Straightforward induction on $s$. □

These two results enable us to state our first characterisation result.

**Definition 5.6** *For any typed process $\Delta \vdash P$ let*

$$\mathrm{Seq}_{\mathcal{I}}(\Delta \vdash P) = \{\, s \mid \mathcal{I}; \Delta \vdash P \stackrel{s}{\Longrightarrow} \,\}$$

*Then we write*

$$\mathcal{I} \models (\Delta \vdash P) \ll_{may} (\Delta \vdash Q)$$

*if* $\mathrm{Seq}_{\mathcal{I}}(\Delta \vdash P) \subseteq \mathrm{Seq}_{\mathcal{I}}(\Delta \vdash Q)$.

Our aim is to show that $\sqsubseteq_{may}$ coincides with the open extension of $\ll_{may}$, $\ll^o_{may}$. It is the use of the open extension which enables us to capture the effect of tests which generate new names, unknown to the process being observed.

**Theorem 5.7 (Alternative Characterisation of May Testing)**
$\mathcal{I} \vdash (\Delta \vdash P) \sqsubseteq_{may} (\Delta' \vdash Q)$ *if and only if* $\mathcal{I} \models (\Delta \vdash P) \ll^o_{may} (\Delta' \vdash Q)$.

**Proof:**(Outline) First suppose $\mathcal{I} \models (\Delta \vdash P) \ll^o_{may} (\Delta' \vdash Q)$. and $P$ **may** $T$ for some test $T$ such that $\mathcal{I} \vdash T$. Using the structural congruence the test $T$ may be written as $T \equiv (\mathsf{new}\, \tilde{c} : \tilde{C})\ T'$, where $T'$ is $\mathsf{new}free$; moreover it is easy to check, using Proposition 2.3, that for any process $R$, $R$ **may** $T$ if and only if $R$ **may** $T'$. So we establish $Q$ **may** $T'$.

From $P$ **may** $T'$ we know there is a computation

$$T' \mid P \xrightarrow{\tau} R_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} R_n$$

for some $R_n$ which reports success. We know $\mathcal{I}, \tilde{c} : \tilde{C} \vdash T'$, and by weakening $\Delta, \tilde{c} : \tilde{C} \vdash P$. Therefore using Trace Decomposition we can obtain

$$\mathcal{I}, \tilde{c} : \tilde{C}; \Delta, \tilde{c} : \tilde{C} \vdash P \stackrel{s}{\Longrightarrow} \mathcal{I}'; \Delta' \vdash P'$$
$$T' \stackrel{\bar{s}}{\Longrightarrow} T''$$

where $T'' \xrightarrow{\omega\langle\!\rangle}$. The hypothesis ensures

$$Seq_{\mathcal{I},\tilde{c}:\tilde{C}}(\Delta, \tilde{c} : \tilde{C} \vdash P) \subseteq Seq_{\mathcal{I},\tilde{c}:\tilde{C}}(\Delta', \tilde{c} : \tilde{C} \vdash Q)$$

and therefore we have a typed trace

$$\mathcal{I}, \tilde{c} : \tilde{C}; \Delta', \tilde{c} : \tilde{C} \vdash Q \stackrel{s}{\Longrightarrow}$$

Now Trace Composition can be used to obtain a successful computation from $T' \mid Q$.

The converse requires the definition of testing processes which can determine if a process can perform a particular trace. These use the terms $\mathcal{C}^{\mathcal{I}}_{\alpha}$, defined in Section 4.2, and since their construction is very similar to those used in Theorem 5.11 below. We leave the details to the reader. □

### *5.2 Must Testing*

The characterisation of *must* testing requires a notion of *convergence* and *acceptance set* for the lts **Conf**.

**Definition 5.8 (Convergence)** *We say the configuration* $\mathcal{C}$ converges,

*written* $\mathcal{C} \Downarrow$, *if there is no infinite sequence of derivations*

$$\mathcal{C} \xrightarrow{\tau} \mathcal{C}_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} \mathcal{C}_k \xrightarrow{\tau}$$

*This is generalised to trace sets by*

$\mathcal{C} \Downarrow \varepsilon$ *if* $\mathcal{C} \Downarrow$

$\mathcal{C} \Downarrow \alpha \cdot s$ *if* $\mathcal{C} \Downarrow$ *and* $\mathcal{C}' \Downarrow s$ *whenever* $\mathcal{C}' \xrightarrow{\alpha} \mathcal{C}'$

**Definition 5.9 (Acceptance Sets)** *For any configuration let its* ready set *be defined by*

$$\mathcal{R}(\mathcal{C}) = \{\, a? \mid \exists v. \mathcal{C} \xrightarrow{a?v} \,\} \cup \{\, a! \mid \exists v, \tilde{c}. \mathcal{C} \xrightarrow{(\tilde{c})a!v} \,\}.$$

*Then its* acceptance set after $S$ *is given by*

$$\mathrm{Acc}(\mathcal{C}, s) = \{\, \mathcal{R}(\mathcal{C}') \mid \mathcal{C} \stackrel{s}{\Longrightarrow} \mathcal{C}' \not\xrightarrow{\tau} \,\}.$$

**Definition 5.10** *We write* $\mathcal{I} \models (\Delta \vdash P) \ll_{must} (\Delta' \vdash Q)$ *if for every trace* $s$

$$(\mathcal{I}; \Delta \vdash P) \Downarrow s \text{ implies } (a)\ (\mathcal{I}; \Delta' \vdash Q) \Downarrow s$$
$$(b)\ \forall D \in \mathrm{Acc}(\mathcal{I}; \Delta' \vdash Q, s)$$
$$\exists C \in \mathrm{Acc}(\mathcal{I}; \Delta \vdash P, s) \text{ such that } C \subseteq D.$$

**Theorem 5.11 (Must Characterisation)** $\mathcal{I} \models (\Delta \vdash P) \sqsubseteq_{must} (\Delta' \vdash Q)$ *if and only if* $\mathcal{I} \models (\Delta \vdash P) \ll^{o}_{must} (\Delta' \vdash Q)$.

**Proof:** We leave the reader to prove one direction, $\mathcal{I} \models (\Delta \vdash P) \ll^{o}_{must} (\Delta' \vdash Q)$ implies $\mathcal{I} \models (\Delta \vdash P) \sqsubseteq_{must} (\Delta' \vdash Q)$. It follows the lines of Lemma 4.4.13 and Theorem 2.2.5 of [8], but using the Decomposition and Composition Theorems given above; as in Theorem 5.7 tests with new names are handled via the open extension.

To prove the converse we need to show that if $\mathcal{I}, \mathcal{I}' \models \Delta, \mathcal{I}' \vdash P \not\ll_{must} \Delta', \mathcal{I}' \vdash Q$ then there is a test $T$ such that $P$ **must** $T$ while $Q$ $\not$**must** $T$. We only consider the case when $\mathcal{I}'$ is empty; it contains all the ingredients of the more general case.

There are two possible reasons for $\mathcal{I} \models \Delta \vdash P \not\ll_{must}; \Delta' \vdash Q$ not being true, concerning convergence and acceptances respectively. Let us first consider convergence. Here the situation is that $\mathcal{I}; \Delta \vdash P \Downarrow s$, for some $s$, but $\mathcal{I}; \Delta' \vdash Q \stackrel{s}{\Longrightarrow} \mathcal{I}'; \Delta'' \vdash Q'$ where $Q'$ diverges. We construct a test $C(s)$ such that $\mathcal{I} \vdash C(s)$, $P$ **must** $C(s)$, and $Q$ $\not$**must** $C(s)$. The definition uses the following notation:

- $P \oplus Q$ is used as shorthand for $(\mathsf{new}\, n)\ n!\langle\rangle \,|\, n?()\, P \,|\, n?()\, Q$, the internal choice between $P$ and $Q$.

- $\mathcal{C}^{\mathcal{I}}_{\alpha,n}$ will be used to denote the context associated with the action $\alpha$ in Proposition 4.6, however with occurrences of $\delta!\langle v\rangle$ replaced by $n?()\,\delta!\langle v\rangle$.

If $s$ is the empty sequence then $C(s)$ is simply $\omega!\langle\rangle \oplus \omega!\langle\rangle$. Otherwise suppose it has the form $\alpha \cdot s'$. Then $C(s)$ is defined to be

$$(\mathsf{new}\, n)\ n!\langle\rangle \mid n?()\,\omega!\langle\rangle \quad \mid \\ (\mathsf{new}\,\delta : \mathsf{rw}\langle\mathcal{I}'\rangle)\ \mathcal{C}^{\mathcal{I}}_{\alpha,n} \mid \delta?(X : (\mathcal{I}'))\ (C(s')\{\!\{X/v_{\mathcal{I}'}\}\!\})$$

where $\mathcal{I}'$ is $\mathcal{I}\ \mathit{after}\,\alpha$.

The $s$ derivation from $\mathcal{I};\Delta' \vdash Q$ ensures that $\mathcal{I} \vdash C(s)$. The fact that $\mathcal{I};\Delta \vdash P \Downarrow s$ ensures that $P$ **must** $C(s)$ since any stable state reachable from $C(s)$ must be successful. Finally the derivation from $\mathcal{I};\Delta' \vdash Q$ ending in the divergent $Q'$ ensures that $Q$ $\not$**must** $C(s)$.

The second possibility is that there is some $D \in \mathit{Acc}(\mathcal{I};\Delta' \vdash Q, s)$ which has no corresponding acceptance set in $\mathit{Acc}(\mathcal{I};\Delta \vdash P, s)$; here we can assume that both configurations $\mathcal{I};\Delta \vdash P$ and $\mathcal{I};\Delta' \vdash Q$ converge with respect to $s$.

Let $C_1, \ldots, C_n$ be all the acceptance sets in $\mathit{Acc}(\mathcal{I};\Delta \vdash P, s)$. Then we know that there is a set $\{c_1, \ldots c_n\}$ such that $c_i \in C_i - D$, for each $i$. Note $n$ may be zero but this will not affect our argument.

First let us construct a test from this set:

$$T_D = T(c_1) \mid \ldots \mid T(c_n)$$

where the tests $T(c_i)$ depend on the form of $c_i$:

- $c_i$ is an input $a?$: Here we know there is a derivation

  $$\mathcal{I};\Delta \vdash P \stackrel{s}{\Longrightarrow} (\mathcal{I}\ \mathit{after}\, s);\Delta' \vdash P'' \stackrel{a?v}{\longrightarrow}$$

  for some $v$. So let $T(c_i)$ be $a!\langle v\rangle\,\omega!\langle\rangle$.

- $c_i$ is an output $a!v$: Here we have

  $$\mathcal{I};\Delta \vdash P \stackrel{s}{\Longrightarrow} (\mathcal{I}\ \mathit{after}\, s);\Delta'' \vdash P' \stackrel{a!v}{\longrightarrow}$$

  So we let $T(c_i)$ be $a?(X : \mathrm{A})\,\omega!\langle\rangle$, where A denotes $(\mathcal{I}\ \mathit{after}\, s)^r(a)$, which by virtue of the last move we know exists.

Note we have constructed $T_D$ so that it can be typed by $(\mathcal{I}\ \mathit{after}\, s)$.

Now we construct the test $A(s, D)$ by induction on $s$, in the same manner as $C(s)$. The only difference is in the base case, when $s$ is the empty sequence, where $A(s, D)$ is defined to be $T_D$.

Again one can check that

- $\mathcal{I} \vdash A(s, D)$
- $Q \not\textbf{must} A(s, D)$ because of the derivation from $Q$ which gives rise to the acceptance set $D$
- but by construction $P$ **must** $A(s, D)$; note this holds even in the case when $Acc(\mathcal{I}; \Delta \vdash P, s)$ is empty.

□

## 6 Bisimulation

We now describe our characterisation of the co-inductively defined behavioural equivalence, $\approx^{\text{cxt}}_{\text{obs}}$, outlined in Section 3.2.

First we recall the definition of weak bisimulation from [13].

**Definition 6.1** *Given a labelled transition system $\mathcal{T}$, we say that a binary relation $\mathcal{R}$ on $\mathcal{T}$ is a* bisimulation *if whenever $n\ \mathcal{R}\ m$ then*

- *if $n \xrightarrow{\mu} n'$ then there exists a $m \stackrel{\hat{\mu}}{\Longrightarrow} m'$ such that $n'\ \mathcal{R}\ m'$*
- *if $m \xrightarrow{\mu} m'$ then there exists a $n \stackrel{\hat{\mu}}{\Longrightarrow} n'$ such that $n'\ \mathcal{R}\ m'$*

*where $\hat{\mu}$ is $\varepsilon$, the empty string, if $\mu$ is $\tau$ and $\mu$ otherwise.*

Our intention is to show that $\approx^{\text{cxt}}_{\text{obs}}$ can be characterised in terms of a bisimulation over **Conf**.

However as in Section 5 we have a mismatch between the formalisation of this relation, $\approx^{\text{cxt}}_{\text{obs}}$, in Section 3.2, which only uses one type environment, of the process being observed, and that of bisimulation equivalence, which uses two type environments. As with testing, we reconcile this difference by extending the definition of $\approx^{\text{cxt}}_{\text{obs}}$ so that it takes into account both environments.

First we generalise Definition 3.2 by now saying that an (extended) typed relation is a family $\mathcal{R}$ of relations over *typed processes*, parametrised, as before, by closed type environments, which satisfies: $(\Delta \vdash P)\ R_{\mathcal{I}}\ (\Delta' \vdash Q)$ implies $\mathcal{I}; \Delta \vdash P$ and $\mathcal{I}; \Delta \vdash Q$ are configurations. To conform to our previous notation we write this as

$$\mathcal{I} \models (\Delta \vdash P)\ \ R\ \ (\Delta' \vdash Q).$$

although effectively these are restricted forms of relations over configurations.

**Definition 6.2** *Let* (typed) bisimulation equivalence *be the largest typed relation $\approx$ which is*

- *a weak bisimulation*
- w-closed, *that is satisfying* $\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q)$ *implies* $\mathcal{I}, \Delta'' \models (\Delta, \Delta'' \vdash P)\ \mathcal{R}\ (\Delta', \Delta'' \vdash Q)$

*Bisimulation equivalence will be written as*

$$\mathcal{I} \models (\Delta \vdash P) \approx (\Delta' \vdash Q).$$

Note that the second requirement is required because we have already seen that $\eqsim_{\mathrm{obs}}^{\mathrm{cxt}}$ is *w-closed*. Intuitively its inclusion allows environments to pass *new* values to processes under investigation.

Two natural properties of (typed) bisimulation equivalence is given in the following proposition:

**Proposition 6.3** *Suppose* $\mathcal{I} \models (\Delta \vdash P) \approx (\Delta' \vdash Q)$. *Then*

- *for any appropriate* $\Delta''$, $\mathcal{I}, \Delta'' \models (\Delta, \Delta'' \vdash P) \approx (\Delta', \Delta'' \vdash Q)$.
- *If* $\mathcal{I} <: \mathcal{I}'$ *then* $\mathcal{I}' \models (\Delta \vdash P) \approx (\Delta' \vdash Q)$

**Proof:** The first result is simply a re-iteration of the fact that $\approx$ is *w-closed*. Intuitively the second property is true because $\mathcal{I}$ constrains the behaviour under which $P$ and $Q$ are compared. If they are equivalent under the constraint $\mathcal{I}$ then they should remain equivalent when they are constrained further, by $\mathcal{I}'$. To prove it formally let the family $\mathcal{R}$ be defined by

$$\mathcal{I}' \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q)$$

if $\mathcal{I} \models (\Delta \vdash P) \approx (\Delta' \vdash Q)$ for some $\mathcal{I} <: \mathcal{I}'$. This family is *w-closed* by definition, and it is straightforward to show that it is a bisimulation. It follows that $\mathcal{R} \subseteq \approx$, pointwise, from which the result follows. □

Let us now turn our attention to giving a similar formulation to $\eqsim_{\mathrm{obs}}^{\mathrm{cxt}}$, using two, rather than one, type environments. The definitions of **reduction closed** and **barb preserving** generalise immediately to extended typed relations. However that of being **contextual** is more complicated. Rather than giving a general definition based on arbitrary contexts we give a set of specific rules for our constructors; we say the extended relation $R$ is **contextual** if it satisfies the rules given in Figure 6. Finally let $\eqsim_{\mathrm{obs}}^{\mathrm{cxt}}$ be, as before, be the largest symmetric, reduction closed, barbed preserving contextual extended typed relation.

The requirements in Figure 6 are for the most part natural generalisations of the standard requirements for a relation to be preserved by constructors, generalised to take into account the type environments. Note

(CXT-SPEC)
$$\frac{\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q),\ \ \mathcal{I} <: \mathcal{I}'}{\mathcal{I}' \models (\Delta, \vdash P)\ \mathcal{R}\ (\Delta', \vdash Q)}$$

(CXT-WEAK)
$$\frac{\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q)}{\mathcal{I}, \Delta'' \models (\Delta, \Delta'' \vdash P)\ \mathcal{R}\ (\Delta', \Delta'' \vdash Q)}$$

(CXT-IN)
$$\frac{\mathcal{I} \vdash a : \mathsf{r}\langle \mathrm{A} \rangle \quad \mathcal{I}, \Delta'' \models (\Delta, \Delta'' \vdash T[v/X])\ \mathcal{R}\ (\Delta', \Delta'' \vdash U[v/X]),\ \text{ whenever } \mathcal{I}, \Delta'' \vdash v : \mathrm{A}}{\mathcal{I} \models (\Delta \vdash a?(X : \mathrm{A})\, T)\ \mathcal{R}\ (\Delta' \vdash a?(X : \mathrm{A})\, .U)}$$

(CXT-OUT)
$$\frac{\mathcal{I} \vdash u : \mathsf{w}\langle \mathrm{A} \rangle \quad \mathcal{I} \vdash v : \mathrm{A} \quad \mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q)}{\mathcal{I} \models (\Delta \vdash u!\langle v \rangle\, P)\ \mathcal{R}\ (\Delta' \vdash u!\langle v \rangle\, Q)}$$

(CXT-MATCH)
$$\frac{\Delta \vdash u : \mathrm{A}, v : \mathrm{A}' \qquad \Delta' \vdash u : \mathrm{B}, v : \mathrm{B}' \quad \mathcal{I} \models (\Delta \vdash P')\ \mathcal{R}\ (\Delta' \vdash Q') \quad \mathcal{I} \models (\Delta \sqcap \{u : \mathrm{A}', v : \mathrm{A}\} \vdash P)\ \mathcal{R}\ (\Delta' \sqcap \{u : \mathrm{B}', v : \mathrm{B}\} \vdash Q)}{\mathcal{I} \models (\Delta \vdash \mathsf{if}\ u = v\ \mathsf{then}\ P\ \mathsf{else}\ P')\ \mathcal{R}\ (\Delta' \vdash \mathsf{if}\ u = v\ \mathsf{then}\ Q\ \mathsf{else}\ Q')}$$

(CXT-NEW)
$$\frac{\mathcal{I}, a : \top \models (\Delta, a : \mathrm{A} \vdash P)\ \mathcal{R}\ (\Delta', a : \mathrm{A} \vdash Q)}{\mathcal{I} \models (\Delta \vdash (\mathsf{new}\, a : \mathrm{A})\ P)\ \mathcal{R}\ (\Delta' \vdash (\mathsf{new}\, a : \mathrm{A})\ Q)}$$

(CXT-PAR)
$$\frac{\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q) \quad \mathcal{I} \vdash R}{\mathcal{I} \models (\Delta \vdash P \mid R)\ \mathcal{R}\ (\Delta' \vdash Q \mid R)}$$
$$\mathcal{I} \models (\Delta \vdash R \mid P)\ \mathcal{R}\ (\Delta' \vdash R \mid Q)$$

(CXT-ITER)
$$\frac{\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q)}{\mathcal{I} \models (\Delta \vdash *P)\ \mathcal{R}\ (\Delta' \vdash *Q)}$$

FIGURE 6. Contextuality for indexed relations over configurations

however that the first two rules, (CXT-SPEC) and (CXT-WEAK), automatically build in *specialisation* and *weakening* properties, respectively. This may seem artificial but is justified by the following result, which shows that we do indeed have a generalisation of the definition of $\approx_{\mathrm{obs}}^{\mathrm{cxt}}$ from Section 3.2:

**Proposition 6.4** *$\Delta \models (\Delta \vdash P) \approx_{obs}^{cxt} (\Delta \vdash Q)$ if and only if $\Delta \models P \approx_{obs}^{cxt} Q$.*

**Proof:** We first show the *if* direction. Define a typed relation $\mathcal{R}$ by letting

$$\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta \vdash Q)$$

if $\Delta \models P \approx_{\mathrm{obs}}^{\mathrm{cxt}} Q$ and $\Delta <: \mathcal{I}$. $\mathcal{R}$ is symmetric, reduction closed and barb preserving. Using the fact that $\approx_{\mathrm{obs}}^{\mathrm{cxt}}$, as a family of relations over processes, is contextual, we can show that it satisfies all of the rules in Figure 6.

Therefore $\mathcal{R}$ is contained pointwise in $\approx_{\mathrm{obs}}^{\mathrm{cxt}}$, from which the result follows, since $\Delta \models P \approx_{\mathrm{obs}}^{\mathrm{cxt}} Q$ implies $\Delta \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta \vdash Q)$.

The converse is similar. Let the family of relations $\mathcal{R}$, over processes, be defined by

$$\Delta \models P\ \ \mathcal{R}\ \ Q \quad \text{if}\ \ \Delta \models (\Delta \vdash P) \approx_{\mathrm{obs}}^{\mathrm{cxt}} (\Delta \vdash Q).$$

Here the result will follow if we can show that $\mathcal{R}$ is contained pointwise in $\approx_{\mathrm{obs}}^{\mathrm{cxt}}$, which in turn will follow if we can show that $\mathcal{R}$ satisfies all the defining properties of $\approx_{\mathrm{obs}}^{\mathrm{cxt}}$. The proof that it is symmetric, reduction closed and barb preserving is straightforward.

It remains to show contextuality, that $\Gamma' \models T\ \mathcal{R}^o\ U$ and $\Gamma \vdash C[\cdot_{\Gamma'}]$ implies $\Gamma \models C[T]\ \mathcal{R}^o\ C[U]$. This is proved by induction on the derivation of $\Gamma \vdash C[\cdot_{\Gamma}]$, using the rules in Figure 6. Note that the rule (CXT-SPEC) is essential in the proof of the case in which the context is deduced using (T-NEW).

□

The remainder of this section is devoted to showing that this generalised contextual equivalence coincides with weak bisimulation on **Conf**; that is $\mathcal{I} \models (\Delta \vdash P) \approx_{\mathrm{obs}}^{\mathrm{cxt}} (\Delta' \vdash Q)$ if and only if $\mathcal{I} \models (\Delta \vdash P) \approx (\Delta' \vdash Q)$.

### *6.1 Soundness*

First let us show that typed bisimulation equivalence is preserved, in some appropriate manner, by the principal operators of the language.

**Proposition 6.5** *If* $\mathcal{I}, a : \top \models (\Delta, a : \mathrm{A} \vdash P) \approx (\Delta', a : \mathrm{A} \vdash Q)$ *then* $\mathcal{I} \models (\Delta \vdash (\mathsf{new}\, a : \mathrm{A})\ P) \approx (\Delta' \vdash (\mathsf{new}\, a : \mathrm{A})\ Q)$.

**Proof:** Let the relation $\mathcal{R}$ over typed processes be defined by

$$\mathcal{I} \models (\Delta \vdash R)\ \mathcal{R}\ (\Delta \vdash S)$$

if

- $\mathcal{I} \models (\Delta \vdash R) \approx (\Delta \vdash S)$
- or $R\ S$, have the form $(\mathsf{new}\, a : \mathrm{A})\ P$, $(\mathsf{new}\, a : \mathrm{A})\ Q$, respectively, and $\mathcal{I}, a : \top \models (\Delta, a : \mathrm{A} \vdash P) \approx (\Delta', a : \mathrm{A} \vdash Q)$.

  Then $\mathcal{R}$ is *w-closed* by definition. We show it is a bisimulation, from which the result will follow since we will have established that, pointwise, $\mathcal{R}$ is contained in $\approx$.

  We show how every possible move from $\mathcal{I}; \Delta \vdash R$ can be matched by one from $\mathcal{I}; \Delta \vdash S$. The only non-trivial cases are when $R$, $S$ have the second form above. From the definition of typed actions in Figure 5 there are two possibilities.

  1. The move is inferred using the rule (TYLTS-OPEN):

     $$\mathcal{I}; \Delta \vdash (\mathsf{new}\, a : \mathrm{A})\ P \xrightarrow{(a)\alpha} \mathcal{I}'; \Delta_\alpha \vdash P',$$

     for some output move $\alpha$, because

     $$\mathcal{I}, a : \top; \Delta, a : \mathrm{A} \vdash P \xrightarrow{\alpha} \mathcal{I}'; \Delta_\alpha \vdash P'.$$

     Here the latter move can be matched by

     $$\mathcal{I}, a : \top; \Delta', a : \mathrm{A} \vdash Q \stackrel{\alpha}{\Longrightarrow} \mathcal{I}'; \Delta'_\alpha \vdash Q'$$

     for some $Q'$ such that

     $$\mathcal{I}' \models (\Delta_\alpha, \vdash P) \approx (\Delta'_\alpha, \vdash Q),$$

     that is

     $$\mathcal{I}' \models (\Delta_\alpha \vdash P)\ \mathcal{R}\ (\Delta'_\alpha \vdash Q).$$

     However an application of the rule (TYLTS-OPEN), together with some reductions, gives the required matching move

     $$\mathcal{I}; \Delta' \vdash (\mathsf{new}\, a : \mathrm{A})\ Q \stackrel{(a)\alpha}{\Longrightarrow} \mathcal{I}'; \Delta'_\alpha \vdash Q'.$$

  2. The second possibility is that the move is inferred using the rule (TYLTS-CTXT):

     $$\mathcal{I}; \Delta \vdash (\mathsf{new}\, a : \mathrm{A})\ P \xrightarrow{\mu} \mathcal{I}'; \Delta_\alpha \vdash (\mathsf{new}\, a : \mathrm{A})\ P',$$

     because

     $$\mathcal{I}, a : \top; \Delta, a : \mathrm{A} \vdash P \xrightarrow{\mu} \mathcal{I}'; \Delta_\alpha \vdash P',$$

where $a \notin \mathsf{n}(\mu)$.

Here the proof is similar. We can find a matching move from $\mathcal{I}, a : \top; \Delta', a : \mathrm{A} \vdash Q$ and then use (TYLTS-CTXT) to obtain the required matching move from $\mathcal{I}; \Delta \vdash (\mathsf{new}\, a : \mathrm{A})\ Q$.

□

**Proposition 6.6** *Suppose* $\mathcal{I} \vdash R$. *Then* $\mathcal{I} \models (\Delta \vdash P) \approx (\Delta' \vdash Q)$ *implies* $\mathcal{I} \models (\Delta \vdash P \mid R) \approx (\Delta' \vdash Q \mid R)$.

**Proof:** Here, because of the possible internal communications between $R$ and $P$, $Q$, the required definition of the relation over typed processes is somewhat complicated.

Define the relation $\mathcal{R}$ such that

$$\mathcal{I} \models (\Delta \vdash (\mathsf{new}\, \Delta_0)\ P \mid R)\ \mathcal{R}\ (\Delta' \vdash (\mathsf{new}\, \Delta_0')\ Q \mid R)$$

if and only if there exists an $\mathcal{I}_0$ compatible with $\Delta_0$ and $\Delta_0'$ such that

$$\mathcal{I}, \mathcal{I}_0 \models (\Delta, \Delta_0 \vdash P) \approx (\Delta', \Delta_0' \vdash Q) \text{ and } \mathcal{I}, \mathcal{I}_0 \vdash R$$

and show that $\mathcal{R}$ forms a bisimulation.

Suppose then that

$$\mathcal{I} \models (\Delta \vdash (\mathsf{new}\, \Delta_0)\ P \mid R)\ \mathcal{R}\ (\Delta' \vdash (\mathsf{new}\, \Delta_0')\ Q \mid R)$$

and that

$$\mathcal{I}; \Delta \vdash P \mid R \xrightarrow{\mu} \mathcal{I}'; \Delta'' \vdash P'.$$

This presupposes the existence of an environment $\mathcal{I}_0$ compatible with both $\Delta_0$ and $\Delta_0'$ with the properties outlined in the definition of $\mathcal{R}$ . If $\mu$ is a not a $\tau$-action then we know that the transition derives either from $P$ or from $R$. In either case, we can use the hypothesis to obtain a corresponding transition from $Q$ or from $R$ again. So, the interesting case is when $\mu$ is a $\tau$ action. Consider how this can occur:

(i) $P$ or $R$ performs a $\tau$ action independently.

(ii) $P \xrightarrow{(\tilde{c}:\tilde{C})a!v} P_0$ and $R \xrightarrow{a?v} R'$ so that $P'$ is $(\mathsf{new}\, \Delta_0, \tilde{c} : \tilde{\mathrm{C}})\ P_0 \mid R'$ for some $\tilde{\mathrm{C}}$.

(iii) $P \xrightarrow{a?v} P_0$ and $R \xrightarrow{(\tilde{c}:\tilde{C})a!v} R'$ so that $P'$ is $(\mathsf{new}\, \Delta_0, \tilde{c} : \tilde{\mathrm{C}})\ P_0 \mid R'$

Obviously the first case (i) is treated as the case above when $\mu$ is not a $\tau$ action.

Suppose case (ii) holds. We know that $\mathcal{I}, \mathcal{I}_0 \vdash R$ so that, by Subject Reduction 2.2, $(\mathcal{I}, \mathcal{I}_0)^r(a) \downarrow$. Then, Lemma 4.3 tells us that

$$\mathcal{I}, \mathcal{I}_0; \Delta, \Delta_0 \vdash P \xrightarrow{(\tilde{c})a!v} \mathcal{I}, \mathcal{I}_0 \; \mathit{after}\,(\tilde{c})a!v; \Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash P_0.$$

We know by hypothesis that

$$\mathcal{I}, \mathcal{I}_0 \models (\Delta, \Delta_0 \vdash P) \approx (\Delta', \Delta_0' \vdash Q)$$

which means there must exist a transition from $Q$ which matches $P$'s output. That is, there is some $Q_0$ such that

$$\mathcal{I}, \mathcal{I}_0; \Delta', \Delta_0' \vdash Q \xRightarrow{(\tilde{c})a!v} \mathcal{I}, \mathcal{I}_0 \; \mathit{after}\,(\tilde{c})a!v; \Delta', \Delta_0', \tilde{c} : \tilde{\mathrm{C}}' \vdash Q_0$$

with

$$\mathcal{I}, \mathcal{I}_0 \; \mathit{after}\,(\tilde{c})a!v \models (\Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash P_0) \approx (\Delta', \Delta_0', \tilde{c} : \tilde{\mathrm{C}}' \vdash Q_0).$$

We also know that $R$ can interact with $Q$ to produce:

$$\mathcal{I}; \Delta' \vdash (\mathsf{new}\, \Delta_0') \;\; Q \mid R \Longrightarrow \mathcal{I}; \Delta' \vdash (\mathsf{new}\, \Delta_0', \tilde{c} : \tilde{\mathrm{C}}') \;\; Q_0 \mid R'$$

and observe that

$$\mathcal{I} \models (\Delta \vdash (\mathsf{new}\, \Delta_0, \tilde{c} : \tilde{\mathrm{C}}) \;\; P_0 \mid R') \; \mathcal{R} \; (\Delta' \vdash (\mathsf{new}\, \Delta_0', \tilde{c} : \tilde{\mathrm{C}}') \;\; Q_0 \mid R')$$

because $\mathcal{I}' \models (\Delta_0 \vdash P_0) \; \mathcal{R} \; (\Delta_0' \vdash Q_0)$ and $\mathcal{I}, \mathcal{I}_0 \; \mathit{after}\,(\tilde{c})a!v \vdash R'$. This last fact is guaranteed by the premise $\mathcal{I}, \mathcal{I}_0 \vdash R$ after applying Subject Reduction 2.2.

Finally, suppose (iii) holds. Again we have $\mathcal{I}, \mathcal{I}_0 \vdash R$ so that Subject Reduction, Theorem 2.2, tells us $(\mathcal{I}, \mathcal{I}_0)^w(a) \downarrow$ and, if we let $\mathcal{I}^+$ denote $\mathcal{I}, \mathcal{I}_0, \tilde{c} : \tilde{\mathrm{C}}$, we also have $\mathcal{I}^+ \vdash v : (\mathcal{I}^+)^w(a)$. This allows us to appeal to Lemma 4.3 to observe:

$$\mathcal{I}^+; \Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash P \xrightarrow{a?v} \mathcal{I}^+; \Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash P_0.$$

By hypothesis we know,

$$\mathcal{I}, \mathcal{I}_0 \models (\Delta, \Delta_0 \vdash P) \approx (\Delta', \Delta_0' \vdash Q)$$

so, because $\approx$ is $\omega$-closed we also have,

$$\mathcal{I}^+ \models (\Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash P) \approx (\Delta', \Delta_0', \tilde{c} : \tilde{\mathrm{C}} \vdash Q).$$

This guarantees our matching transition

$$\mathcal{I}^+; \Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash Q \xRightarrow{a?v} \mathcal{I}^+; \Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash Q_0$$

with $\mathcal{I}^+ \models (\Delta, \Delta_0, \tilde{c} : \tilde{\mathrm{C}} \vdash P_0) \approx (\Delta', \Delta_0', \tilde{c} : \tilde{\mathrm{C}} \vdash Q_0)$. We use the interaction with $R$ to obtain the reductions

$$\mathcal{I}; \Delta' \vdash (\mathsf{new}\, \Delta_0) \;\; Q \mid R \Longrightarrow \mathcal{I}; \Delta' \vdash (\mathsf{new}\, \Delta_0, \tilde{c} : \tilde{\mathrm{C}}) \;\; Q_0 \mid R'$$

and, again by Subject Reduction, Theorem 2.2, it is easy to see that $\mathcal{I}^+ \vdash R'$, whence

$$\mathcal{I} \models (\Delta \vdash (\mathsf{new}\,\Delta_0, \tilde{c} : \tilde{\mathrm{C}})\ P_0 \mid R')\ \mathcal{R}\ (\Delta' \vdash (\mathsf{new}\,\Delta'_0, \tilde{c} : \tilde{\mathrm{C}})\ Q_0 \mid R')$$

as required.

□

We now have most of the ingredients to prove:

**Theorem 6.7** *(Soundness)*

$$\textit{If } \mathcal{I} \models (\Delta \vdash P) \approx (\Delta' \vdash Q) \textit{ then } \mathcal{I} \models (\Delta \vdash P) \mathrel{\overset{cxt}{\approx}_{obs}} (\Delta' \vdash Q).$$

**Proof:** It is easy to see that $\approx$ is a reduction closed, symmetric and barb preserving relation over typed processes. If we can demonstrate that it is also contextual then, because of the the fact that $\overset{\text{cxt}}{\approx}_{\text{obs}}$ is the largest such relation we have our result. Therefore we only have to prove that $\approx$ satisfies all the rules in Figure 6.

The rules (CXT-SPEC) and (CXT-WEAK) are covered by Proposition 6.3, while (CXT-NEW) and (CXT-PAR) have just been established in the previous two Propositions. The remaining rules can be handled in a similar manner, by setting up an appropriate *w-closed* relation over typed processes and showing it is a bisimulation.

□

### *6.2 Completeness*

Here we show the converse of Theorem 6.7, *completeness*, namely that contextual equivalence implies bisimularity. To do so we only need a restricted version of contextual equivalence. Let $\overset{\text{p-cxt}}{\approx}_{\text{obs}}$ denote the largest relation over configurations which is reduction closed, barb preserving and contextual with respect to parallel and new name contexts, that is satisfies the rules (CXT-SPEC), (CXT-WEAK), (CXT-PAR) and (CXT-NEW) from Figure 6. It is clear that $\overset{\text{cxt}}{\approx}_{\text{obs}}$ implies $\overset{\text{p-cxt}}{\approx}_{\text{obs}}$ so, in fact, it suffices to prove completeness for the latter and we shall use this relation from now on.

Before we prove this theorem it will be useful to present a technical lemma. It is here that we utilize the exported names in the terms which witness the contextuality of labels. Essentially, the lemma states that the environment really can collate the information gained via the lts.

**Lemma 6.8** *Suppose $\mathcal{I}'$ is compatible with $\Delta, \tilde{c} : \tilde{\mathrm{C}}$ and $\Delta', \tilde{c} : \tilde{\mathrm{C}}$ and $\delta$ is fresh to $P, Q$. Then*

$$\mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \models (\Delta, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \vdash (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}})\ P \mid \delta!v_{\mathcal{I}'}) \\ \approx_{obs}^{p\text{-}cxt} (\Delta', \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \vdash (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}}')\ Q \mid \delta!v_{\mathcal{I}'})$$

*implies*

$$\mathcal{I}' \models (\Delta, \tilde{c} : \tilde{\mathrm{C}} \vdash P) \approx_{obs}^{p\text{-}cxt} (\Delta', \tilde{c} : \tilde{\mathrm{C}}' \vdash Q).$$

**Proof:** We prove this by co-induction. Let the relation $\mathcal{R}_{\mathcal{I}'}$ be defined for $\mathcal{I}'$ compatible with $\Delta, \Delta_0$ and $\Delta', \Delta_0'$, so that

$$\mathcal{I}' \models (\Delta, \Delta_0 \vdash (\mathsf{new}\, \Delta_1)\ P)\ \mathcal{R}\ (\Delta', \Delta_0 \vdash (\mathsf{new}\, \Delta_1')\ Q)$$

if and only if there is some $\delta : \mathsf{rw}\langle(\mathcal{I}')\rangle$ such that

$$\mathcal{I}, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \models (\Delta, \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \vdash (\mathsf{new}\, \Delta_0, \Delta_1)\ P \mid \delta!\langle v_{\mathcal{I}'}\rangle) \\ \approx_{\mathrm{obs}}^{\text{p-cxt}} (\Delta', \delta : \mathsf{rw}\langle(\mathcal{I}')\rangle \vdash (\mathsf{new}\, \Delta_0', \Delta_1')\ Q \mid \delta!\langle v_{\mathcal{I}'}\rangle).$$

We simply need to show that $\mathcal{R}$ is reduction closed, barb preserving, and closed with respect to rules (CXT-SPEC), (CXT-WEAK), (CXT-PAR) and (CXT-NEW). Reduction closure is immediate by the definition of $\mathcal{R}$, as is closure with respect to (CXT-SPEC) and (CXT-WEAK). For the other requirements we proceed by supposing that

$$\mathcal{I}' \models (\Delta, \Delta_0 \vdash P)\ \mathcal{R}\ (\Delta', \Delta_0' \vdash Q)$$

such that $\delta : \mathsf{rw}\langle(\mathcal{I}')\rangle$ with

$$\mathcal{I}, \delta \models (\Delta, \delta \vdash (\mathsf{new}\, \Delta_0)\ P \mid \delta!\langle v_{\mathcal{I}'}\rangle) \approx_{\mathrm{obs}}^{\text{p-cxt}} (\Delta', \delta \vdash (\mathsf{new}\, \Delta_0')\ Q \mid \delta!\langle v_{\mathcal{I}'}\rangle).$$

In the above equation, for the sake of presentation, we have omitted, and shall continue to do so for the remainder of this proof, to give the type information associated with the barb $\delta$.

We first show closure with respect to (CXT-PAR). Suppose $\mathcal{I}' \vdash R$. We need to show that $\mathcal{I}' \models (\Delta, \Delta_0 \vdash P \mid R)\ \mathcal{R}\ (\Delta', \Delta_0' \vdash Q \mid R)$. To do this we choose some fresh $\delta'$ and construct $R' = \delta?(X : \mathcal{I}')\ (R[X/\mathsf{n}(\mathcal{I}')] \mid \delta'!\langle\rangle)$ (recall that $\mathsf{n}(\Gamma)$ refers to the names in the domain of $\Gamma$). It should be evident that $\delta, \delta' \vdash R'$ and, by closure of $\approx_{\mathrm{obs}}^{\text{p-cxt}}$ with respect to (CXT-SPEC), (CXT-WEAK), (CXT-NEW) and (CXT-PAR) we have

$$\mathcal{I}, \delta' \models (\Delta, \delta' \vdash (\mathsf{new}\, \Delta_0, \delta)\ P \mid \delta!\langle v_{\mathcal{I}'}\rangle \mid R') \approx_{\mathrm{obs}}^{\text{p-cxt}} \\ (\Delta', \delta' \vdash (\mathsf{new}\, \Delta_0', \delta)\ Q \mid \delta!\langle v_{\mathcal{I}'}\rangle \mid R')$$

It is fairly easy to check that

$$\mathcal{I}', \delta' \models (\Delta, \delta' \vdash (\mathsf{new}\, \Delta_0, \delta)\ (P \mid \delta!\langle v_{\mathcal{I}'}\rangle \mid R')) \approx_{\mathrm{obs}}^{\text{p-cxt}} \\ (\Delta', \delta' \vdash (\mathsf{new}\, \Delta_0)\ (P \mid R \mid \delta'!\langle v_{\mathcal{I}'}\rangle))$$

and similarly for $Q$. Hence,

$$\mathcal{I}, \delta' \models (\Delta, \delta' \vdash (\mathsf{new}\,\Delta_0)\ P|R|\delta'!\langle v_{\mathcal{I}'}\rangle) \mathrel{\overline{\approx}}^{\text{p-cxt}}_{\text{obs}} (\Delta', \delta' \vdash (\mathsf{new}\,\Delta'_0)\ Q|R|\delta'!\langle v_{\mathcal{I}'}\rangle)$$

This serves to witness

$$\mathcal{I}' \models (\Delta, \Delta_0 \vdash P \mid R)\ \mathcal{R}\ (\Delta', \Delta'_0 \vdash Q \mid R)$$

as required.

The closure of $\mathcal{R}$ with respect to (CXT-NEW) follows easily from the closure of $\mathrel{\overline{\approx}}^{\text{p-cxt}}_{\text{obs}}$ with respect to this rule. So we will finish by showing that $\mathcal{R}$ is barb preserving.

We suppose that $\mathcal{I}'; \Delta, \Delta_0 \vdash P \Downarrow^{\text{barb}} a$ for some $a$ such that $\mathcal{I}' \vdash a : \mathsf{rw}\langle A\rangle$. Choose a fresh $\delta' : \mathsf{rw}\langle\top\rangle$ and build $R = \delta?(X : \mathcal{I}')\ X_a?(Y : A)\,\delta'!\langle\rangle$ where $X_a$ refers to the component of $X$ to which the name $a$ will become bound to as it receives the value $v_{\mathcal{I}'}$. We know that $\delta, \delta' \vdash R$,

$$\mathcal{I}, \delta, \delta' \models (\Delta \vdash (\mathsf{new}\,\Delta_0)\ P \,|\, \delta!\langle v_{\mathcal{I}'}\rangle \,|\, R) \mathrel{\overline{\approx}}^{\text{p-cxt}}_{\text{obs}} (\Delta' \vdash (\mathsf{new}\,\Delta'_0)\ Q \,|\, \delta! v_{\mathcal{I}'} \,|\, R)$$

and

$$\mathcal{I}, \delta, \delta'; \Delta \vdash (\mathsf{new}\,\Delta_0)\ P \mid \delta!\langle v_{\mathcal{I}'}\rangle \mid R \Downarrow^{\text{barb}} \delta'.$$

This means that, by the barb preservation property of $\mathrel{\overline{\approx}}^{\text{p-cxt}}_{\text{obs}}$, we know

$$\mathcal{I}, \delta, \delta' :; \Delta' \vdash (\mathsf{new}\,\Delta'_0)\ Q \mid \delta!\langle v_{\mathcal{I}'}\rangle \mid R \Downarrow^{\text{barb}} \delta'$$

also. But, as $\delta'$ is fresh, this could only have arisen by interaction with $R$ along $a$, whence

$$\mathcal{I}'; \Delta', \Delta'_0 \vdash Q \Downarrow^{\text{barb}} a$$

as required. □

We can state the central theorem which allows us to achieve completeness:

**Theorem 6.9** *(Completeness)*

$$\textit{If } \mathcal{I} \models (\Delta \vdash P) \mathrel{\overline{\approx}}^{cxt}_{obs} (\Delta' \vdash Q) \textit{ then } \mathcal{I} \models (\Delta \vdash P) \approx (\Delta' \vdash Q).$$

**Proof:** Again, the proof proceeds by co-induction, this time we define the relation $\mathcal{R}$ by letting $\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q)$ if $\mathcal{I} \models (\Delta \vdash P) \mathrel{\overline{\approx}}^{\text{p-cxt}}_{\text{obs}} (\Delta' \vdash Q)$. By definition it is *w-closed*. We demonstrate that it forms a bisimulation. To this end, suppose $\mathcal{I} \models (\Delta \vdash P)\ \mathcal{R}\ (\Delta' \vdash Q)$ and that $\mathcal{I}; \Delta \vdash P \xrightarrow{\mu} \mathcal{I}'; \Delta_0 \vdash P'$. We use the contextuality of labels to find a matching transition and proceed by cases on $\mu$. We only show the case for $\mu$ is $(\tilde{c})a!v$ here, it being the most involved. Note that, in this case, $\Delta_0$ is $\Delta, \tilde{c} : \tilde{\mathbb{C}}$.

We choose a fresh $\delta : \mathrm{A}_\delta$ where $\mathrm{A}_\delta$ denotes $\mathsf{rw}\langle(\mathcal{I}')\rangle$, and use Proposition 4.6 to find a term such that $\mathcal{I}, \delta : \mathrm{A}_\delta \vdash \mathcal{C}_\mu^{\mathcal{I}}$ with the appropriate properties. In fact, the first property tells us that

$$\mathcal{I}, \delta : \mathrm{A}_\delta; \Delta, \delta : \mathrm{A}_\delta \vdash P \,|\, \mathcal{C}_\mu^{\mathcal{I}} \Longrightarrow \mathcal{I}, \delta : \mathrm{A}_\delta; \Delta, \delta : \mathrm{A}_\delta \vdash (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}})\ (P' \,|\, \delta!\langle v_{\mathcal{I}'}\rangle)$$

Using $\mathcal{C}_\mu^{\mathcal{I}}$ we can build a test term by choosing further fresh names $\delta' : \mathrm{A}_\delta,\ a : \mathsf{rw}\langle\top\rangle$ and letting

$$\mathcal{C}_{\delta'} = a!\langle\rangle \mid \delta?(x)\ \ a?(y)\,.\delta'!\langle x\rangle$$

we note immediately that $\mathcal{C}_{\delta'} \Downarrow^{\mathrm{barb}} a$.

From contextual closure (omitting some type information) we know that

$$\mathcal{I}, \delta' \models (\Delta, \delta' \vdash (\mathsf{new}\, \delta)\ (P \,|\, \mathcal{C}_\mu^{\mathcal{I}} \,|\, \mathcal{C}_{\delta'})) \mathbin{\overset{\text{p-cxt}}{\approx}_{\mathrm{obs}}} (\Delta', \delta' \vdash (\mathsf{new}\, \delta)\ (Q \,|\, \mathcal{C}_\mu^{\mathcal{I}} \,|\, \mathcal{C}_{\delta'}))$$

We also know that the left hand side of this equation may reduce (up to a minor structural equivalence) to

$$\mathcal{I}, \delta'; \Delta, \delta' \vdash (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}})\ \ P' \mid \delta'!\langle v_{\mathcal{I}'}\rangle\,.$$

We use $C_P$ to refer to this configuration and observe that $C_P \not\Downarrow^{\mathrm{barb}} a$ but $C_P \Downarrow^{\mathrm{barb}} \delta'$.

Reduction closure now tells us that there must exist some matching reductions

$$\mathcal{I}, \delta'; \Delta', \delta' \vdash (\mathsf{new}\, \delta)\ (Q \,|\, \mathcal{C}_\mu^{\mathcal{I}} \,|\, \mathcal{C}_{\delta'}) \Longrightarrow C_Q$$

for some $C_Q$ such that $C_P \mathbin{\overset{\text{p-cxt}}{\approx}_{\mathrm{obs}}} C_Q$.

Now $\mathbin{\overset{\text{p-cxt}}{\approx}_{\mathrm{obs}}}$ preserves barbs, so this means, in particular, that $C_Q \not\Downarrow^{\mathrm{barb}} a$ and $C_Q \Downarrow^{\mathrm{barb}} \delta'$ also. Hence we know that $C_Q$ must be (again up to a minor structural equivalence) of a very specific form:

$$\mathcal{I}, \delta'; \Delta', \delta' \vdash (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}}')\ \ Q' \mid \delta'\langle !v_0\rangle)$$

for some $Q'$ and some $v_0$. By the construction of $\mathcal{C}_{\delta'}$ and the fact that $C_Q \not\Downarrow^{\mathrm{barb}} a$ we know that the reductions to $C_Q$ must have been generated by interaction with reductions of the form

$$(\mathcal{I}, \delta; \Delta', \delta \vdash Q \,|\, \mathcal{C}_\mu^{\mathcal{I}}) \Longrightarrow (\mathcal{I}, \delta; \Delta', \delta \vdash (\mathsf{new}\, \tilde{c} : \tilde{\mathrm{C}}')\ \ Q' \mid \delta!\langle v_0\rangle$$

which, by Proposition 4.6, must themselves have been generated by interaction with transitions of the form

$$\mathcal{I}; \Delta' \vdash Q \overset{\mu}{\Longrightarrow} \mathcal{I}'; \Delta', \tilde{c} : \tilde{\mathrm{C}}' \vdash Q'.$$

It only remains to demonstrate that

$$\mathcal{I}' \models (\Delta, \tilde{c} : \tilde{\mathrm{C}} \vdash P')\ \mathcal{R}\ (\Delta', \tilde{c} : \tilde{\mathrm{C}}' \vdash Q').$$

This follows from the fact that $C_P \mathbin{\overset{\text{p-cxt}}{\approx}_{\mathrm{obs}}} C_Q$ and Lemma 6.8. □

Soundness, Completeness and Proposition 6.4 allows us to now conclude with the main result of the paper:

**Corollary 6.10** *If* $\Gamma \models P \mathrel{\overline{\approx}}^{cxt}_{obs} Q$ *if and only if* $\Gamma \models (\Gamma \vdash P) \approx^{o} (\Gamma \vdash Q)$.

### *6.3 Example*

The characterisation of the previous sections provide a convenient coinductive method for establishing contextual observational equivalence between terms. We provide a short example which demonstrates the utility of the bisimulation proof method. The processes that we consider provide two different implementations of a producer/consumer unit server.

Clients send requests for service along a global channel $req$, which must be accompanied by a reply channel which has type at least $\mathrm{R} = \mathsf{w}\langle(\mathsf{w}\langle\top\rangle, \mathsf{r}\langle\top\rangle)\rangle$. The server creates dedicated produce and consume channels, exclusively for the client, at type $\mathrm{A} = \mathsf{rw}\langle\top\rangle$, and returns these along the reply channel. Note that because of the type of the return channel the client only receives the write capability on the produce channel and the read capability on the consume channel. The server then manages the simple protocol that for every call on the produce channel, a corresponding request can be made of the consume channel:

$$CU_1 = *req?(x : \mathrm{R})\,(\mathsf{new}\, p, c : \mathrm{A})\;\; x!\langle p, c\rangle \;\;\; * p?()\,.c!\langle\rangle$$

Here the server and uses the process $*p?()\,.c!\langle\rangle$ to manage the produce and consume requests.

Another implementation is given by:

$$CU_2 = *req?(x : \mathrm{R})\,(\mathsf{new}\, p, c : \mathrm{A})\;\; x!(p, c).\;\; (*p?()\,.c!\langle\rangle \mid *c?()\,.p!\langle\rangle).$$

The behaviour of this server, when managing the produce/consume requests is a little different. Here the server itself, in addition to the client, may consume a request; if it does so it then reproduces a message in recompense.

There is a bug in the second implementation because, having set up a protocol for a client, when a message is sent by the client it may be consumed by the server itself, consequently unleashing an infinite sequence of produce/consume messages internal to the server. This can be formally demonstrated using the *must* testing equivalence.

Let $\Delta^{<}_{req}$ be any typing environment such that $\Delta^{<}_{req}(req) = \mathsf{rw}\langle\mathsf{rw}\langle(\mathrm{B}, \mathrm{C})\rangle\rangle$, where $\mathrm{B} <: \mathsf{w}\langle\top\rangle$. Then

$$\Delta^{<}_{req} \models CU_1 \not\approx_{must} CU_2.$$

To prove this result we exhibit a test $T$ such that $\Delta^{<}_{req} \vdash T$, $CU_1$ **must** $T$

but $CU_1 \not\mathbf{must}\ T$. The required $T$ is given by

$$(\mathsf{new}\, r : \mathrm{A}_r)\ req!\langle r\rangle\ r?((x,y) : (\mathsf{w}\langle\top\rangle, \mathsf{r}\langle\top\rangle))\ x!\langle\rangle\, \omega!\langle\rangle$$

where $\mathrm{A}_r$ is the type $\mathsf{rw}\langle(\mathrm{B},\mathrm{C})\rangle$. It is straightforward to show that this can be typed by $\Delta_{req}$, that it is guaranteed by $CU_1$ but when applied to $CU_2$, may lead to a non-terminating computation.

It is well-known that contextual observational equivalence is insensitive to such internal divergent behaviour. However there are further reasons for these two servers to exhibit different behaviour, in certain type environments. Suppose for example that $\Delta_d$ maps the channel $req$ to the type $\mathsf{rw}\langle\mathsf{rw}\langle(\mathrm{B},\mathrm{C})\rangle\rangle$, where $B, C$ are $\mathsf{rw}\langle\top\rangle$, $\mathsf{r}\langle\top\rangle$ respectively. This type enables the environment, when it receives a produce/consume pair $p, c$, to both write *and* read on $p$; with this capability $CU_1$ and $CU_2$ can be distinguished. For example consider the context $C[\,]$:

$$T \mid [\,]$$

where $T$ is the process

$$(\mathsf{new}\, r : \mathrm{A}_r)\ req!\langle r\rangle\ r?((x,y) : (\mathsf{rw}\langle\top\rangle, \mathsf{r}\langle\top\rangle))\ x!\langle\rangle\, x?()\, d!\langle\rangle\,.$$

Then it is easy to see, assuming $\Delta_d$ has the appropriate type for $d$, that $\Delta \models C[CU_2] \Downarrow^{\mathrm{barb}} d$ whereas $\Delta \models C[CU_1] \not\Downarrow^{\mathrm{barb}} d$. It follows that

$$\Delta_d \models CU_1 \not\approx^{\mathrm{cxt}}_{\mathrm{obs}} CU_2.$$

Note that a similar example can be constructed if, instead, we allow the environment write access on the consume channel $c$.

However if we limit the environment's access to the produce/consume pair to be write, read respectively then we can show that the two servers are contextually equivalent:

**Proposition 6.11** *Suppose that $\Delta_{req}$ is any typing environment such that*

$$\Delta_{req}(req) = \mathsf{rw}\langle\mathsf{rw}\langle(\mathrm{B},\mathrm{C})\rangle\rangle,$$

*where* $\mathrm{B} :> \mathsf{w}\langle\top\rangle$ *and* $\mathrm{C} :> \mathsf{r}\langle\top\rangle$. *Then*

$$\Delta_{req} \models CU_1 \approx^{cxt}_{obs} CU_2.$$

**Proof:** We will establish that these processes are in fact bisimilar and then by soundness we may conclude that they are contextual observational equivalent. To show that they are bisimilar we may appeal to congruence properties of bisimulation equivalence and show that, for $\Delta_A = p : A, c : A$,

$$P_1 = *p?().c!\langle\,\rangle \quad \text{and} \quad P_2 = *p?().c!\langle\,\rangle \mid *c?().p!\langle\,\rangle$$

and for any $\mathcal{I} :> p : \mathrm{B}, c : \mathrm{C}$, we have

$$\mathcal{I} \models (\Delta_A \vdash P_1) \approx (\Delta_A \vdash P_2).$$

To demonstrate this we define our candidate relation for the bisimulation as follows. Let $c!^n$ for a non-negative integer $n$ mean the $n$-fold parallel composition of the terms $c!\langle\,\rangle$. We relate terms such that (upto $\equiv$)

$$\mathcal{I} \models (\Delta_A \vdash P_1 \mid c!^n) \;\mathcal{R}\; (\Delta_A \vdash P_2 \mid c!^m \mid p!^k)$$

whenever $n = m + k$. We must show that the $\omega$-closure of $\mathcal{R}$ forms a bisimulation.

This is reasonably straightforward, as an example of the work involved (ignoring $\omega$-closures) we suppose that

$$\mathcal{I} \models (\Delta_A \vdash P_1 \mid c!^n) \;\mathcal{R}\; (\Delta_A \vdash P_2 \mid c!^m \mid p!^k)$$

and that $\mathcal{I}; \Delta_A \vdash P_1 \mid c!^n \xrightarrow{\mu} \mathcal{I}; \Delta_A \vdash P'$ (note that $\mathcal{I}$ and $\Delta_A$ will not change throughout this proof and $n = m + k$).

We consider how this could arise: firstly, if the transition originated in $P_1$ then we know $\mu$ must be of the form $p?()$. This is only possible in case the environment $\mathcal{I}$ has write capability on channel $p$, which it may. There is an obvious matching transition from $P_2$ which is always ready to receive produce requests on $p$ also. The resulting states reached are in $\mathcal{R}$ :

$$\mathcal{I} \models (\Delta_A \vdash P_1 \mid c!^{n+1}) \;\mathcal{R}\; (\Delta_A \vdash P_2 \mid c!^{m+1} \mid p!^k)$$

as $(n + 1) = (m + 1) + k$.

The second possibility is that the transition from originates from $c!^n$ and is an output of the form $c!()$ (that is $n > 0$ and $P'$ is $P_1 \mid c!^{n-1}$). This can only be possible if the environment $\mathcal{I}$ has read capability on this channel, which it may. The matching transition for this would of course simply be an output $c!()$ from $P_2 \mid c!^m \mid p!^k$, but we do not necessarily know that $m > 0$. However, we do know that $n = m + k > 0$ so, in the case $m = 0$ we must have $k > 0$ and an internal communication between $p!^k$ and $P_2$ ensures the availability of a $c!()$ transition. Thus, the matching transitions are, for $m > 0$,

$$(\mathcal{I}; \Delta_A \vdash P_2 \mid c!^m \mid p!^k) \xrightarrow{c!\langle\rangle} (\mathcal{I}; \Delta_A \vdash P_2 \mid c!^{m-1} \mid p!^k),$$

and for $m = 0$,

$$(\mathcal{I}; \Delta_A \vdash P_2 \mid c!^m \mid p!^k) \xrightarrow{\tau} \xrightarrow{c!\langle\rangle} (\mathcal{I}; \Delta_A \vdash P_2 \mid c!^m \mid p!^{k-1}).$$

In either case we end up back in $\mathcal{R}$ .

Similar arguments can be used to match transitions from $P_2 \mid c!^m \mid p!^k$. The only extra transitions here arise as internal communications. There

are two possibilities for these: an interaction between $P_2$ and $c!^m$ and an interaction between $P_2$ and $p!^k$. Note that in each case the resulting state is

$$P_2 \mid c!^{m-1} \mid p!^{k+1}$$

for the former and

$$P_2 \mid c!^{m+1} \mid p!^{k-1}$$

for the latter. In either case the total $m + k$ is invariant. This means that the extra internal transitions exhibited by $P_2 \mid c!^m \mid p!^k$ may be matched in $\mathcal{R}$ by an empty transition from $P_1 \mid c!^n$.

It is worth mentioning here that it is not possible to observe output transitions of the form $p!\langle\rangle$ from $\mathcal{I}; \Delta_A \vdash P_2 \mid c!^m \mid p!^k$ as we have supposed that $\mathcal{I}(p) :> \mathrm{B} :> \mathsf{w}\langle\top\rangle$ and thus cannot the read capability required to make this observation. Similarly, it is not possible to observe input transitions of the form $c?\langle\rangle$ from $\mathcal{I}; \Delta_A \vdash P_2$.

□

This short example demonstrates the use of a co-inductive proof for establishing contextual observational equivalence. The use of the bisimulation method allows us to establish equivalence without quantifying over all possible clients for these servers. In effect, the environment plays the role of an arbitrary client.

## 7 Conclusion

In this paper we have studied typed behavioural equivalences for the $\pi$-calculus. In particular we have shown that natural typed versions testing and barbed congruences can be captured by applying standard techniques to a new lts of *typed actions*, **Conf**. Thus, at least in principle, it should be possible to use, or adapt, existing proof methodologies and verification systems, [4, 5] to prove type dependent equivalences between processes. Admittedly the *states*, $\mathcal{I}; \Delta \vdash P$, in the lts are a priori complicated, consisting of a process term $P$, a type environment for its computing context $\mathcal{I}$ and a separate type environment for the process itself $\Delta$. But the observant reader will have noticed that in the rules for generated **Conf**, in Figure 5, the last type environment $\Delta$ plays no role. Technically its presence has been convenient for deriving our results, which depend on the fact that processes are well-typed with respect to some environment coherent with $\mathcal{I}$, but in an implementation of **Conf** they could be safely omitted.

Typed process equivalences, as opposed to untyped ones, have nu-

merous interesting applications. For example such an equivalence has been used in [18] to investigate translations of the $\lambda$-calculus into the $\pi$-calculus; the use of types enables stronger results to be demonstrated. In [24] complier optimisations are justified using a typed equivalence, for a language similar to ours. We also intend to develop typed equivalences for the higher-order process language in [25, 22], where types are used to resources and computing hosts from malicious agents; in such scenarios demonstrating that a particular policy does indeed offer host protection would amount to proving typed equivalences.

We have based our notion of contextual equivalence on the approach of [9, 7] rather than that used, for example in [15]. In the latter the behavioural equivalence itself is not required to be itself contextual, but instead the largest contextual relation contained in it is the focus of study. The two approaches are conceptually not very different and in many cases they actually generate the same behavioural relation; see for example [9, 6]. However proofs characterising the latter in terms of bisimulation relations are often complex, dependent on the precise constructs of the language being investigated, and sometimes even require infinitary syntactic constructs; see for example the characterisation proofs in [15, 23]. Indeed more recent characterisation theorems, such as that in [1] tend to be restricted to *finite-branching processes*. On the other hand characterisation theorems for the kind of formalisation we have chosen are usually conceptually more simple, or at least not very dependent on the precise language constructs to hand; see for example the proofs in [9, 11].

There have been some previous attempts at characterising typed contextual equivalences [2, 17] using an observer's view of the type environment. In [17] the observer's view is used to account for contextual equivalences in the presence of polymorphic types in the $\pi$-calculus but subtyping is not considered. Also no complete characterisation of barbed congruence was provided in this setting. Although this paper did not deal with subtyping the subsequent work in [2] did tackle this problem albeit in a theoretically very different setting to ours. The chief difference lays in the absence of the name equality test. We crucially use the name equality operator to witness that each of our actions $\alpha$ may be constructed as a test $C_\alpha[\ \ ]$ and that the type information gained during this test may be accumulated. In [2] the lack of the equality test impacts upon the labelled transition system in the sense that names being sent to the observer cannot be recognised as names encountered during a prior test. In particular this precludes accumulation of knowledge about a name and the labelled transition system presented in this paper reflects this. A co-inductive characterisation of barbed congruence is provided although the

equations holding in this setting vary considerably from ours. For instance the well-known Replication Theorem of $\pi$-calculus used to illustrate their technique fails to hold in the presence of equality testing.

Our system allows for a gradual increase in knowledge about types of names and provides a fresh approach to understanding the effects of subtyping on process equivalence.

## References

[1] Roberto M. Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous $\pi$-calculus. *Theoretical Computer Science*, 195(2):291–324, 30 March 1998.

[2] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *Proc. 13th LICS Conf.* IEEE Computer Society Press, 1998.

[3] G. Boudol. Typing the use of resources in a concurrent calculus. In *Proceedings of the ASIAN'97*, number 1345 in Lecture Notes in Computer Science, pages 239–253, 1997.

[4] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics based verification tool for finite state systems. *ACM Transactions on Programming Systems*, 15:36–72, 1989.

[5] Rance Cleaveland. The concurrency factory: A development environment for concurrent systems. In R. Alur and T. Henzinger, editors, *Proceedings of CAV'96*, volume 1102 of *Lecture Notes in Computer Science*, pages 398–401. Springer-Verlag, 1988.

[6] C. Fournet and G.Gonthier. A hierarchy of equivalences for asynchronous calculi (extended abstract). In *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 844–855. Springer-Verlag, 1988.

[7] C. Fournet, G. Gonthier, J.J. Levy, L. Marganget, and D. Remy. A calculus of mobile agents. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 406–421, Pisa, August 1996. Springer Verlag.

[8] M. Hennessy. *An Algebraic Theory of Processes.* MIT Press, 1988.

[9] Kohei Honda and Mario Tokoro. On asynchronous communication semantics. In P. Wegner M. Tokoro, O. Nierstrasz, editor, *Proceedings of the ECOOP '91 Workshop on Object-Based Concurrent Computing*, volume 612 of *LNCS 612.* Springer-Verlag, 1992.

[10] A. Jeffrey. A distributed object calculus. In *Proc. ACM Foundations of Object Oriented Languages.* IEEE Computer Society Press, 2000.

[11] A. Jeffrey and J. Rathke. A theory of bisimulation for a fragment of concurrent ml with local names. In *Proc. LICS2000,* $15^{th}$ *Annual Symposium on Logic in Computer Science, Santa Barbara*, pages 311–321. IEEE Computer Society Press, 2000.

[12] Naoki Kobayashi. A partially deadlock-free typed process calculus. In *Proceedings, Twelth Annual IEEE Symposium on Logic in Computer Science*, pages 128–139, Warsaw, Poland, 29 June–2 July 1997. IEEE Computer Society Press.

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

[14] R. Milner. *Comunicating and mobile systems: the $\pi$-calculus.* Cambridge Univer-

sity Press, 1999.

[15] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proc. 19th ICALP*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer-Verlag, 1992.

[16] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 24:83–113, 1984.

[17] B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. In *Proc. 24th POPL*. ACM Press, 1997. Full paper to appear in Journal of the ACM.

[18] Benjamin Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996. Extended abstract in LICS '93.

[19] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University, 1997. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, Gordon Plotkin, Colin Stirling, and Mads Tofte, editors, MIT Press.

[20] R. Pugliese R. DeNicola, G. Ferrari. Klaim: a kernel language for agents interaction and mobility. In *IEEE Transactions on Software Engineering*, number 5 in 24, pages 315–330. IEEE Computer Society, 1998.

[21] James Riely and Matthew Hennessy. Resource access control in systems of mobile agents (extended abstract). In *Proceedings of 3rd International Workshop on High-Level Concurrent Languages*, Nice, France, September 1998. Full version available as Computer Science Technical Report 2/98, University of Sussex, 1997. Available from `http://www.cogs.susx.ac.uk/` To be published in Information and Computation.

[22] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents (extended abstract). In *Conference Record of POPL '99 The 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 93–104, 1999.

[23] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.

[24] Akinori Yonnezawa, Motoki Nakade, and Naoki Kobayashi. Static analysis of communication for asynchronous concurrent programming languages. In A. Mycroft, editor, *Static Analysis. Proceedings*, volume 983 of *Lecture Notes in Computer Science*, pages 225–242. Springer-Verlag, 1995.

[25] N. Yoshida and M. Hennessy. Assigning types to processes (extended abstract). In *Proceedings, Fifteenth Annual IEEE Symposium on Logic in Computer Science*, pages 334–348, Santa Barbara, US, 19–23 June 2000. IEEE Computer Society Press.